

## Introduction

This reference manual targets application developers. It provides complete information on how to use the STM32F75xxx and STM32F74xxx microcontroller memory and peripherals.

The STM32F75xxx and STM32F74xxx is a family of microcontrollers with different memory sizes, packages and peripherals.

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

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

STM32F75xxx and STM32F74xxx microcontrollers include ST state-of-the-art patented technology.

## Related documents

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

- STM32F74xxx and STM32F756xx datasheets
- STM32F750x8 datasheet
- STM32F74xx and STM32F75xx device errata sheet (ES0290)
- *STM32F7 Series Cortex<sup>®</sup>-M7 processor programming manual* (PM0253)

# Contents

<b>1</b>	<b>Documentation conventions</b>	<b>63</b>
1.1	General information	63
1.2	List of abbreviations for registers	63
1.3	Register reset value	63
1.4	Glossary	64
1.5	Availability of peripherals	64
<b>2</b>	<b>System and memory overview</b>	<b>65</b>
2.1	System architecture	65
2.1.1	Multi AHB BusMatrix	66
2.1.2	AHB/APB bridges (APB)	66
2.1.3	CPU AXIM bus	67
2.1.4	ITCM bus	67
2.1.5	DTCM bus	67
2.1.6	CPU AHBS bus	67
2.1.7	AHB peripheral bus	67
2.1.8	DMA memory bus	68
2.1.9	DMA peripheral bus	68
2.1.10	Ethernet DMA bus	68
2.1.11	USB OTG HS DMA bus	68
2.1.12	LCD-TFT controller DMA bus	68
2.1.13	DMA2D bus	68
2.2	Memory organization	69
2.2.1	Introduction	69
2.2.2	Memory map and register boundary addresses	70
2.3	Embedded SRAM	74
2.4	Flash memory overview	74
2.5	Boot configuration	75
<b>3</b>	<b>Embedded Flash memory (FLASH)</b>	<b>77</b>
3.1	Introduction	77
3.2	Flash main features	77
3.3	Flash functional description	78

3.3.1	Flash memory organization .....	78
3.3.2	Read access latency .....	79
3.3.3	Flash program and erase operations .....	81
3.3.4	Unlocking the Flash control register .....	82
3.3.5	Maximum program/erase parallelism .....	82
3.3.6	Flash erase sequences .....	83
3.3.7	Flash programming sequences .....	83
3.3.8	Flash Interrupts .....	84
3.4	FLASH Option bytes .....	85
3.4.1	Option bytes description .....	85
3.4.2	Option bytes programming .....	88
3.5	FLASH memory protection .....	89
3.5.1	Read protection (RDP) .....	89
3.5.2	Write protections .....	90
3.6	One-time programmable bytes .....	91
3.7	FLASH registers .....	92
3.7.1	Flash access control register (FLASH_ACR) .....	92
3.7.2	Flash key register (FLASH_KEYR) .....	93
3.7.3	Flash option key register (FLASH_OPTKEYR) .....	93
3.7.4	Flash status register (FLASH_SR) .....	94
3.7.5	Flash control register (FLASH_CR) .....	95
3.7.6	Flash option control register (FLASH_OPTCR) .....	96
3.7.7	Flash option control register (FLASH_OPTCR1) .....	97
3.7.8	Flash interface register map .....	99
<b>4</b>	<b>Power controller (PWR) .....</b>	<b>100</b>
4.1	Power supplies .....	100
4.1.1	Independent A/D converter supply and reference voltage .....	101
4.1.2	Independent USB transceivers supply .....	102
4.1.3	Battery backup domain .....	103
4.1.4	Voltage regulator .....	105
4.2	Power supply supervisor .....	108
4.2.1	Power-on reset (POR)/power-down reset (PDR) .....	108
4.2.2	Brownout reset (BOR) .....	109
4.2.3	Programmable voltage detector (PWD) .....	109
4.3	Low-power modes .....	110

---

4.3.1	Debug mode .....	114
4.3.2	Run mode .....	114
4.3.3	Low-power mode .....	114
4.3.4	Sleep mode .....	115
4.3.5	Stop mode .....	117
4.3.6	Standby mode .....	120
4.3.7	Programming the RTC alternate functions to wake up the device from the Stop and Standby modes .....	121
4.4	Power control registers .....	124
4.4.1	PWR power control register (PWR_CR1) .....	124
4.4.2	PWR power control/status register (PWR_CSR1) .....	126
4.4.3	PWR power control/status register 2 (PWR_CR2) .....	128
4.4.4	PWR power control register 2 (PWR_CSR2) .....	130
4.5	PWR register map .....	132
<b>5</b>	<b>Reset and clock control (RCC) .....</b>	<b>133</b>
5.1	Reset .....	133
5.1.1	System reset .....	133
5.1.2	Power reset .....	133
5.1.3	Backup domain reset .....	134
5.2	Clocks .....	134
5.2.1	HSE clock .....	138
5.2.2	HSI clock .....	139
5.2.3	PLL .....	140
5.2.4	LSE clock .....	141
5.2.5	LSI clock .....	141
5.2.6	System clock (SYSCLK) selection .....	141
5.2.7	Clock security system (CSS) .....	141
5.2.8	RTC/AWU clock .....	142
5.2.9	Watchdog clock .....	143
5.2.10	Clock-out capability .....	143
5.2.11	Internal/external clock measurement using TIM5/TIM11 .....	143
5.2.12	Peripheral clock enable register (RCC_AHBxENR, RCC_APBxENRy) .....	145
5.3	RCC registers .....	145
5.3.1	RCC clock control register (RCC_CR) .....	145
5.3.2	RCC PLL configuration register (RCC_PLLCFGR) .....	148

5.3.3	RCC clock configuration register (RCC_CFGR) . . . . .	150
5.3.4	RCC clock interrupt register (RCC_CIR) . . . . .	152
5.3.5	RCC AHB1 peripheral reset register (RCC_AHB1RSTR) . . . . .	155
5.3.6	RCC AHB2 peripheral reset register (RCC_AHB2RSTR) . . . . .	158
5.3.7	RCC AHB3 peripheral reset register (RCC_AHB3RSTR) . . . . .	159
5.3.8	RCC APB1 peripheral reset register (RCC_APB1RSTR) . . . . .	159
5.3.9	RCC APB2 peripheral reset register (RCC_APB2RSTR) . . . . .	163
5.3.10	RCC AHB1 peripheral clock register (RCC_AHB1ENR) . . . . .	165
5.3.11	RCC AHB2 peripheral clock enable register (RCC_AHB2ENR) . . . . .	167
5.3.12	RCC AHB3 peripheral clock enable register (RCC_AHB3ENR) . . . . .	168
5.3.13	RCC APB1 peripheral clock enable register (RCC_APB1ENR) . . . . .	168
5.3.14	RCC APB2 peripheral clock enable register (RCC_APB2ENR) . . . . .	172
5.3.15	RCC AHB1 peripheral clock enable in low-power mode register (RCC_AHB1LPENR) . . . . .	175
5.3.16	RCC AHB2 peripheral clock enable in low-power mode register (RCC_AHB2LPENR) . . . . .	177
5.3.17	RCC AHB3 peripheral clock enable in low-power mode register (RCC_AHB3LPENR) . . . . .	178
5.3.18	RCC APB1 peripheral clock enable in low-power mode register (RCC_APB1LPENR) . . . . .	179
5.3.19	RCC APB2 peripheral clock enabled in low-power mode register (RCC_APB2LPENR) . . . . .	183
5.3.20	RCC backup domain control register (RCC_BDCR) . . . . .	185
5.3.21	RCC clock control & status register (RCC_CSR) . . . . .	186
5.3.22	RCC spread spectrum clock generation register (RCC_SSCGR) . . . . .	188
5.3.23	RCC PLLI2S configuration register (RCC_PLLI2SCFGR) . . . . .	189
5.3.24	RCC PLLSAI configuration register (RCC_PLLSAICFGR) . . . . .	192
5.3.25	RCC dedicated clocks configuration register (RCC_DCKCFGR1) . . . . .	193
5.3.26	RCC dedicated clocks configuration register (DCKCFGR2) . . . . .	195
5.3.27	RCC register map . . . . .	198
<b>6</b>	<b>General-purpose I/Os (GPIO) . . . . .</b>	<b>201</b>
6.1	Introduction . . . . .	201
6.2	GPIO main features . . . . .	201
6.3	GPIO functional description . . . . .	201
6.3.1	General-purpose I/O (GPIO) . . . . .	204
6.3.2	I/O pin alternate function multiplexer and mapping . . . . .	204
6.3.3	I/O port control registers . . . . .	205

---

6.3.4	I/O port data registers . . . . .	205
6.3.5	I/O data bitwise handling . . . . .	205
6.3.6	GPIO locking mechanism . . . . .	206
6.3.7	I/O alternate function input/output . . . . .	206
6.3.8	External interrupt/wake-up lines . . . . .	206
6.3.9	Input configuration . . . . .	207
6.3.10	Output configuration . . . . .	207
6.3.11	Alternate function configuration . . . . .	208
6.3.12	Analog configuration . . . . .	209
6.3.13	Using the HSE or LSE oscillator pins as GPIOs . . . . .	210
6.3.14	Using the GPIO pins in the backup supply domain . . . . .	210
6.4	GPIO registers . . . . .	211
6.4.1	GPIO port mode register (GPIO <sub>x</sub> _MODER) (x = A to K) . . . . .	211
6.4.2	GPIO port output type register (GPIO <sub>x</sub> _OTYPER) (x = A to K) . . . . .	211
6.4.3	GPIO port output speed register (GPIO <sub>x</sub> _OSPEEDR) (x = A to K) . . . . .	212
6.4.4	GPIO port pull-up/pull-down register (GPIO <sub>x</sub> _PUPDR) (x = A to K) . . . . .	212
6.4.5	GPIO port input data register (GPIO <sub>x</sub> _IDR) (x = A to K) . . . . .	213
6.4.6	GPIO port output data register (GPIO <sub>x</sub> _ODR) (x = A to K) . . . . .	213
6.4.7	GPIO port bit set/reset register (GPIO <sub>x</sub> _BSRR) (x = A to K) . . . . .	213
6.4.8	GPIO port configuration lock register (GPIO <sub>x</sub> _LCKR) (x = A to K) . . . . .	214
6.4.9	GPIO alternate function low register (GPIO <sub>x</sub> _AFRL) (x = A to K) . . . . .	215
6.4.10	GPIO alternate function high register (GPIO <sub>x</sub> _AFRH) (x = A to J) . . . . .	216
6.4.11	GPIO register map . . . . .	218
7	<b>System configuration controller (SYSCFG) . . . . .</b>	<b>220</b>
7.1	I/O compensation cell . . . . .	220
7.2	SYSCFG registers . . . . .	220
7.2.1	SYSCFG memory remap register (SYSCFG_MEMRMP) . . . . .	220
7.2.2	SYSCFG peripheral mode configuration register (SYSCFG_PMC) . . . . .	221

7.2.3	SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1) .....	222
7.2.4	SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2) .....	222
7.2.5	SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3) .....	223
7.2.6	SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4) .....	224
7.2.7	Compensation cell control register (SYSCFG_CMPCR) .....	224
7.2.8	SYSCFG register maps .....	225
<b>8</b>	<b>Direct memory access controller (DMA) .....</b>	<b>226</b>
8.1	DMA introduction .....	226
8.2	DMA main features .....	226
8.3	DMA functional description .....	228
8.3.1	DMA block diagram .....	228
8.3.2	DMA overview .....	228
8.3.3	DMA transactions .....	229
8.3.4	Channel selection .....	229
8.3.5	Arbiter .....	231
8.3.6	DMA streams .....	231
8.3.7	Source, destination and transfer modes .....	231
8.3.8	Pointer incrementation .....	235
8.3.9	Circular mode .....	236
8.3.10	Double-buffer mode .....	236
8.3.11	Programmable data width, packing/unpacking, endianness .....	237
8.3.12	Single and burst transfers .....	238
8.3.13	FIFO .....	239
8.3.14	DMA transfer completion .....	242
8.3.15	DMA transfer suspension .....	243
8.3.16	Flow controller .....	244
8.3.17	Summary of the possible DMA configurations .....	245
8.3.18	Stream configuration procedure .....	245
8.3.19	Error management .....	246
8.4	DMA interrupts .....	247
8.5	DMA registers .....	248
8.5.1	DMA low interrupt status register (DMA_LISR) .....	248
8.5.2	DMA high interrupt status register (DMA_HISR) .....	249

---

8.5.3	DMA low interrupt flag clear register (DMA_LIFCR) . . . . .	250
8.5.4	DMA high interrupt flag clear register (DMA_HIFCR) . . . . .	250
8.5.5	DMA stream x configuration register (DMA_SxCR) . . . . .	251
8.5.6	DMA stream x number of data register (DMA_SxNDTR) . . . . .	254
8.5.7	DMA stream x peripheral address register (DMA_SxPAR) . . . . .	255
8.5.8	DMA stream x memory 0 address register (DMA_SxM0AR) . . . . .	255
8.5.9	DMA stream x memory 1 address register (DMA_SxM1AR) . . . . .	255
8.5.10	DMA stream x FIFO control register (DMA_SxFCR) . . . . .	256
8.5.11	DMA register map . . . . .	257
<b>9</b>	<b>Chrom-ART Accelerator controller (DMA2D) . . . . .</b>	<b>261</b>
9.1	DMA2D introduction . . . . .	261
9.2	DMA2D main features . . . . .	261
9.3	DMA2D functional description . . . . .	262
9.3.1	DMA2D block diagram . . . . .	262
9.3.2	DMA2D control . . . . .	263
9.3.3	DMA2D foreground and background FIFOs . . . . .	263
9.3.4	DMA2D foreground and background pixel format converter (PFC) . . . . .	263
9.3.5	DMA2D foreground and background CLUT interface . . . . .	265
9.3.6	DMA2D blender . . . . .	266
9.3.7	DMA2D output PFC . . . . .	267
9.3.8	DMA2D output FIFO . . . . .	267
9.3.9	DMA2D AHB master port timer . . . . .	267
9.3.10	DMA2D transactions . . . . .	268
9.3.11	DMA2D configuration . . . . .	268
9.3.12	DMA2D transfer control (start, suspend, abort and completion) . . . . .	271
9.3.13	Watermark . . . . .	271
9.3.14	Error management . . . . .	271
9.3.15	AHB dead time . . . . .	271
9.4	DMA2D interrupts . . . . .	272
9.5	DMA2D registers . . . . .	272
9.5.1	DMA2D control register (DMA2D_CR) . . . . .	272
9.5.2	DMA2D interrupt status register (DMA2D_ISR) . . . . .	274
9.5.3	DMA2D interrupt flag clear register (DMA2D_IFCR) . . . . .	274
9.5.4	DMA2D foreground memory address register (DMA2D_FGMAR) . . . . .	275

9.5.5	DMA2D foreground offset register (DMA2D_FGOR) . . . . .	275
9.5.6	DMA2D background memory address register (DMA2D_BGMAR) . .	276
9.5.7	DMA2D background offset register (DMA2D_BGOR) . . . . .	276
9.5.8	DMA2D foreground PFC control register (DMA2D_FGPFCCR) . . .	277
9.5.9	DMA2D foreground color register (DMA2D_FGCOLR) . . . . .	278
9.5.10	DMA2D background PFC control register (DMA2D_BGPFCCR) . . .	279
9.5.11	DMA2D background color register (DMA2D_BGCOLR) . . . . .	280
9.5.12	DMA2D foreground CLUT memory address register (DMA2D_FGCMAR) . . . . .	281
9.5.13	DMA2D background CLUT memory address register (DMA2D_BGCMAR) . . . . .	281
9.5.14	DMA2D output PFC control register (DMA2D_OPFCCR) . . . . .	282
9.5.15	DMA2D output color register (DMA2D_OCOLR) . . . . .	282
9.5.16	DMA2D output color register [alternate] (DMA2D_OCOLR) . . .	283
9.5.17	DMA2D output color register [alternate] (DMA2D_OCOLR) . . .	283
9.5.18	DMA2D output color register [alternate] (DMA2D_OCOLR) . . .	284
9.5.19	DMA2D output memory address register (DMA2D_OMAR) . . . . .	284
9.5.20	DMA2D output offset register (DMA2D_OOR) . . . . .	285
9.5.21	DMA2D number of line register (DMA2D_NLR) . . . . .	285
9.5.22	DMA2D line watermark register (DMA2D_LWR) . . . . .	286
9.5.23	DMA2D AHB master timer configuration register (DMA2D_AMTCR) .	286
9.5.24	DMA2D foreground CLUT (DMA2D_FGCLUTx) . . . . .	287
9.5.25	DMA2D background CLUT (DMA2D_BGCLUTx) . . . . .	287
9.5.26	DMA2D register map . . . . .	288
<b>10</b>	<b>Nested vectored interrupt controller (NVIC) . . . . .</b>	<b>290</b>
10.1	NVIC features . . . . .	290
10.1.1	SysTick calibration value register . . . . .	290
10.1.2	Interrupt and exception vectors . . . . .	290
<b>11</b>	<b>Extended interrupts and events controller (EXTI) . . . . .</b>	<b>295</b>
11.1	EXTI main features . . . . .	295
11.2	EXTI block diagram . . . . .	295
11.3	Wakeup event management . . . . .	296
11.4	Functional description . . . . .	296
11.5	Hardware interrupt selection . . . . .	296
11.6	Hardware event selection . . . . .	296

---

11.7	Software interrupt/event selection . . . . .	297
11.8	External interrupt/event line mapping . . . . .	297
11.9	EXTI registers . . . . .	298
11.9.1	Interrupt mask register (EXTI_IMR) . . . . .	298
11.9.2	Event mask register (EXTI_EMR) . . . . .	298
11.9.3	Rising trigger selection register (EXTI_RTSR) . . . . .	299
11.9.4	Falling trigger selection register (EXTI_FTSR) . . . . .	299
11.9.5	Software interrupt event register (EXTI_SWIER) . . . . .	300
11.9.6	Pending register (EXTI_PR) . . . . .	300
11.9.7	EXTI register map . . . . .	301
<b>12</b>	<b>Cyclic redundancy check calculation unit (CRC) . . . . .</b>	<b>302</b>
12.1	Introduction . . . . .	302
12.2	CRC main features . . . . .	302
12.3	CRC functional description . . . . .	303
12.3.1	CRC block diagram . . . . .	303
12.3.2	CRC internal signals . . . . .	303
12.3.3	CRC operation . . . . .	303
12.4	CRC registers . . . . .	305
12.4.1	CRC data register (CRC_DR) . . . . .	305
12.4.2	CRC independent data register (CRC_IDR) . . . . .	305
12.4.3	CRC control register (CRC_CR) . . . . .	306
12.4.4	CRC initial value (CRC_INIT) . . . . .	307
12.4.5	CRC polynomial (CRC_POL) . . . . .	307
12.4.6	CRC register map . . . . .	308
<b>13</b>	<b>Flexible memory controller (FMC) . . . . .</b>	<b>309</b>
13.1	Introduction . . . . .	309
13.2	FMC main features . . . . .	309
13.3	FMC block diagram . . . . .	310
13.4	AHB interface . . . . .	311
13.4.1	Supported memories and transactions . . . . .	311
13.5	External device address mapping . . . . .	312
13.5.1	NOR/PSRAM address mapping . . . . .	313
13.5.2	NAND flash memory address mapping . . . . .	314
13.5.3	SDRAM address mapping . . . . .	315

13.6	NOR flash/PSRAM controller . . . . .	318
13.6.1	External memory interface signals . . . . .	319
13.6.2	Supported memories and transactions . . . . .	321
13.6.3	General timing rules . . . . .	323
13.6.4	NOR flash/PSRAM controller asynchronous transactions . . . . .	323
13.6.5	Synchronous transactions . . . . .	340
13.6.6	NOR/PSRAM controller registers . . . . .	347
13.7	NAND flash controller . . . . .	354
13.7.1	External memory interface signals . . . . .	354
13.7.2	NAND flash supported memories and transactions . . . . .	355
13.7.3	Timing diagrams for NAND flash memory . . . . .	356
13.7.4	NAND flash operations . . . . .	357
13.7.5	NAND flash prewait functionality . . . . .	357
13.7.6	Computation of the error correction code (ECC) in NAND flash memory . . . . .	358
13.7.7	NAND flash controller registers . . . . .	359
13.8	SDRAM controller . . . . .	365
13.8.1	SDRAM controller main features . . . . .	365
13.8.2	SDRAM External memory interface signals . . . . .	365
13.8.3	SDRAM controller functional description . . . . .	366
13.8.4	Low-power modes . . . . .	372
13.8.5	SDRAM controller registers . . . . .	376
13.8.6	FMC register map . . . . .	382
<b>14</b>	<b>Quad-SPI interface (QUADSPI) . . . . .</b>	<b>385</b>
14.1	Introduction . . . . .	385
14.2	QUADSPI main features . . . . .	385
14.3	QUADSPI functional description . . . . .	385
14.3.1	QUADSPI block diagram . . . . .	385
14.3.2	QUADSPI pins . . . . .	386
14.3.3	QUADSPI command sequence . . . . .	386
14.3.4	QUADSPI signal interface protocol modes . . . . .	389
14.3.5	QUADSPI indirect mode . . . . .	391
14.3.6	QUADSPI automatic status-polling mode . . . . .	393
14.3.7	QUADSPI memory-mapped mode . . . . .	393
14.3.8	QUADSPI flash memory configuration . . . . .	394
14.3.9	QUADSPI delayed data sampling . . . . .	394

---

14.3.10	QUADSPI configuration . . . . .	394
14.3.11	QUADSPI use . . . . .	395
14.3.12	Sending the instruction only once . . . . .	397
14.3.13	QUADSPI error management . . . . .	397
14.3.14	QUADSPI busy bit and abort functionality . . . . .	397
14.3.15	NCS behavior . . . . .	398
14.4	QUADSPI interrupts . . . . .	400
14.5	QUADSPI registers . . . . .	400
14.5.1	QUADSPI control register (QUADSPI_CR) . . . . .	400
14.5.2	QUADSPI device configuration register (QUADSPI_DCR) . . . . .	403
14.5.3	QUADSPI status register (QUADSPI_SR) . . . . .	404
14.5.4	QUADSPI flag clear register (QUADSPI_FCR) . . . . .	405
14.5.5	QUADSPI data length register (QUADSPI_DLR) . . . . .	405
14.5.6	QUADSPI communication configuration register (QUADSPI_CCR) . . . . .	406
14.5.7	QUADSPI address register (QUADSPI_AR) . . . . .	408
14.5.8	QUADSPI alternate-byte register (QUADSPI_ABR) . . . . .	408
14.5.9	QUADSPI data register (QUADSPI_DR) . . . . .	409
14.5.10	QUADSPI polling status mask register (QUADSPI_PSMKR) . . . . .	409
14.5.11	QUADSPI polling status match register (QUADSPI_PSMAR) . . . . .	410
14.5.12	QUADSPI polling interval register (QUADSPI_PIR) . . . . .	410
14.5.13	QUADSPI low-power timeout register (QUADSPI_LPTR) . . . . .	411
14.5.14	QUADSPI register map . . . . .	411
<b>15</b>	<b>Analog-to-digital converter (ADC) . . . . .</b>	<b>413</b>
15.1	ADC introduction . . . . .	413
15.2	ADC main features . . . . .	413
15.3	ADC functional description . . . . .	414
15.3.1	ADC on-off control . . . . .	415
15.3.2	ADC1/2 and ADC3 connectivity . . . . .	416
15.3.3	ADC clock . . . . .	419
15.3.4	Channel selection . . . . .	419
15.3.5	Single conversion mode . . . . .	420
15.3.6	Continuous conversion mode . . . . .	420
15.3.7	Timing diagram . . . . .	420
15.3.8	Analog watchdog . . . . .	421
15.3.9	Scan mode . . . . .	422
15.3.10	Injected channel management . . . . .	422

15.3.11	Discontinuous mode .....	423
15.4	Data alignment .....	424
15.5	Channel-wise programmable sampling time .....	425
15.6	Conversion on external trigger and trigger polarity .....	426
15.7	Fast conversion mode .....	427
15.8	Data management .....	428
15.8.1	Using the DMA .....	428
15.8.2	Managing a sequence of conversions without using the DMA .....	428
15.8.3	Conversions without DMA and without overrun detection .....	429
15.9	Multi ADC mode .....	429
15.9.1	Injected simultaneous mode .....	432
15.9.2	Regular simultaneous mode .....	433
15.9.3	Interleaved mode .....	434
15.9.4	Alternate trigger mode .....	436
15.9.5	Combined regular/injected simultaneous mode .....	438
15.9.6	Combined regular simultaneous + alternate trigger mode .....	438
15.10	Temperature sensor .....	439
15.11	Battery charge monitoring .....	441
15.12	ADC interrupts .....	442
15.13	ADC registers .....	443
15.13.1	ADC status register (ADC_SR) .....	443
15.13.2	ADC control register 1 (ADC_CR1) .....	444
15.13.3	ADC control register 2 (ADC_CR2) .....	446
15.13.4	ADC sample time register 1 (ADC_SMPR1) .....	447
15.13.5	ADC sample time register 2 (ADC_SMPR2) .....	448
15.13.6	ADC injected channel data offset register x (ADC_JOFR <sub>x</sub> ) ( $x=1..4$ ) ..	448
15.13.7	ADC watchdog higher threshold register (ADC_HTR) .....	449
15.13.8	ADC watchdog lower threshold register (ADC_LTR) .....	449
15.13.9	ADC regular sequence register 1 (ADC_SQR1) .....	450
15.13.10	ADC regular sequence register 2 (ADC_SQR2) .....	450
15.13.11	ADC regular sequence register 3 (ADC_SQR3) .....	451
15.13.12	ADC injected sequence register (ADC_JSQR) .....	452
15.13.13	ADC injected data register x (ADC_JDR <sub>x</sub> ) ( $x=1..4$ ) .....	452
15.13.14	ADC regular data register (ADC_DR) .....	453
15.13.15	ADC Common status register (ADC_CSR) .....	453
15.13.16	ADC common control register (ADC_CCR) .....	454

---

15.13.17	ADC common regular data register for dual and triple modes (ADC_CDR) . . . . .	457
15.13.18	ADC register map . . . . .	457
<b>16</b>	<b>Digital-to-analog converter (DAC) . . . . .</b>	<b>460</b>
16.1	DAC introduction . . . . .	460
16.2	DAC main features . . . . .	460
16.3	DAC functional description . . . . .	461
16.3.1	DAC channel enable . . . . .	461
16.3.2	DAC output buffer enable . . . . .	462
16.3.3	DAC data format . . . . .	462
16.3.4	DAC conversion . . . . .	463
16.3.5	DAC output voltage . . . . .	464
16.3.6	DAC trigger selection . . . . .	464
16.3.7	DMA request . . . . .	465
16.3.8	Noise generation . . . . .	465
16.3.9	Triangle-wave generation . . . . .	466
16.4	Dual DAC channel conversion . . . . .	467
16.4.1	Independent trigger without wave generation . . . . .	468
16.4.2	Independent trigger with single LFSR generation . . . . .	468
16.4.3	Independent trigger with different LFSR generation . . . . .	468
16.4.4	Independent trigger with single triangle generation . . . . .	469
16.4.5	Independent trigger with different triangle generation . . . . .	469
16.4.6	Simultaneous software start . . . . .	469
16.4.7	Simultaneous trigger without wave generation . . . . .	470
16.4.8	Simultaneous trigger with single LFSR generation . . . . .	470
16.4.9	Simultaneous trigger with different LFSR generation . . . . .	470
16.4.10	Simultaneous trigger with single triangle generation . . . . .	471
16.4.11	Simultaneous trigger with different triangle generation . . . . .	471
16.5	DAC registers . . . . .	472
16.5.1	DAC control register (DAC_CR) . . . . .	472
16.5.2	DAC software trigger register (DAC_SWTRIGR) . . . . .	475
16.5.3	DAC channel1 12-bit right-aligned data holding register (DAC_DHR12R1) . . . . .	475
16.5.4	DAC channel1 12-bit left aligned data holding register (DAC_DHR12L1) . . . . .	476
16.5.5	DAC channel1 8-bit right aligned data holding register (DAC_DHR8R1) . . . . .	476

16.5.6	DAC channel2 12-bit right aligned data holding register (DAC_DHR12R2) . . . . .	477
16.5.7	DAC channel2 12-bit left aligned data holding register (DAC_DHR12L2) . . . . .	477
16.5.8	DAC channel2 8-bit right-aligned data holding register (DAC_DHR8R2) . . . . .	477
16.5.9	Dual DAC 12-bit right-aligned data holding register (DAC_DHR12RD) . . . . .	478
16.5.10	DUAL DAC 12-bit left aligned data holding register (DAC_DHR12LD) . . . . .	478
16.5.11	DUAL DAC 8-bit right aligned data holding register (DAC_DHR8RD) . . . . .	479
16.5.12	DAC channel1 data output register (DAC_DOR1) . . . . .	479
16.5.13	DAC channel2 data output register (DAC_DOR2) . . . . .	479
16.5.14	DAC status register (DAC_SR) . . . . .	480
16.5.15	DAC register map . . . . .	481
<b>17</b>	<b>Digital camera interface (DCMI) . . . . .</b>	<b>482</b>
17.1	Introduction . . . . .	482
17.2	DCMI main features . . . . .	482
17.3	DCMI functional description . . . . .	482
17.3.1	DCMI block diagram . . . . .	483
17.3.2	DCMI pins . . . . .	483
17.3.3	DCMI clocks . . . . .	483
17.3.4	DCMI DMA interface . . . . .	484
17.3.5	DCMI physical interface . . . . .	484
17.3.6	DCMI synchronization . . . . .	486
17.3.7	DCMI capture modes . . . . .	488
17.3.8	DCMI crop feature . . . . .	489
17.3.9	DCMI JPEG format . . . . .	490
17.3.10	DCMI FIFO . . . . .	490
17.3.11	DCMI data format description . . . . .	491
17.4	DCMI interrupts . . . . .	493
17.5	DCMI registers . . . . .	493
17.5.1	DCMI control register (DCMI_CR) . . . . .	493
17.5.2	DCMI status register (DCMI_SR) . . . . .	496
17.5.3	DCMI raw interrupt status register (DCMI_RIS) . . . . .	496
17.5.4	DCMI interrupt enable register (DCMI_IER) . . . . .	497

---

17.5.5	DCMI masked interrupt status register (DCMI_MIS) . . . . .	498
17.5.6	DCMI interrupt clear register (DCMI_ICR) . . . . .	499
17.5.7	DCMI embedded synchronization code register (DCMI_ESCR) . . . . .	500
17.5.8	DCMI embedded synchronization unmask register (DCMI_ESUR) . . . . .	500
17.5.9	DCMI crop window start (DCMI_CWSTRT) . . . . .	501
17.5.10	DCMI crop window size (DCMI_CWSIZE) . . . . .	502
17.5.11	DCMI data register (DCMI_DR) . . . . .	502
17.5.12	DCMI register map . . . . .	503
<b>18</b>	<b>LCD-TFT display controller (LTDC) . . . . .</b>	<b>504</b>
18.1	Introduction . . . . .	504
18.2	LTDC main features . . . . .	504
18.3	LTDC functional description . . . . .	505
18.3.1	LTDC block diagram . . . . .	505
18.3.2	LTDC pins and external signal interface . . . . .	505
18.3.3	LTDC reset and clocks . . . . .	506
18.4	LTDC programmable parameters . . . . .	507
18.4.1	LTDC global configuration parameters . . . . .	507
18.4.2	Layer programmable parameters . . . . .	510
18.5	LTDC interrupts . . . . .	515
18.6	LTDC programming procedure . . . . .	515
18.7	LTDC registers . . . . .	516
18.7.1	LTDC synchronization size configuration register (LTDC_SSCR) . . . . .	516
18.7.2	LTDC back porch configuration register (LTDC_BPCR) . . . . .	518
18.7.3	LTDC active width configuration register (LTDC_AWCR) . . . . .	519
18.7.4	LTDC total width configuration register (LTDC_TWCR) . . . . .	519
18.7.5	LTDC global control register (LTDC_GCR) . . . . .	520
18.7.6	LTDC shadow reload configuration register (LTDC_SRCR) . . . . .	521
18.7.7	LTDC background color configuration register (LTDC_BCCR) . . . . .	522
18.7.8	LTDC interrupt enable register (LTDC_IER) . . . . .	522
18.7.9	LTDC interrupt status register (LTDC_ISR) . . . . .	523
18.7.10	LTDC Interrupt clear register (LTDC_ICR) . . . . .	524
18.7.11	LTDC line interrupt position configuration register (LTDC_LIPCR) . . . . .	524
18.7.12	LTDC current position status register (LTDC_CPSR) . . . . .	525
18.7.13	LTDC current display status register (LTDC_CDSR) . . . . .	525
18.7.14	LTDC layer x control register (LTDC_LxCR) . . . . .	526

18.7.15	LTDC layer x window horizontal position configuration register (LTDC_LxWHPCR) .....	527
18.7.16	LTDC layer x window vertical position configuration register (LTDC_LxWVPCR) .....	528
18.7.17	LTDC layer x color keying configuration register (LTDC_LxCKCR) .....	529
18.7.18	LTDC layer x pixel format configuration register (LTDC_LxPFCR) .....	529
18.7.19	LTDC layer x constant alpha configuration register (LTDC_LxCACR) .....	530
18.7.20	LTDC layer x default color configuration register (LTDC_LxDCCR) .....	530
18.7.21	LTDC layer x blending factors configuration register (LTDC_LxBFCR) .....	531
18.7.22	LTDC layer x color frame buffer address register (LTDC_LxCFBAR) .....	532
18.7.23	LTDC layer x color frame buffer length register (LTDC_LxCFBLR) .....	532
18.7.24	LTDC layer x color frame buffer line number register (LTDC_LxCFBLNR) .....	533
18.7.25	LTDC layer x CLUT write register (LTDC_LxCLUTWR) .....	533
18.7.26	LTDC register map .....	534
<b>19</b>	<b>True random number generator (RNG) .....</b>	<b>537</b>
19.1	Introduction .....	537
19.2	RNG main features .....	537
19.3	RNG functional description .....	538
19.3.1	RNG block diagram .....	538
19.3.2	RNG internal signals .....	538
19.3.3	Random number generation .....	539
19.3.4	RNG initialization .....	541
19.3.5	RNG operation .....	541
19.3.6	RNG clocking .....	542
19.3.7	Error management .....	542
19.3.8	RNG low-power use .....	543
19.4	RNG interrupts .....	543
19.5	RNG processing time .....	543
19.6	RNG entropy source validation .....	543
19.6.1	Introduction .....	543

---

19.6.2	Validation conditions . . . . .	544
19.6.3	Data collection . . . . .	544
19.7	RNG registers . . . . .	544
19.7.1	RNG control register (RNG_CR) . . . . .	544
19.7.2	RNG status register (RNG_SR) . . . . .	545
19.7.3	RNG data register (RNG_DR) . . . . .	546
19.7.4	RNG register map . . . . .	547
<b>20</b>	<b>Cryptographic processor (CRYP) . . . . .</b>	<b>548</b>
20.1	Introduction . . . . .	548
20.2	CRYP main features . . . . .	548
20.3	CRYP implementation . . . . .	549
20.4	CRYP functional description . . . . .	550
20.4.1	CRYP block diagram . . . . .	550
20.4.2	CRYP internal signals . . . . .	550
20.4.3	CRYP DES/TDES cryptographic core . . . . .	551
20.4.4	CRYP AES cryptographic core . . . . .	552
20.4.5	CRYP procedure to perform a cipher operation . . . . .	558
20.4.6	CRYP busy state . . . . .	560
20.4.7	Preparing the CRYP AES key for decryption . . . . .	561
20.4.8	CRYP stealing and data padding . . . . .	561
20.4.9	CRYP suspend/resume operations . . . . .	563
20.4.10	CRYP DES/TDES basic chaining modes (ECB, CBC) . . . . .	564
20.4.11	CRYP AES basic chaining modes (ECB, CBC) . . . . .	569
20.4.12	CRYP AES counter mode (AES-CTR) . . . . .	574
20.4.13	CRYP AES Galois/counter mode (GCM) . . . . .	578
20.4.14	CRYP AES Galois message authentication code (GMAC) . . . . .	583
20.4.15	CRYP AES Counter with CBC-MAC (CCM) . . . . .	584
20.4.16	CRYP data registers and data swapping . . . . .	589
20.4.17	CRYP key registers . . . . .	593
20.4.18	CRYP initialization vector registers . . . . .	593
20.4.19	CRYP DMA interface . . . . .	594
20.4.20	CRYP error management . . . . .	596
20.5	CRYP interrupts . . . . .	597
20.6	CRYP processing time . . . . .	598
20.7	CRYP registers . . . . .	599

20.7.1	CRYP control register (CRYP_CR) . . . . .	599
20.7.2	CRYP status register (CRYP_SR) . . . . .	601
20.7.3	CRYP data input register (CRYP_DIN) . . . . .	602
20.7.4	CRYP data output register (CRYP_DOUT) . . . . .	602
20.7.5	CRYP DMA control register (CRYP_DMACR) . . . . .	603
20.7.6	CRYP interrupt mask set/clear register (CRYP_IMSCR) . . . . .	604
20.7.7	CRYP raw interrupt status register (CRYP_RISR) . . . . .	604
20.7.8	CRYP masked interrupt status register (CRYP_MISR) . . . . .	605
20.7.9	CRYP key register 0L (CRYP_K0LR) . . . . .	606
20.7.10	CRYP key register 0R (CRYP_K0RR) . . . . .	606
20.7.11	CRYP key register 1L (CRYP_K1LR) . . . . .	607
20.7.12	CRYP key register 1R (CRYP_K1RR) . . . . .	607
20.7.13	CRYP key register 2L (CRYP_K2LR) . . . . .	608
20.7.14	CRYP key register 2R (CRYP_K2RR) . . . . .	608
20.7.15	CRYP key register 3L (CRYP_K3LR) . . . . .	609
20.7.16	CRYP key register 3R (CRYP_K3RR) . . . . .	609
20.7.17	CRYP initialization vector register 0L (CRYP_IV0LR) . . . . .	610
20.7.18	CRYP initialization vector register 0R (CRYP_IV0RR) . . . . .	610
20.7.19	CRYP initialization vector register 1L (CRYP_IV1LR) . . . . .	611
20.7.20	CRYP initialization vector register 1R (CRYP_IV1RR) . . . . .	611
20.7.21	CRYP register map . . . . .	611
<b>21</b>	<b>Hash processor (HASH) . . . . .</b>	<b>613</b>
21.1	Introduction . . . . .	613
21.2	HASH main features . . . . .	613
21.3	HASH implementation . . . . .	614
21.4	HASH functional description . . . . .	614
21.4.1	HASH block diagram . . . . .	614
21.4.2	HASH internal signals . . . . .	615
21.4.3	About secure hash algorithms . . . . .	615
21.4.4	Message data feeding . . . . .	615
21.4.5	Message digest computing . . . . .	617
21.4.6	Message padding . . . . .	618
21.4.7	HMAC operation . . . . .	620
21.4.8	HASH suspend/resume operations . . . . .	622
21.4.9	HASH DMA interface . . . . .	624
21.4.10	HASH error management . . . . .	624

---

21.4.11	HASH processing time . . . . .	624
21.5	HASH interrupts . . . . .	625
21.6	HASH registers . . . . .	626
21.6.1	HASH control register (HASH_CR) . . . . .	626
21.6.2	HASH data input register (HASH_DIN) . . . . .	628
21.6.3	HASH start register (HASH_STR) . . . . .	629
21.6.4	HASH digest registers . . . . .	630
21.6.5	HASH interrupt enable register (HASH_IMR) . . . . .	631
21.6.6	HASH status register (HASH_SR) . . . . .	632
21.6.7	HASH context swap registers . . . . .	632
21.6.8	HASH register map . . . . .	633
<b>22</b>	<b>Advanced-control timers (TIM1/TIM8) . . . . .</b>	<b>635</b>
22.1	TIM1/TIM8 introduction . . . . .	635
22.2	TIM1/TIM8 main features . . . . .	635
22.3	TIM1/TIM8 functional description . . . . .	637
22.3.1	Time-base unit . . . . .	637
22.3.2	Counter modes . . . . .	639
22.3.3	Repetition counter . . . . .	650
22.3.4	External trigger input . . . . .	652
22.3.5	Clock selection . . . . .	653
22.3.6	Capture/compare channels . . . . .	657
22.3.7	Input capture mode . . . . .	659
22.3.8	PWM input mode . . . . .	660
22.3.9	Forced output mode . . . . .	661
22.3.10	Output compare mode . . . . .	662
22.3.11	PWM mode . . . . .	663
22.3.12	Asymmetric PWM mode . . . . .	666
22.3.13	Combined PWM mode . . . . .	667
22.3.14	Combined 3-phase PWM mode . . . . .	668
22.3.15	Complementary outputs and dead-time insertion . . . . .	669
22.3.16	Using the break function . . . . .	671
22.3.17	Clearing the OCxREF signal on an external event . . . . .	677
22.3.18	6-step PWM generation . . . . .	679
22.3.19	One-pulse mode . . . . .	680
22.3.20	Retriggerable one pulse mode . . . . .	681
22.3.21	Encoder interface mode . . . . .	682

22.3.22	UIF bit remapping . . . . .	684
22.3.23	Timer input XOR function . . . . .	685
22.3.24	Interfacing with Hall sensors . . . . .	685
22.3.25	Timer synchronization . . . . .	688
22.3.26	ADC synchronization . . . . .	692
22.3.27	DMA burst mode . . . . .	692
22.3.28	Debug mode . . . . .	693
22.4	TIM1/TIM8 registers . . . . .	694
22.4.1	TIMx control register 1 (TIMx_CR1)(x = 1, 8) . . . . .	694
22.4.2	TIMx control register 2 (TIMx_CR2)(x = 1, 8) . . . . .	695
22.4.3	TIMx slave mode control register (TIMx_SMCR)(x = 1, 8) . . . . .	698
22.4.4	TIMx DMA/interrupt enable register (TIMx_DIER)(x = 1, 8) . . . . .	700
22.4.5	TIMx status register (TIMx_SR)(x = 1, 8) . . . . .	702
22.4.6	TIMx event generation register (TIMx_EGR)(x = 1, 8) . . . . .	704
22.4.7	TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 1, 8) . . . . .	705
22.4.8	TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 1, 8) . . . . .	706
22.4.9	TIMx capture/compare mode register 2 (TIMx_CCMR2)(x = 1, 8) . . . . .	709
22.4.10	TIMx capture/compare mode register 2 [alternate] (TIMx_CCMR2)(x = 1, 8) . . . . .	710
22.4.11	TIMx capture/compare enable register (TIMx_CCER)(x = 1, 8) . . . . .	711
22.4.12	TIMx counter (TIMx_CNT)(x = 1, 8) . . . . .	715
22.4.13	TIMx prescaler (TIMx_PSC)(x = 1, 8) . . . . .	715
22.4.14	TIMx auto-reload register (TIMx_ARR)(x = 1, 8) . . . . .	715
22.4.15	TIMx repetition counter register (TIMx_RCR)(x = 1, 8) . . . . .	716
22.4.16	TIMx capture/compare register 1 (TIMx_CCR1)(x = 1, 8) . . . . .	716
22.4.17	TIMx capture/compare register 2 (TIMx_CCR2)(x = 1, 8) . . . . .	717
22.4.18	TIMx capture/compare register 3 (TIMx_CCR3)(x = 1, 8) . . . . .	717
22.4.19	TIMx capture/compare register 4 (TIMx_CCR4)(x = 1, 8) . . . . .	718
22.4.20	TIMx break and dead-time register (TIMx_BDTR)(x = 1, 8) . . . . .	718
22.4.21	TIMx DMA control register (TIMx_DCR)(x = 1, 8) . . . . .	721

---

22.4.22	TIMx DMA address for full transfer (TIMx_DMAR)( $x = 1, 8$ ) . . . . .	722
22.4.23	TIMx capture/compare mode register 3 (TIMx_CCMR3)( $x = 1, 8$ ) . . . . .	723
22.4.24	TIMx capture/compare register 5 (TIMx_CCR5)( $x = 1, 8$ ) . . . . .	724
22.4.25	TIMx capture/compare register 6 (TIMx_CCR6)( $x = 1, 8$ ) . . . . .	725
22.4.26	TIM1 register map . . . . .	726
22.4.27	TIM8 register map . . . . .	728
<b>23</b>	<b>General-purpose timers (TIM2/TIM3/TIM4/TIM5) . . . . .</b>	<b>731</b>
23.1	TIM2/TIM3/TIM4/TIM5 introduction . . . . .	731
23.2	TIM2/TIM3/TIM4/TIM5 main features . . . . .	731
23.3	TIM2/TIM3/TIM4/TIM5 functional description . . . . .	733
23.3.1	Time-base unit . . . . .	733
23.3.2	Counter modes . . . . .	735
23.3.3	Clock selection . . . . .	745
23.3.4	Capture/Compare channels . . . . .	749
23.3.5	Input capture mode . . . . .	751
23.3.6	PWM input mode . . . . .	752
23.3.7	Forced output mode . . . . .	753
23.3.8	Output compare mode . . . . .	754
23.3.9	PWM mode . . . . .	755
23.3.10	Asymmetric PWM mode . . . . .	758
23.3.11	Combined PWM mode . . . . .	759
23.3.12	Clearing the OCxREF signal on an external event . . . . .	760
23.3.13	One-pulse mode . . . . .	762
23.3.14	Retriggerable one pulse mode . . . . .	763
23.3.15	Encoder interface mode . . . . .	764
23.3.16	UIF bit remapping . . . . .	766
23.3.17	Timer input XOR function . . . . .	766
23.3.18	Timers and external trigger synchronization . . . . .	767
23.3.19	Timer synchronization . . . . .	770
23.3.20	DMA burst mode . . . . .	775
23.3.21	Debug mode . . . . .	776
23.4	TIM2/TIM3/TIM4/TIM5 registers . . . . .	777
23.4.1	TIMx control register 1 (TIMx_CR1)( $x = 2$ to 5) . . . . .	777

23.4.2	TIMx control register 2 (TIMx_CR2)( $x = 2$ to $5$ ) . . . . .	778
23.4.3	TIMx slave mode control register (TIMx_SMCR)( $x = 2$ to $5$ ) . . . . .	780
23.4.4	TIMx DMA/Interrupt enable register (TIMx_DIER)( $x = 2$ to $5$ ) . . . . .	783
23.4.5	TIMx status register (TIMx_SR)( $x = 2$ to $5$ ) . . . . .	784
23.4.6	TIMx event generation register (TIMx_EGR)( $x = 2$ to $5$ ) . . . . .	785
23.4.7	TIMx capture/compare mode register 1 (TIMx_CCMR1)( $x = 2$ to $5$ ) . . . . .	786
23.4.8	TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)( $x = 2$ to $5$ ) . . . . .	788
23.4.9	TIMx capture/compare mode register 2 (TIMx_CCMR2)( $x = 2$ to $5$ ) . . . . .	790
23.4.10	TIMx capture/compare mode register 2 [alternate] (TIMx_CCMR2)( $x = 2$ to $5$ ) . . . . .	791
23.4.11	TIMx capture/compare enable register (TIMx_CCER)( $x = 2$ to $5$ ) . . . . .	792
23.4.12	TIMx counter (TIMx_CNT)( $x = 2$ to $5$ ) . . . . .	793
23.4.13	TIMx counter [alternate] (TIMx_CNT)( $x = 2$ to $5$ ) . . . . .	794
23.4.14	TIMx prescaler (TIMx_PSC)( $x = 2$ to $5$ ) . . . . .	794
23.4.15	TIMx auto-reload register (TIMx_ARR)( $x = 2$ to $5$ ) . . . . .	795
23.4.16	TIMx capture/compare register 1 (TIMx_CCR1)( $x = 2$ to $5$ ) . . . . .	795
23.4.17	TIMx capture/compare register 2 (TIMx_CCR2)( $x = 2$ to $5$ ) . . . . .	795
23.4.18	TIMx capture/compare register 3 (TIMx_CCR3)( $x = 2$ to $5$ ) . . . . .	796
23.4.19	TIMx capture/compare register 4 (TIMx_CCR4)( $x = 2$ to $5$ ) . . . . .	796
23.4.20	TIMx DMA control register (TIMx_DCR)( $x = 2$ to $5$ ) . . . . .	797
23.4.21	TIMx DMA address for full transfer (TIMx_DMAR)( $x = 2$ to $5$ ) . . . . .	798
23.4.22	TIM2 option register (TIM2_OR) . . . . .	798
23.4.23	TIM5 option register (TIM5_OR) . . . . .	798
23.4.24	TIMx register map . . . . .	800
<b>24</b>	<b>General-purpose timers (TIM9/TIM10/TIM11/TIM12/TIM13/TIM14) . . . . .</b>	<b>803</b>
24.1	TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 introduction . . . . .	803
24.2	TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 main features . . . . .	803
24.2.1	TIM9/TIM12 main features . . . . .	803
24.2.2	TIM10/TIM11/TIM13/TIM14 main features . . . . .	804
24.3	TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 functional description . . . . .	806
24.3.1	Time-base unit . . . . .	806
24.3.2	Counter modes . . . . .	808
24.3.3	Clock selection . . . . .	811
24.3.4	Capture/compare channels . . . . .	813
24.3.5	Input capture mode . . . . .	815

---

24.3.6	PWM input mode (only for TIM9/TIM12) . . . . .	816
24.3.7	Forced output mode . . . . .	817
24.3.8	Output compare mode . . . . .	817
24.3.9	PWM mode . . . . .	818
24.3.10	Combined PWM mode (TIM9/TIM12 only) . . . . .	819
24.3.11	One-pulse mode . . . . .	821
24.3.12	Retriggerable one pulse mode (TIM12 only) . . . . .	822
24.3.13	UIF bit remapping . . . . .	823
24.3.14	TIM9/TIM12 external trigger synchronization . . . . .	823
24.3.15	Slave mode – combined reset + trigger mode . . . . .	826
24.3.16	Timer synchronization (TIM9/TIM12) . . . . .	827
24.3.17	Using timer output as trigger for other timers (TIM10/TIM11/TIM13/TIM14) 827	
24.3.18	Debug mode . . . . .	827
24.4	TIM9/TIM12 registers . . . . .	827
24.4.1	TIMx control register 1 (TIMx_CR1)(x = 9, 12) . . . . .	827
24.4.2	TIMx slave mode control register (TIMx_SMCR)(x = 9, 12) . . . . .	828
24.4.3	TIMx Interrupt enable register (TIMx_DIER)(x = 9, 12) . . . . .	830
24.4.4	TIMx status register (TIMx_SR)(x = 9, 12) . . . . .	830
24.4.5	TIMx event generation register (TIMx_EGR)(x = 9, 12) . . . . .	832
24.4.6	TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 9, 12) . . . . .	833
24.4.7	TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 9, 12) . . . . .	834
24.4.8	TIMx capture/compare enable register (TIMx_CCER)(x = 9, 12) . . . . .	836
24.4.9	TIMx counter (TIMx_CNT)(x = 9, 12) . . . . .	837
24.4.10	TIMx prescaler (TIMx_PSC)(x = 9, 12) . . . . .	838
24.4.11	TIMx auto-reload register (TIMx_ARR)(x = 9, 12) . . . . .	838
24.4.12	TIMx capture/compare register 1 (TIMx_CCR1)(x = 9, 12) . . . . .	838
24.4.13	TIMx capture/compare register 2 (TIMx_CCR2)(x = 9, 12) . . . . .	839
24.4.14	TIM9/TIM12 register map . . . . .	840
24.5	TIM10/TIM11/TIM13/TIM14 registers . . . . .	842
24.5.1	TIMx control register 1 (TIMx_CR1)(x = 10, 11, 13, 14) . . . . .	842
24.5.2	TIMx Interrupt enable register (TIMx_DIER)(x = 10, 11, 13, 14) . . . . .	843
24.5.3	TIMx status register (TIMx_SR)(x = 10, 11, 13, 14) . . . . .	843
24.5.4	TIMx event generation register (TIMx_EGR)(x = 10, 11, 13, 14) . . . . .	844
24.5.5	TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 10, 11, 13, 14) . . . . .	845

24.5.6	TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 10, 11, 13, 14) . . . . .	846
24.5.7	TIMx capture/compare enable register (TIMx_CCER)(x = 10, 11, 13, 14) . . . . .	848
24.5.8	TIMx counter (TIMx_CNT)(x = 10, 11, 13, 14) . . . . .	849
24.5.9	TIMx prescaler (TIMx_PSC)(x = 10, 11, 13, 14) . . . . .	850
24.5.10	TIMx auto-reload register (TIMx_ARR)(x = 10, 11, 13, 14) . . . . .	850
24.5.11	TIMx capture/compare register 1 (TIMx_CCR1)(x = 10, 11, 13, 14) . . . . .	850
24.5.12	TIM11 option register 1 (TIM11_OR) . . . . .	851
24.5.13	TIM10/TIM11/TIM13/TIM14 register map . . . . .	851
<b>25</b>	<b>Basic timers (TIM6/TIM7) . . . . .</b>	<b>853</b>
25.1	TIM6/TIM7 introduction . . . . .	853
25.2	TIM6/TIM7 main features . . . . .	853
25.3	TIM6/TIM7 functional description . . . . .	854
25.3.1	Time-base unit . . . . .	854
25.3.2	Counting mode . . . . .	856
25.3.3	UIF bit remapping . . . . .	859
25.3.4	Clock source . . . . .	859
25.3.5	Debug mode . . . . .	860
25.4	TIM6/TIM7 registers . . . . .	860
25.4.1	TIMx control register 1 (TIMx_CR1)(x = 6 to 7) . . . . .	860
25.4.2	TIMx control register 2 (TIMx_CR2)(x = 6 to 7) . . . . .	862
25.4.3	TIMx DMA/Interrupt enable register (TIMx_DIER)(x = 6 to 7) . . . . .	862
25.4.4	TIMx status register (TIMx_SR)(x = 6 to 7) . . . . .	863
25.4.5	TIMx event generation register (TIMx_EGR)(x = 6 to 7) . . . . .	863
25.4.6	TIMx counter (TIMx_CNT)(x = 6 to 7) . . . . .	863
25.4.7	TIMx prescaler (TIMx_PSC)(x = 6 to 7) . . . . .	864
25.4.8	TIMx auto-reload register (TIMx_ARR)(x = 6 to 7) . . . . .	864
25.4.9	TIMx register map . . . . .	865
<b>26</b>	<b>Low-power timer (LPTIM) . . . . .</b>	<b>866</b>
26.1	Introduction . . . . .	866
26.2	LPTIM main features . . . . .	866
26.3	LPTIM implementation . . . . .	867
26.4	LPTIM functional description . . . . .	867
26.4.1	LPTIM block diagram . . . . .	867

---

26.4.2	LPTIM trigger mapping . . . . .	868
26.4.3	LPTIM reset and clocks . . . . .	868
26.4.4	Glitch filter . . . . .	868
26.4.5	Prescaler . . . . .	869
26.4.6	Trigger multiplexer . . . . .	870
26.4.7	Operating mode . . . . .	870
26.4.8	Timeout function . . . . .	872
26.4.9	Waveform generation . . . . .	872
26.4.10	Register update . . . . .	873
26.4.11	Counter mode . . . . .	874
26.4.12	Timer enable . . . . .	874
26.4.13	Encoder mode . . . . .	875
26.4.14	Debug mode . . . . .	876
26.5	LPTIM low-power modes . . . . .	876
26.6	LPTIM interrupts . . . . .	877
26.7	LPTIM registers . . . . .	877
26.7.1	LPTIM interrupt and status register (LPTIM_ISR) . . . . .	878
26.7.2	LPTIM interrupt clear register (LPTIM_ICR) . . . . .	879
26.7.3	LPTIM interrupt enable register (LPTIM_IER) . . . . .	879
26.7.4	LPTIM configuration register (LPTIM_CFGR) . . . . .	880
26.7.5	LPTIM control register (LPTIM_CR) . . . . .	883
26.7.6	LPTIM compare register (LPTIM_CMP) . . . . .	884
26.7.7	LPTIM autoreload register (LPTIM_ARR) . . . . .	885
26.7.8	LPTIM counter register (LPTIM_CNT) . . . . .	885
26.7.9	LPTIM register map . . . . .	886
<b>27</b>	<b>Independent watchdog (IWDG) . . . . .</b>	<b>887</b>
27.1	Introduction . . . . .	887
27.2	IWDG main features . . . . .	887
27.3	IWDG functional description . . . . .	887
27.3.1	IWDG block diagram . . . . .	887
27.3.2	Window option . . . . .	888
27.3.3	Hardware watchdog . . . . .	889
27.3.4	Low-power freeze . . . . .	889
27.3.5	Register access protection . . . . .	889
27.3.6	Debug mode . . . . .	889

27.4	IWDG registers . . . . .	890
27.4.1	IWDG key register (IWDG_KR) . . . . .	890
27.4.2	IWDG prescaler register (IWDG_PR) . . . . .	891
27.4.3	IWDG reload register (IWDG_RLR) . . . . .	892
27.4.4	IWDG status register (IWDG_SR) . . . . .	893
27.4.5	IWDG window register (IWDG_WINR) . . . . .	894
27.4.6	IWDG register map . . . . .	895
<b>28</b>	<b>System window watchdog (WWDG) . . . . .</b>	<b>896</b>
28.1	Introduction . . . . .	896
28.2	WWDG main features . . . . .	896
28.3	WWDG functional description . . . . .	896
28.3.1	WWDG block diagram . . . . .	897
28.3.2	Enabling the watchdog . . . . .	897
28.3.3	Controlling the down-counter . . . . .	897
28.3.4	How to program the watchdog timeout . . . . .	897
28.3.5	Debug mode . . . . .	899
28.4	WWDG interrupts . . . . .	899
28.5	WWDG registers . . . . .	899
28.5.1	WWDG control register (WWDG_CR) . . . . .	899
28.5.2	WWDG configuration register (WWDG_CFR) . . . . .	900
28.5.3	WWDG status register (WWDG_SR) . . . . .	900
28.5.4	WWDG register map . . . . .	901
<b>29</b>	<b>Real-time clock (RTC) . . . . .</b>	<b>902</b>
29.1	Introduction . . . . .	902
29.2	RTC main features . . . . .	903
29.3	RTC functional description . . . . .	904
29.3.1	RTC block diagram . . . . .	904
29.3.2	GPIOs controlled by the RTC . . . . .	905
29.3.3	Clock and prescalers . . . . .	907
29.3.4	Real-time clock and calendar . . . . .	908
29.3.5	Programmable alarms . . . . .	908
29.3.6	Periodic auto-wake-up . . . . .	909
29.3.7	RTC initialization and configuration . . . . .	910
29.3.8	Reading the calendar . . . . .	911

---

29.3.9	Resetting the RTC . . . . .	912
29.3.10	RTC synchronization . . . . .	913
29.3.11	RTC reference clock detection . . . . .	913
29.3.12	RTC smooth digital calibration . . . . .	914
29.3.13	Time-stamp function . . . . .	916
29.3.14	Tamper detection . . . . .	917
29.3.15	Calibration clock output . . . . .	919
29.3.16	Alarm output . . . . .	919
29.4	RTC low-power modes . . . . .	919
29.5	RTC interrupts . . . . .	920
29.6	RTC registers . . . . .	921
29.6.1	RTC time register (RTC_TR) . . . . .	921
29.6.2	RTC date register (RTC_DR) . . . . .	922
29.6.3	RTC control register (RTC_CR) . . . . .	923
29.6.4	RTC initialization and status register (RTC_ISR) . . . . .	926
29.6.5	RTC prescaler register (RTC_PRER) . . . . .	929
29.6.6	RTC wake-up timer register (RTC_WUTR) . . . . .	930
29.6.7	RTC alarm A register (RTC_ALRMAR) . . . . .	931
29.6.8	RTC alarm B register (RTC_ALRMBR) . . . . .	932
29.6.9	RTC write protection register (RTC_WPR) . . . . .	933
29.6.10	RTC sub second register (RTC_SSR) . . . . .	933
29.6.11	RTC shift control register (RTC_SHIFTR) . . . . .	934
29.6.12	RTC timestamp time register (RTC_TSTR) . . . . .	935
29.6.13	RTC timestamp date register (RTC_TSDR) . . . . .	936
29.6.14	RTC time-stamp sub second register (RTC_TSSSR) . . . . .	937
29.6.15	RTC calibration register (RTC_CALR) . . . . .	938
29.6.16	RTC tamper configuration register (RTC_TAMPCCR) . . . . .	939
29.6.17	RTC alarm A sub second register (RTC_ALRMASSR) . . . . .	942
29.6.18	RTC alarm B sub second register (RTC_ALRMBSSR) . . . . .	943
29.6.19	RTC option register (RTC_OR) . . . . .	944
29.6.20	RTC backup registers (RTC_BKPxR) . . . . .	944
29.6.21	RTC register map . . . . .	945
<b>30</b>	<b>Inter-integrated circuit interface (I2C) . . . . .</b>	<b>947</b>
30.1	Introduction . . . . .	947
30.2	I2C main features . . . . .	947

30.3	I2C implementation .....	948
30.4	I2C functional description .....	948
30.4.1	I2C block diagram .....	949
30.4.2	I2C pins and internal signals .....	949
30.4.3	I2C clock requirements .....	950
30.4.4	I2C mode selection .....	950
30.4.5	I2C initialization .....	951
30.4.6	I2C reset .....	955
30.4.7	I2C data transfer .....	955
30.4.8	I2C target mode .....	957
30.4.9	I2C controller mode .....	966
30.4.10	I2C_TIMINGR register configuration examples .....	977
30.4.11	SMBus specific features .....	979
30.4.12	SMBus initialization .....	982
30.4.13	SMBus I2C_TIMEOUTR register configuration examples .....	984
30.4.14	SMBus target mode .....	984
30.4.15	SMBus controller mode .....	988
30.4.16	Error conditions .....	991
30.5	I2C in low-power modes .....	993
30.6	I2C interrupts .....	993
30.7	I2C DMA requests .....	994
30.7.1	Transmission using DMA .....	994
30.7.2	Reception using DMA .....	994
30.8	I2C debug modes .....	995
30.9	I2C registers .....	995
30.9.1	I2C control register 1 (I2C_CR1) .....	995
30.9.2	I2C control register 2 (I2C_CR2) .....	998
30.9.3	I2C own address 1 register (I2C_OAR1) .....	1000
30.9.4	I2C own address 2 register (I2C_OAR2) .....	1001
30.9.5	I2C timing register (I2C_TIMINGR) .....	1002
30.9.6	I2C timeout register (I2C_TIMEOUTR) .....	1003
30.9.7	I2C interrupt and status register (I2C_ISR) .....	1004
30.9.8	I2C interrupt clear register (I2C_ICR) .....	1006
30.9.9	I2C PEC register (I2C_PECR) .....	1007
30.9.10	I2C receive data register (I2C_RXDR) .....	1007
30.9.11	I2C transmit data register (I2C_TXDR) .....	1008

---

30.9.12	I2C register map	1009
<b>31</b>	<b>Universal synchronous/asynchronous receiver transmitter (USART/UART)</b>	<b>1010</b>
31.1	Introduction	1010
31.2	USART main features	1010
31.3	USART extended features	1011
31.4	USART implementation	1012
31.5	USART functional description	1012
31.5.1	USART character description	1015
31.5.2	USART transmitter	1017
31.5.3	USART receiver	1019
31.5.4	USART baud rate generation	1025
31.5.5	Tolerance of the USART receiver to clock deviation	1028
31.5.6	USART auto baud rate detection	1029
31.5.7	Multiprocessor communication using USART	1030
31.5.8	Modbus communication using USART	1032
31.5.9	USART parity control	1033
31.5.10	USART LIN (local interconnection network) mode	1034
31.5.11	USART synchronous mode	1036
31.5.12	USART single-wire half-duplex communication	1039
31.5.13	USART smartcard mode	1039
31.5.14	USART IrDA SIR ENDEC block	1044
31.5.15	USART continuous communication in DMA mode	1046
31.5.16	RS232 hardware flow control and RS485 driver enable using USART	1048
31.6	USART in low-power modes	1050
31.7	USART interrupts	1051
31.8	USART registers	1053
31.8.1	USART control register 1 (USART_CR1)	1053
31.8.2	USART control register 2 (USART_CR2)	1056
31.8.3	USART control register 3 (USART_CR3)	1059
31.8.4	USART baud rate register (USART_BRR)	1062
31.8.5	USART guard time and prescaler register (USART_GTPR)	1063
31.8.6	USART receiver timeout register (USART_RTOR)	1064
31.8.7	USART request register (USART_RQR)	1065
31.8.8	USART interrupt and status register (USART_ISR)	1066

31.8.9	USART interrupt flag clear register (USART_ICR) . . . . .	1070
31.8.10	USART receive data register (USART_RDR) . . . . .	1071
31.8.11	USART transmit data register (USART_TDR) . . . . .	1072
31.8.12	USART register map . . . . .	1072
<b>32</b>	<b>Serial peripheral interface / integrated interchip sound (SPI/I2S) . 1074</b>	
32.1	Introduction . . . . .	1074
32.2	SPI main features . . . . .	1074
32.3	I2S main features . . . . .	1075
32.4	SPI/I2S implementation . . . . .	1075
32.5	SPI functional description . . . . .	1076
32.5.1	General description . . . . .	1076
32.5.2	Communications between one master and one slave . . . . .	1077
32.5.3	Standard multislave communication . . . . .	1079
32.5.4	Multimaster communication . . . . .	1080
32.5.5	Slave select (NSS) pin management . . . . .	1081
32.5.6	Communication formats . . . . .	1082
32.5.7	Configuration of SPI . . . . .	1084
32.5.8	Procedure for enabling SPI . . . . .	1085
32.5.9	Data transmission and reception procedures . . . . .	1085
32.5.10	SPI status flags . . . . .	1095
32.5.11	SPI error flags . . . . .	1096
32.5.12	NSS pulse mode . . . . .	1097
32.5.13	TI mode . . . . .	1097
32.5.14	CRC calculation . . . . .	1098
32.6	SPI interrupts . . . . .	1100
32.7	I2S functional description . . . . .	1101
32.7.1	I2S general description . . . . .	1101
32.7.2	I2S full duplex . . . . .	1102
32.7.3	Supported audio protocols . . . . .	1103
32.7.4	Start-up description . . . . .	1110
32.7.5	Clock generator . . . . .	1112
32.7.6	I <sup>2</sup> S master mode . . . . .	1115
32.7.7	I <sup>2</sup> S slave mode . . . . .	1116
32.7.8	I2S status flags . . . . .	1118
32.7.9	I2S error flags . . . . .	1119

---

32.7.10	DMA features . . . . .	1120
32.8	I2S interrupts . . . . .	1120
32.9	SPI and I2S registers . . . . .	1121
32.9.1	SPI control register 1 (SPIx_CR1) . . . . .	1121
32.9.2	SPI control register 2 (SPIx_CR2) . . . . .	1123
32.9.3	SPI status register (SPIx_SR) . . . . .	1125
32.9.4	SPI data register (SPIx_DR) . . . . .	1127
32.9.5	SPI CRC polynomial register (SPIx_CRCPR) . . . . .	1127
32.9.6	SPI Rx CRC register (SPIx_RXCRCR) . . . . .	1127
32.9.7	SPI Tx CRC register (SPIx_TXCRCR) . . . . .	1128
32.9.8	SPIx_I2S configuration register (SPIx_I2SCFGR) . . . . .	1128
32.9.9	SPIx_I2S prescaler register (SPIx_I2SPR) . . . . .	1130
32.9.10	SPI/I2S register map . . . . .	1132
<b>33</b>	<b>Serial audio interface (SAI) . . . . .</b>	<b>1133</b>
33.1	Introduction . . . . .	1133
33.2	SAI main features . . . . .	1133
33.3	SAI functional description . . . . .	1135
33.3.1	SAI block diagram . . . . .	1135
33.3.2	SAI pins and internal signals . . . . .	1136
33.3.3	Main SAI modes . . . . .	1136
33.3.4	SAI synchronization mode . . . . .	1137
33.3.5	Audio data size . . . . .	1138
33.3.6	Frame synchronization . . . . .	1139
33.3.7	Slot configuration . . . . .	1142
33.3.8	SAI clock generator . . . . .	1144
33.3.9	Internal FIFOs . . . . .	1146
33.3.10	AC'97 link controller . . . . .	1148
33.3.11	SPDIF output . . . . .	1150
33.3.12	Specific features . . . . .	1153
33.3.13	Error flags . . . . .	1157
33.3.14	Disabling the SAI . . . . .	1160
33.3.15	SAI DMA interface . . . . .	1160
33.4	SAI interrupts . . . . .	1161
33.5	SAI registers . . . . .	1163
33.5.1	SAI global configuration register (SAI_GCR) . . . . .	1163

33.5.2	SAI configuration register 1 (SAI_ACR1) . . . . .	1163
33.5.3	SAI configuration register 2 (SAI_ACR2) . . . . .	1166
33.5.4	SAI frame configuration register (SAI_AFRCR) . . . . .	1168
33.5.5	SAI slot register (SAI_ASLOTR) . . . . .	1169
33.5.6	SAI interrupt mask register (SAI_AIM) . . . . .	1170
33.5.7	SAI status register (SAI_ASР) . . . . .	1171
33.5.8	SAI clear flag register (SAI_ACLRFR) . . . . .	1173
33.5.9	SAI data register (SAI_ADR) . . . . .	1174
33.5.10	SAI configuration register 1 (SAI_BCR1) . . . . .	1175
33.5.11	SAI configuration register 2 (SAI_BCR2) . . . . .	1177
33.5.12	SAI frame configuration register (SAI_BFRCR) . . . . .	1179
33.5.13	SAI slot register (SAI_BSLOTR) . . . . .	1180
33.5.14	SAI interrupt mask register (SAI_BIM) . . . . .	1181
33.5.15	SAI status register (SAI_BSR) . . . . .	1182
33.5.16	SAI clear flag register (SAI_BCLRFR) . . . . .	1184
33.5.17	SAI data register (SAI_BDR) . . . . .	1185
33.5.18	SAI register map . . . . .	1186
<b>34</b>	<b>SPDIF receiver interface (SPDIFRX) . . . . .</b>	<b>1188</b>
34.1	SPDIFRX interface introduction . . . . .	1188
34.2	SPDIFRX main features . . . . .	1188
34.3	SPDIFRX functional description . . . . .	1188
34.3.1	S/PDIF protocol (IEC-60958) . . . . .	1189
34.3.2	SPDIFRX decoder (SPDIFRX_DC) . . . . .	1192
34.3.3	SPDIFRX tolerance to clock deviation . . . . .	1195
34.3.4	SPDIFRX synchronization . . . . .	1195
34.3.5	SPDIFRX handling . . . . .	1197
34.3.6	Data reception management . . . . .	1199
34.3.7	Dedicated control flow . . . . .	1201
34.3.8	Reception errors . . . . .	1202
34.3.9	Clocking strategy . . . . .	1204
34.3.10	DMA interface . . . . .	1204
34.3.11	Interrupt generation . . . . .	1205
34.3.12	Register protection . . . . .	1206
34.4	Programming procedures . . . . .	1207
34.4.1	Initialization phase . . . . .	1207
34.4.2	Handling of interrupts coming from SPDIFRX . . . . .	1208

---

34.4.3	Handling of interrupts coming from DMA .....	1208
34.5	SPDIFRX interface registers .....	1209
34.5.1	SPDIFRX control register (SPDIFRX_CR) .....	1209
34.5.2	SPDIFRX interrupt mask register (SPDIFRX_IMR) .....	1211
34.5.3	SPDIFRX status register (SPDIFRX_SR) .....	1212
34.5.4	SPDIFRX interrupt flag clear register (SPDIFRX_IFCR) .....	1214
34.5.5	SPDIFRX data input register (SPDIFRX_FMT0_DR) .....	1215
34.5.6	SPDIFRX data input register (SPDIFRX_FMT1_DR) .....	1215
34.5.7	SPDIFRX data input register (SPDIFRX_FMT2_DR) .....	1216
34.5.8	SPDIFRX channel status register (SPDIFRX_CSR) .....	1217
34.5.9	SPDIFRX debug information register (SPDIFRX_DIR) .....	1217
34.5.10	SPDIFRX interface register map .....	1218
<b>35</b>	<b>SD/SDIO/MMC card host interface (SDMMC) .....</b>	<b>1219</b>
35.1	SDMMC main features .....	1219
35.2	SDMMC bus topology .....	1219
35.3	SDMMC functional description .....	1221
35.3.1	SDMMC adapter .....	1223
35.3.2	SDMMC APB2 interface .....	1234
35.4	Card functional description .....	1235
35.4.1	Card identification mode .....	1235
35.4.2	Card reset .....	1236
35.4.3	Operating voltage range validation .....	1236
35.4.4	Card identification process .....	1236
35.4.5	Block write .....	1237
35.4.6	Block read .....	1238
35.4.7	Stream access, stream write and stream read (MultiMediaCard only) .....	1238
35.4.8	Erase: group erase and sector erase .....	1240
35.4.9	Wide bus selection or deselection .....	1240
35.4.10	Protection management .....	1240
35.4.11	Card status register .....	1244
35.4.12	SD status register .....	1247
35.4.13	SD I/O mode .....	1251
35.4.14	Commands and responses .....	1252
35.5	Response formats .....	1255
35.5.1	R1 (normal response command) .....	1256

35.5.2	R1b .....	1256
35.5.3	R2 (CID, CSD register) .....	1256
35.5.4	R3 (OCR register) .....	1257
35.5.5	R4 (Fast I/O) .....	1257
35.5.6	R4b .....	1257
35.5.7	R5 (interrupt request) .....	1258
35.5.8	R6 .....	1258
35.6	SDIO I/O card-specific operations .....	1259
35.6.1	SDIO I/O read wait operation by SDMMC_D2 signalling .....	1259
35.6.2	SDIO read wait operation by stopping SDMMC_CK .....	1260
35.6.3	SDIO suspend/resume operation .....	1260
35.6.4	SDIO interrupts .....	1260
35.7	HW flow control .....	1260
35.8	SDMMC registers .....	1261
35.8.1	SDMMC power control register (SDMMC_POWER) .....	1261
35.8.2	SDMMC clock control register (SDMMC_CLKCR) .....	1261
35.8.3	SDMMC argument register (SDMMC_ARG) .....	1264
35.8.4	SDMMC command register (SDMMC_CMD) .....	1264
35.8.5	SDMMC command response register (SDMMC_RESPCMD) .....	1265
35.8.6	SDMMC response 1..4 register (SDMMC_RESPx) .....	1265
35.8.7	SDMMC data timer register (SDMMC_DTIMER) .....	1266
35.8.8	SDMMC data length register (SDMMC_DLEN) .....	1267
35.8.9	SDMMC data control register (SDMMC_DCTRL) .....	1267
35.8.10	SDMMC data counter register (SDMMC_DCOUNT) .....	1270
35.8.11	SDMMC status register (SDMMC_STA) .....	1270
35.8.12	SDMMC interrupt clear register (SDMMC_ICR) .....	1271
35.8.13	SDMMC mask register (SDMMC_MASK) .....	1273
35.8.14	SDMMC FIFO counter register (SDMMC_FIFOCNT) .....	1275
35.8.15	SDMMC data FIFO register (SDMMC_FIFO) .....	1276
35.8.16	SDMMC register map .....	1277
<b>36</b>	<b>Controller area network (bxCAN) .....</b>	<b>1279</b>
36.1	Introduction .....	1279
36.2	bxCAN main features .....	1279
36.3	bxCAN general description .....	1280
36.3.1	CAN 2.0B active core .....	1280

---

36.3.2	Control, status, and configuration registers . . . . .	1280
36.3.3	Tx mailboxes . . . . .	1280
36.3.4	Acceptance filters . . . . .	1281
36.4	bxCAN operating modes . . . . .	1282
36.4.1	Initialization mode . . . . .	1282
36.4.2	Normal mode . . . . .	1282
36.4.3	Sleep mode (low-power) . . . . .	1283
36.5	Test mode . . . . .	1284
36.5.1	Silent mode . . . . .	1284
36.5.2	Loop back mode . . . . .	1284
36.5.3	Loop back combined with silent mode . . . . .	1285
36.6	Behavior in debug mode . . . . .	1285
36.7	bxCAN functional description . . . . .	1285
36.7.1	Transmission handling . . . . .	1285
36.7.2	Time triggered communication mode . . . . .	1287
36.7.3	Reception handling . . . . .	1287
36.7.4	Identifier filtering . . . . .	1289
36.7.5	Message storage . . . . .	1293
36.7.6	Error management . . . . .	1294
36.7.7	Bit timing . . . . .	1294
36.8	bxCAN interrupts . . . . .	1298
36.9	CAN registers . . . . .	1299
36.9.1	Register access protection . . . . .	1299
36.9.2	CAN control and status registers . . . . .	1299
36.9.3	CAN mailbox registers . . . . .	1309
36.9.4	CAN filter registers . . . . .	1315
36.9.5	bxCAN register map . . . . .	1319
37	<b>USB on-the-go full-speed/high-speed (OTG_FS/OTG_HS) . . . . .</b>	<b>1323</b>
37.1	Introduction . . . . .	1323
37.2	OTG_FS/OTG_HS main features . . . . .	1325
37.2.1	General features . . . . .	1325
37.2.2	Host-mode features . . . . .	1326
37.2.3	Peripheral-mode features . . . . .	1326
37.3	OTG_FS/OTG_HS implementation . . . . .	1327

37.4	OTG_FS/OTG_HS functional description . . . . .	1328
37.4.1	OTG_FS/OTG_HS block diagram . . . . .	1328
37.4.2	OTG_FS/OTG_HS pin and internal signals . . . . .	1329
37.4.3	OTG_FS/OTG_HS core . . . . .	1330
37.4.4	Embedded full-speed OTG PHY connected to OTG_FS . . . . .	1330
37.4.5	Embedded full-speed OTG PHY connected to OTG_HS . . . . .	1331
37.4.6	OTG detections . . . . .	1331
37.4.7	High-speed OTG PHY connected to OTG_HS . . . . .	1331
37.5	OTG_FS/OTG_HS dual role device (DRD) . . . . .	1332
37.5.1	ID line detection . . . . .	1332
37.5.2	HNP dual role device . . . . .	1332
37.5.3	SRP dual role device . . . . .	1333
37.6	OTG_FS/OTG_HS as a USB peripheral . . . . .	1333
37.6.1	SRP-capable peripheral . . . . .	1334
37.6.2	Peripheral states . . . . .	1334
37.6.3	Peripheral endpoints . . . . .	1335
37.7	OTG_FS/OTG_HS as a USB host . . . . .	1337
37.7.1	SRP-capable host . . . . .	1338
37.7.2	USB host states . . . . .	1338
37.7.3	Host channels . . . . .	1340
37.7.4	Host scheduler . . . . .	1341
37.8	OTG_FS/OTG_HS SOF trigger . . . . .	1342
37.8.1	Host SOFs . . . . .	1342
37.8.2	Peripheral SOFs . . . . .	1342
37.9	OTG_FS/OTG_HS low-power modes . . . . .	1343
37.10	OTG_FS/OTG_HS Dynamic update of the OTG_HFIR register . . . . .	1344
37.11	OTG_FS/OTG_HS data FIFOs . . . . .	1344
37.11.1	Peripheral FIFO architecture . . . . .	1345
37.11.2	Host FIFO architecture . . . . .	1346
37.11.3	FIFO RAM allocation . . . . .	1347
37.12	OTG_FS system performance . . . . .	1349
37.13	OTG_FS/OTG_HS interrupts . . . . .	1349
37.14	OTG_FS/OTG_HS control and status registers . . . . .	1351
37.14.1	CSR memory map . . . . .	1351
37.15	OTG_FS/OTG_HS registers . . . . .	1357
37.15.1	OTG control and status register (OTG_GOTGCTL) . . . . .	1357

---

37.15.2 OTG interrupt register (OTG_GOTGINT) . . . . .	1360
37.15.3 OTG AHB configuration register (OTG_GAHBCFG) . . . . .	1362
37.15.4 OTG USB configuration register (OTG_GUSBCFG) . . . . .	1364
37.15.5 OTG reset register (OTG_GRSTCTL) . . . . .	1367
37.15.6 OTG core interrupt register (OTG_GINTSTS) . . . . .	1370
37.15.7 OTG interrupt mask register (OTG_GINTMSK) . . . . .	1375
37.15.8 OTG receive status debug read register (OTG_GRXSTSR) . . . . .	1378
37.15.9 OTG receive status debug read [alternate] (OTG_GRXSTSR) . . . . .	1379
37.15.10 OTG status read and pop registers (OTG_GRXSTSP) . . . . .	1380
37.15.11 OTG status read and pop registers [alternate] (OTG_GRXSTSP) . . . . .	1381
37.15.12 OTG receive FIFO size register (OTG_GRXFSIZ) . . . . .	1382
37.15.13 OTG host non-periodic transmit FIFO size register (OTG_HNPTXFSIZ)/Endpoint 0 Transmit FIFO size (OTG_DIEPTXF0) . . . . .	1383
37.15.14 OTG non-periodic transmit FIFO/queue status register (OTG_HNPTXSTS) . . . . .	1384
37.15.15 OTG general core configuration register (OTG_GCCFG) . . . . .	1385
37.15.16 OTG core ID register (OTG_CID) . . . . .	1386
37.15.17 OTG core LPM configuration register (OTG_GLPMCFG) . . . . .	1386
37.15.18 OTG host periodic transmit FIFO size register (OTG_HPTXFSIZ) . . . . .	1390
37.15.19 OTG device IN endpoint transmit FIFO x size register (OTG_DIEPTXFx) . . . . .	1390
37.15.20 Host-mode registers . . . . .	1391
37.15.21 OTG host configuration register (OTG_HCFG) . . . . .	1391
37.15.22 OTG host frame interval register (OTG_HFIR) . . . . .	1392
37.15.23 OTG host frame number/frame time remaining register (OTG_HFNUM) . . . . .	1393
37.15.24 OTG_Host periodic transmit FIFO/queue status register (OTG_HPTXSTS) . . . . .	1393
37.15.25 OTG host all channels interrupt register (OTG_HAINT) . . . . .	1394
37.15.26 OTG host all channels interrupt mask register (OTG_HAINTMSK) . . . . .	1395
37.15.27 OTG host port control and status register (OTG_HPRT) . . . . .	1396
37.15.28 OTG host channel x characteristics register (OTG_HCCHARx) . . . . .	1398
37.15.29 OTG host channel x split control register (OTG_HCSPLTx) . . . . .	1399
37.15.30 OTG host channel x interrupt register (OTG_HCINTTx) . . . . .	1400
37.15.31 OTG host channel x interrupt mask register (OTG_HCINTMSKx) . . . . .	1402
37.15.32 OTG host channel x transfer size register (OTG_HCTSIZx) . . . . .	1403

37.15.33 OTG host channel x DMA address register (OTG_HCDMAX) . . . . .	1404
37.15.34 Device-mode registers . . . . .	1405
37.15.35 OTG device configuration register (OTG_DCFG) . . . . .	1405
37.15.36 OTG device control register (OTG_DCTL) . . . . .	1407
37.15.37 OTG device status register (OTG_DSTS) . . . . .	1409
37.15.38 OTG device IN endpoint common interrupt mask register (OTG_DIEPMSK) . . . . .	1410
37.15.39 OTG device OUT endpoint common interrupt mask register (OTG_DOEPMSK) . . . . .	1411
37.15.40 OTG device all endpoints interrupt register (OTG_DAINT) . . . . .	1413
37.15.41 OTG all endpoints interrupt mask register (OTG_DAINTMSK) . . . . .	1413
37.15.42 OTG device V <sub>BUS</sub> discharge time register (OTG_DVBUSDIS) . . . . .	1414
37.15.43 OTG device V <sub>BUS</sub> pulsing time register (OTG_DVBUSPULSE) . . . . .	1414
37.15.44 OTG device threshold control register (OTG_DTHRCTL) . . . . .	1415
37.15.45 OTG device IN endpoint FIFO empty interrupt mask register (OTG_DIEPEMPMSK) . . . . .	1416
37.15.46 OTG device each endpoint interrupt register (OTG_DEACHINT) . . . . .	1417
37.15.47 OTG device each endpoint interrupt mask register (OTG_DEACHINTMSK) . . . . .	1417
37.15.48 OTG device each IN endpoint-1 interrupt mask register (OTG_HS_DIEPEACHMSK1) . . . . .	1418
37.15.49 OTG device each OUT endpoint-1 interrupt mask register (OTG_HS_DOEPEACHMSK1) . . . . .	1419
37.15.50 OTG device control IN endpoint 0 control register (OTG_DIEPCTL0) . . . . .	1420
37.15.51 OTG device IN endpoint x control register (OTG_DIEPCTLx) . . . . .	1422
37.15.52 OTG device IN endpoint x interrupt register (OTG_DIEPINTx) . . . . .	1424
37.15.53 OTG device IN endpoint 0 transfer size register (OTG_DIEPTSIZ0) . . . . .	1426
37.15.54 OTG device IN endpoint x DMA address register (OTG_DIEPDMAx) . . . . .	1427
37.15.55 OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) . . . . .	1427
37.15.56 OTG device IN endpoint x transfer size register (OTG_DIEPTSIZx) .	1428
37.15.57 OTG device control OUT endpoint 0 control register (OTG_DOEPCTL0) . . . . .	1429
37.15.58 OTG device OUT endpoint x interrupt register (OTG_DOEPINTx) .	1430

---

37.15.59	OTG device OUT endpoint 0 transfer size register (OTG_DOEPTSIZ0) . . . . .	1432
37.15.60	OTG device OUT endpoint x DMA address register (OTG_DOEPDMAx) . . . . .	1433
37.15.61	OTG device OUT endpoint x control register (OTG_DOEPCTLx) . . . . .	1434
37.15.62	OTG device OUT endpoint x transfer size register (OTG_DOEPTSIZx) . . . . .	1436
37.15.63	OTG power and clock gating control register (OTG_PCGCCTL) . . .	1437
37.15.64	OTG_FS/OTG_HS register map . . . . .	1438
37.16	OTG_FS/OTG_HS programming model . . . . .	1451
37.16.1	Core initialization . . . . .	1451
37.16.2	Host initialization . . . . .	1452
37.16.3	Device initialization . . . . .	1453
37.16.4	DMA mode . . . . .	1453
37.16.5	Host programming model . . . . .	1453
37.16.6	Device programming model . . . . .	1487
37.16.7	Worst case response time . . . . .	1507
37.16.8	OTG programming model . . . . .	1509
38	<b>Ethernet (ETH): media access control (MAC) with DMA controller . . . . .</b>	<b>1516</b>
38.1	Ethernet introduction . . . . .	1516
38.2	Ethernet main features . . . . .	1516
38.2.1	MAC core features . . . . .	1517
38.2.2	DMA features . . . . .	1518
38.2.3	PTP features . . . . .	1518
38.3	Ethernet pins . . . . .	1519
38.4	Ethernet functional description: SMI, MII and RMII . . . . .	1520
38.4.1	Station management interface: SMI . . . . .	1520
38.4.2	Media-independent interface: MII . . . . .	1524
38.4.3	Reduced media-independent interface: RMII . . . . .	1526
38.4.4	MII/RMII selection . . . . .	1527
38.5	Ethernet functional description: MAC 802.3 . . . . .	1528
38.5.1	MAC 802.3 frame format . . . . .	1528
38.5.2	MAC frame transmission . . . . .	1532
38.5.3	MAC frame reception . . . . .	1539
38.5.4	MAC interrupts . . . . .	1545

38.5.5	MAC filtering . . . . .	1545
38.5.6	MAC loopback mode . . . . .	1548
38.5.7	MAC management counters: MMC . . . . .	1548
38.5.8	Power management: PMT . . . . .	1549
38.5.9	Precision time protocol (IEEE1588 PTP) . . . . .	1552
38.6	Ethernet functional description: DMA controller operation . . . . .	1558
38.6.1	Initialization of a transfer using DMA . . . . .	1559
38.6.2	Host bus burst access . . . . .	1559
38.6.3	Host data buffer alignment . . . . .	1560
38.6.4	Buffer size calculations . . . . .	1560
38.6.5	DMA arbiter . . . . .	1561
38.6.6	Error response to DMA . . . . .	1561
38.6.7	Tx DMA configuration . . . . .	1561
38.6.8	Rx DMA configuration . . . . .	1573
38.6.9	DMA interrupts . . . . .	1584
38.7	Ethernet interrupts . . . . .	1585
38.8	Ethernet registers . . . . .	1586
38.8.1	MAC register description . . . . .	1586
38.8.2	MMC register description . . . . .	1606
38.8.3	IEEE 1588 time stamp registers . . . . .	1611
38.8.4	DMA register description . . . . .	1619
38.8.5	Ethernet register maps . . . . .	1634
<b>39</b>	<b>HDMI-CEC controller (CEC) . . . . .</b>	<b>1638</b>
39.1	HDMI-CEC introduction . . . . .	1638
39.2	HDMI-CEC controller main features . . . . .	1638
39.3	HDMI-CEC functional description . . . . .	1639
39.3.1	HDMI-CEC pin . . . . .	1639
39.3.2	HDMI-CEC block diagram . . . . .	1639
39.3.3	Message description . . . . .	1639
39.3.4	Bit timing . . . . .	1640
39.4	Arbitration . . . . .	1641
39.4.1	SFT option bit . . . . .	1642
39.5	Error handling . . . . .	1643
39.5.1	Bit error . . . . .	1643
39.5.2	Message error . . . . .	1643

---

39.5.3	Bit rising error (BRE) . . . . .	1643
39.5.4	Short bit period error (SBPE) . . . . .	1644
39.5.5	Long bit period error (LBPE) . . . . .	1644
39.5.6	Transmission error detection (TXERR) . . . . .	1645
39.6	HDMI-CEC interrupts . . . . .	1647
39.7	HDMI-CEC registers . . . . .	1648
39.7.1	CEC control register (CEC_CR) . . . . .	1648
39.7.2	CEC configuration register (CEC_CFGR) . . . . .	1649
39.7.3	CEC Tx data register (CEC_TXDR) . . . . .	1651
39.7.4	CEC Rx data register (CEC_RXDR) . . . . .	1651
39.7.5	CEC interrupt and status register (CEC_ISR) . . . . .	1651
39.7.6	CEC interrupt enable register (CEC_IER) . . . . .	1653
39.7.7	HDMI-CEC register map . . . . .	1655
<b>40</b>	<b>Debug support (DBG) . . . . .</b>	<b>1656</b>
40.1	Overview . . . . .	1656
40.2	Reference Arm® documentation . . . . .	1657
40.3	SWJ debug port (serial wire and JTAG) . . . . .	1657
40.3.1	Mechanism to select the JTAG-DP or the SW-DP . . . . .	1658
40.4	Pinout and debug port pins . . . . .	1658
40.4.1	SWJ debug port pins . . . . .	1659
40.4.2	Flexible SWJ-DP pin assignment . . . . .	1659
40.4.3	Internal pull-up and pull-down on JTAG pins . . . . .	1659
40.4.4	Using serial wire and releasing the unused debug pins as GPIOs . . . . .	1661
40.5	STM32F75xxx and STM32F74xxx JTAG Debug Port connection . . . . .	1661
40.6	ID codes and locking mechanism . . . . .	1663
40.6.1	MCU device ID code . . . . .	1663
40.6.2	Boundary scan Debug Port . . . . .	1663
40.6.3	Cortex®-M7 with FPU Debug Port . . . . .	1663
40.6.4	Cortex®-M7 with FPU JEDEC-106 ID code . . . . .	1664
40.7	JTAG debug port . . . . .	1664
40.8	SW debug port . . . . .	1666
40.8.1	SW protocol introduction . . . . .	1666
40.8.2	SW protocol sequence . . . . .	1666
40.8.3	SW-DP state machine (reset, idle states, ID code) . . . . .	1667
40.8.4	DP and AP read/write accesses . . . . .	1668

40.8.5	SW-DP registers . . . . .	1668
40.8.6	SW-AP registers . . . . .	1669
40.9	AHB-AP (AHB access port) - valid for both JTAG-DP and SW-DP . . . . .	1670
40.10	Core debug . . . . .	1671
40.11	Capability of the debugger host to connect under system reset . . . . .	1672
40.12	FPB (Flash patch breakpoint) . . . . .	1672
40.13	DWT (data watchpoint trigger) . . . . .	1673
40.14	ITM (instrumentation trace macrocell) . . . . .	1673
40.14.1	General description . . . . .	1673
40.14.2	Time stamp packets, synchronization and overflow packets . . . . .	1673
40.15	ETM (Embedded trace macrocell) . . . . .	1675
40.15.1	General description . . . . .	1675
40.15.2	Signal protocol, packet types . . . . .	1675
40.15.3	Main ETM registers . . . . .	1676
40.15.4	Configuration example . . . . .	1676
40.16	MCU debug component (DBGMCU) . . . . .	1676
40.16.1	Debug support for low-power modes . . . . .	1676
40.16.2	Debug support for timers, watchdog, bxCAN and I <sup>2</sup> C . . . . .	1677
40.16.3	Debug MCU configuration register . . . . .	1677
40.16.4	DBGMCU_CR register . . . . .	1678
40.16.5	Debug MCU APB1 freeze register (DBGMCU_APB1_FZ) . . . . .	1679
40.16.6	Debug MCU APB2 Freeze register (DBGMCU_APB2_FZ) . . . . .	1681
40.17	Pelican TPIU (trace port interface unit) . . . . .	1682
40.17.1	Introduction . . . . .	1682
40.17.2	TRACE pin assignment . . . . .	1683
40.17.3	TPIU formatter . . . . .	1685
40.17.4	TPIU frame synchronization packets . . . . .	1685
40.17.5	Transmission of the synchronization frame packet . . . . .	1685
40.17.6	Synchronous mode . . . . .	1686
40.17.7	Asynchronous mode . . . . .	1686
40.17.8	TRACECLKIN connection inside the STM32F75xxx and STM32F74xxx . . . . .	1686
40.17.9	TPIU registers . . . . .	1687
40.17.10	Example of configuration . . . . .	1687
40.18	DBG register map . . . . .	1688

<b>41</b>	<b>Device electronic signature</b>	<b>1689</b>
41.1	Unique device ID register (96 bits)	1689
41.2	Flash size	1690
41.3	Package data register	1691
<b>42</b>	<b>Important security notice</b>	<b>1692</b>
<b>43</b>	<b>Revision history</b>	<b>1693</b>

## List of tables

Table 1.	STM32F75xxx and STM32F74xxx register boundary addresses .....	71
Table 2.	Boot modes.....	75
Table 3.	STM32F756xx and STM32F74xxx Flash memory organization .....	79
Table 4.	STM32F750xx Flash memory organization .....	79
Table 5.	Number of wait states according to CPU clock (HCLK) frequency.....	80
Table 6.	Maximum program/erase parallelism .....	82
Table 7.	Flash interrupt request .....	85
Table 8.	Option byte organization.....	85
Table 9.	Access versus read protection level .....	90
Table 10.	OTP area organization .....	91
Table 11.	Flash register map and reset values.....	99
Table 12.	Voltage regulator configuration mode versus device operating mode .....	106
Table 13.	Low-power mode summary .....	111
Table 14.	Features over all modes .....	112
Table 15.	Sleep-now entry and exit .....	116
Table 16.	Sleep-on-exit entry and exit .....	117
Table 17.	Stop operating modes.....	117
Table 18.	Stop mode entry and exit (STM32F75xxx and STM32F74xxx) .....	119
Table 19.	Standby mode entry and exit .....	121
Table 20.	PWR - register map and reset values.....	132
Table 21.	RCC register map and reset values .....	198
Table 22.	Port bit configuration table .....	203
Table 23.	GPIO register map and reset values .....	218
Table 24.	SYSCFG register map and reset values.....	225
Table 25.	DMA1 request mapping .....	230
Table 26.	DMA2 request mapping .....	230
Table 27.	Source and destination address .....	232
Table 28.	Source and destination address registers in double-buffer mode (DBM = 1).....	237
Table 29.	Packing/unpacking and endian behavior (bit PINC = MINC = 1) .....	238
Table 30.	Restriction on NDT versus PSIZE and MSIZE .....	238
Table 31.	FIFO threshold configurations .....	241
Table 32.	Possible DMA configurations .....	245
Table 33.	DMA interrupt requests .....	247
Table 34.	DMA register map and reset values .....	257
Table 35.	Supported color mode in input .....	263
Table 36.	Data order in memory .....	264
Table 37.	Alpha mode configuration .....	265
Table 38.	Supported CLUT color mode .....	266
Table 39.	CLUT data order in system memory .....	266
Table 40.	Supported color mode in output .....	267
Table 41.	Data order in memory .....	267
Table 42.	DMA2D interrupt requests .....	272
Table 43.	DMA2D register map and reset values .....	288
Table 44.	STM32F75xxx and STM32F74xxx vector table .....	290
Table 45.	External interrupt/event controller register map and reset values.....	301
Table 46.	CRC internal input/output signals .....	303
Table 47.	CRC register map and reset values .....	308
Table 48.	NOR/PSRAM bank selection .....	313

---

Table 49.	NOR/PSRAM External memory address . . . . .	314
Table 50.	NAND memory mapping and timing registers. . . . .	314
Table 51.	NAND bank selection . . . . .	314
Table 52.	SDRAM bank selection. . . . .	315
Table 53.	SDRAM address mapping . . . . .	315
Table 54.	SDRAM address mapping with 8-bit data bus width. . . . .	316
Table 55.	SDRAM address mapping with 16-bit data bus width. . . . .	316
Table 56.	SDRAM address mapping with 32-bit data bus width. . . . .	317
Table 57.	Programmable NOR/PSRAM access parameters . . . . .	319
Table 58.	Non-multiplexed I/O NOR flash memory . . . . .	320
Table 59.	16-bit multiplexed I/O NOR flash memory . . . . .	320
Table 60.	Non-multiplexed I/Os PSRAM/SRAM . . . . .	320
Table 61.	16-Bit multiplexed I/O PSRAM . . . . .	321
Table 62.	NOR flash/PSRAM: example of supported memories and transactions . . . . .	322
Table 63.	FMC_BCRx bitfields (mode 1) . . . . .	325
Table 64.	FMC_BTRx bitfields (mode 1) . . . . .	325
Table 65.	FMC_BCRx bitfields (mode A) . . . . .	327
Table 66.	FMC_BTRx bitfields (mode A) . . . . .	327
Table 67.	FMC_BWTRx bitfields (mode A) . . . . .	328
Table 68.	FMC_BCRx bitfields (mode 2/B) . . . . .	330
Table 69.	FMC_BTRx bitfields (mode 2/B) . . . . .	330
Table 70.	FMC_BWTRx bitfields (mode 2/B) . . . . .	331
Table 71.	FMC_BCRx bitfields (mode C) . . . . .	332
Table 72.	FMC_BTRx bitfields (mode C) . . . . .	333
Table 73.	FMC_BWTRx bitfields (mode C) . . . . .	333
Table 74.	FMC_BCRx bitfields (mode D) . . . . .	335
Table 75.	FMC_BTRx bitfields (mode D) . . . . .	335
Table 76.	FMC_BWTRx bitfields (mode D) . . . . .	336
Table 77.	FMC_BCRx bitfields (Muxed mode) . . . . .	337
Table 78.	FMC_BTRx bitfields (Muxed mode) . . . . .	338
Table 79.	FMC_BCRx bitfields (Synchronous multiplexed read mode) . . . . .	343
Table 80.	FMC_BTRx bitfields (Synchronous multiplexed read mode) . . . . .	344
Table 81.	FMC_BCRx bitfields (Synchronous multiplexed write mode) . . . . .	345
Table 82.	FMC_BTRx bitfields (Synchronous multiplexed write mode) . . . . .	346
Table 83.	Programmable NAND flash access parameters . . . . .	354
Table 84.	8-bit NAND flash . . . . .	354
Table 85.	16-bit NAND flash . . . . .	355
Table 86.	Supported memories and transactions . . . . .	355
Table 87.	ECC result relevant bits . . . . .	364
Table 88.	SDRAM signals. . . . .	365
Table 89.	FMC register map and reset values . . . . .	382
Table 90.	QUADSPI pins . . . . .	386
Table 91.	QUADSPI interrupt requests . . . . .	400
Table 92.	QUADSPI register map and reset values . . . . .	411
Table 93.	ADC pins . . . . .	415
Table 94.	Analog watchdog channel selection . . . . .	421
Table 95.	Configuring the trigger polarity . . . . .	426
Table 96.	External trigger for regular channels. . . . .	426
Table 97.	External trigger for injected channels . . . . .	427
Table 98.	ADC interrupts . . . . .	442
Table 99.	ADC global register map . . . . .	457

Table 100.	ADC register map and reset values for each ADC . . . . .	457
Table 101.	ADC register map and reset values (common ADC registers) . . . . .	459
Table 102.	DAC pins . . . . .	461
Table 103.	External triggers . . . . .	464
Table 104.	DAC register map . . . . .	481
Table 105.	DCMI input/output pins . . . . .	483
Table 106.	Positioning of captured data bytes in 32-bit words (8-bit width) . . . . .	485
Table 107.	Positioning of captured data bytes in 32-bit words (10-bit width) . . . . .	485
Table 108.	Positioning of captured data bytes in 32-bit words (12-bit width) . . . . .	485
Table 109.	Positioning of captured data bytes in 32-bit words (14-bit width) . . . . .	486
Table 110.	Data storage in monochrome progressive video format . . . . .	491
Table 111.	Data storage in RGB progressive video format . . . . .	492
Table 112.	Data storage in YCbCr progressive video format . . . . .	492
Table 113.	Data storage in YCbCr progressive video format - Y extraction mode . . . . .	492
Table 114.	DCMI interrupts . . . . .	493
Table 115.	DCMI register map and reset values . . . . .	503
Table 116.	LTDC pins and signal interface . . . . .	505
Table 117.	Clock domain for each register . . . . .	506
Table 118.	Pixel data mapping versus color format . . . . .	511
Table 119.	LTDC interrupt requests . . . . .	515
Table 120.	LTDC register map and reset values . . . . .	534
Table 121.	RNG internal input/output signals . . . . .	538
Table 122.	RNG interrupt requests . . . . .	543
Table 123.	RNG configurations . . . . .	544
Table 124.	RNG register map and reset map . . . . .	547
Table 125.	CRYP internal input/output signals . . . . .	550
Table 126.	Counter mode initialization vector . . . . .	576
Table 127.	GCM last block definition . . . . .	579
Table 128.	GCM mode IV registers initialization . . . . .	579
Table 129.	CCM mode IV registers initialization . . . . .	586
Table 130.	DES/TDES data swapping example . . . . .	590
Table 131.	AES data swapping example . . . . .	591
Table 132.	Key endianness in CRYP_KxR/LR registers (AES 128/192/256-bit keys) . . . . .	593
Table 133.	Key endianness in CRYP_KxR/LR registers (DES K1 and TDES K1/2/3) . . . . .	593
Table 134.	Initialization vector endianness in CRYP_IVx(L/R)R registers (AES) . . . . .	594
Table 135.	Initialization vector endianness in CRYP_IVx(L/R)R registers (DES/TDES) . . . . .	594
Table 136.	Cryptographic processor configuration for memory-to-peripheral DMA transfers . . . . .	594
Table 137.	Cryptographic processor configuration for peripheral-to-memory DMA transfers . . . . .	595
Table 138.	CRYP interrupt requests . . . . .	597
Table 139.	Processing latency for ECB, CBC and CTR . . . . .	598
Table 140.	Processing time (in clock cycle) for GCM and CCM per 128-bit block . . . . .	598
Table 141.	CRYP register map and reset values . . . . .	611
Table 142.	HASH internal input/output signals . . . . .	615
Table 143.	Hash processor outputs . . . . .	618
Table 144.	Processing time (in clock cycle) . . . . .	624
Table 145.	HASH interrupt requests . . . . .	625
Table 146.	HASH register map and reset values . . . . .	633
Table 147.	Behavior of timer outputs versus BRK/BRK2 inputs . . . . .	676
Table 148.	Counting direction versus encoder signals . . . . .	683
Table 149.	TIMx internal trigger connection . . . . .	700

Table 150.	Output control bits for complementary OCx and OCxN channels with break feature . . . . .	714
Table 151.	TIM1 register map and reset values . . . . .	726
Table 152.	TIM8 register map and reset values . . . . .	728
Table 153.	Counting direction versus encoder signals . . . . .	765
Table 154.	TIMx internal trigger connection . . . . .	783
Table 155.	Output control bit for standard OCx channels . . . . .	793
Table 156.	TIM2/TIM3/TIM4/TIM5 register map and reset values . . . . .	800
Table 157.	TIMx internal trigger connection . . . . .	830
Table 158.	Output control bit for standard OCx channels . . . . .	837
Table 159.	TIM9/TIM12 register map and reset values . . . . .	840
Table 160.	Output control bit for standard OCx channels . . . . .	849
Table 161.	TIM10/TIM11/TIM13/TIM14 register map and reset values . . . . .	851
Table 162.	TIMx register map and reset values . . . . .	865
Table 163.	STM32F75xxx and STM32F74xxx LPTIM features . . . . .	867
Table 164.	LPTIM1 external trigger connection . . . . .	868
Table 165.	Prescaler division ratios . . . . .	869
Table 166.	Encoder counting scenarios . . . . .	875
Table 167.	Effect of low-power modes on the LPTIM . . . . .	876
Table 168.	Interrupt events . . . . .	877
Table 169.	LPTIM register map and reset values . . . . .	886
Table 170.	IWDG register map and reset values . . . . .	895
Table 171.	WWDG register map and reset values . . . . .	901
Table 172.	RTC pin PC13 configuration . . . . .	905
Table 173.	RTC pin PI8 configuration . . . . .	906
Table 174.	RTC pin PC2 configuration . . . . .	907
Table 175.	RTC functions over modes . . . . .	907
Table 176.	Effect of low-power modes on RTC . . . . .	919
Table 177.	Interrupt control bits . . . . .	920
Table 178.	RTC register map and reset values . . . . .	945
Table 179.	I2C implementation . . . . .	948
Table 180.	I2C input/output pins . . . . .	949
Table 181.	I2C internal input/output signals . . . . .	949
Table 182.	Comparison of analog and digital filters . . . . .	951
Table 183.	I <sup>2</sup> C-bus and SMBus specification data setup and hold times . . . . .	954
Table 184.	I2C configuration . . . . .	957
Table 185.	I <sup>2</sup> C-bus and SMBus specification clock timings . . . . .	968
Table 186.	Timing settings for f <sub>I2CCLK</sub> of 8 MHz . . . . .	978
Table 187.	Timing settings for f <sub>I2CCLK</sub> of 16 MHz . . . . .	978
Table 188.	Timing settings for f <sub>I2CCLK</sub> of 48 MHz . . . . .	979
Table 189.	SMBus timeout specifications . . . . .	981
Table 190.	SMBus with PEC configuration . . . . .	983
Table 191.	TIMEOUTA[11:0] for maximum t <sub>TIMEOUT</sub> of 25 ms . . . . .	984
Table 192.	TIMEOUTB[11:0] for maximum t <sub>LOW:SEXT</sub> and t <sub>LOW:MEXT</sub> of 8 ms . . . . .	984
Table 193.	TIMEOUTA[11:0] for maximum t <sub>IDLE</sub> of 50 µs . . . . .	984
Table 194.	Effect of low-power modes to I2C . . . . .	993
Table 195.	I2C interrupt requests . . . . .	993
Table 196.	I2C register map and reset values . . . . .	1009
Table 197.	STM32F75xxx and STM32F74xxx USART features . . . . .	1012
Table 198.	Noise detection from sampled data . . . . .	1024
Table 199.	Error calculation for programmed baud rates at f <sub>CK</sub> = 216 MHz in both cases of oversampling by 8 (OVER8 = 1) . . . . .	1027
Table 200.	Error calculation for programmed baud rates at f <sub>CK</sub> = 216 MHz . . . . .	

in both cases of oversampling by 16 (OVER8 = 0) . . . . .	1027
Table 201. Tolerance of the USART receiver when BRR [3:0] = 0000 . . . . .	1028
Table 202. Tolerance of the USART receiver when BRR [3:0] is different from 0000 . . . . .	1028
Table 203. Frame formats . . . . .	1033
Table 204. Effect of low-power modes on the USART . . . . .	1050
Table 205. USART interrupt requests . . . . .	1051
Table 206. USART register map and reset values . . . . .	1072
Table 207. STM32F75xxx and STM32F74xxx SPI and SPI/I2S implementation . . . . .	1075
Table 208. SPI interrupt requests . . . . .	1100
Table 209. Audio-frequency precision using 48 MHz clock derived from HSE . . . . .	1114
Table 210. I2S interrupt requests . . . . .	1120
Table 211. SPI/I2S register map and reset values . . . . .	1132
Table 212. SAI internal input/output signals . . . . .	1136
Table 213. SAI input/output pins . . . . .	1136
Table 214. External synchronization selection . . . . .	1138
Table 215. Example of possible audio frequency sampling range . . . . .	1145
Table 216. SOPD pattern . . . . .	1151
Table 217. Parity bit calculation . . . . .	1151
Table 218. Audio sampling frequency versus symbol rates . . . . .	1152
Table 219. SAI interrupt sources . . . . .	1161
Table 220. SAI register map and reset values . . . . .	1186
Table 221. Transition sequence for preamble . . . . .	1194
Table 222. Minimum SPDIFRX_CLK frequency versus audio sampling rate . . . . .	1204
Table 223. Bit field property versus SPDIFRX state . . . . .	1206
Table 224. SPDIFRX interface register map and reset values . . . . .	1218
Table 225. SDMMC I/O definitions . . . . .	1222
Table 226. Command format . . . . .	1227
Table 227. Short response format . . . . .	1228
Table 228. Long response format . . . . .	1228
Table 229. Command path status flags . . . . .	1228
Table 230. Data token format . . . . .	1231
Table 231. DPSM flags . . . . .	1232
Table 232. Transmit FIFO status flags . . . . .	1233
Table 233. Receive FIFO status flags . . . . .	1233
Table 234. Card status . . . . .	1244
Table 235. SD status . . . . .	1247
Table 236. Speed class code field . . . . .	1248
Table 237. Performance move field . . . . .	1249
Table 238. AU_SIZE field . . . . .	1249
Table 239. Maximum AU size . . . . .	1249
Table 240. Erase size field . . . . .	1250
Table 241. Erase timeout field . . . . .	1250
Table 242. Erase offset field . . . . .	1250
Table 243. Block-oriented write commands . . . . .	1253
Table 244. Block-oriented write protection commands . . . . .	1254
Table 245. Erase commands . . . . .	1254
Table 246. I/O mode commands . . . . .	1254
Table 247. Lock card . . . . .	1255
Table 248. Application-specific commands . . . . .	1255
Table 249. R1 response . . . . .	1256
Table 250. R2 response . . . . .	1256
Table 251. R3 response . . . . .	1257

Table 252. R4 response . . . . .	1257
Table 253. R4b response . . . . .	1257
Table 254. R5 response . . . . .	1258
Table 255. R6 response . . . . .	1259
Table 256. Response type and SDMMC_RESPx registers . . . . .	1266
Table 257. SDMMC register map . . . . .	1277
Table 258. Transmit mailbox mapping . . . . .	1293
Table 259. Receive mailbox mapping . . . . .	1293
Table 260. bxCAN register map and reset values . . . . .	1319
Table 261. OTG_HS speeds supported . . . . .	1324
Table 262. OTG_FS speeds supported . . . . .	1324
Table 263. OTG_FS/OTG_HS implementation . . . . .	1327
Table 264. OTG_FS input/output pins . . . . .	1329
Table 265. OTG_HS input/output pins . . . . .	1329
Table 266. OTG_FS/OTG_HS input/output signals . . . . .	1330
Table 267. Compatibility of STM32 low power modes with the OTG . . . . .	1343
Table 268. Core global control and status registers (CSRs) . . . . .	1351
Table 269. Host-mode control and status registers (CSRs) . . . . .	1352
Table 270. Device-mode control and status registers . . . . .	1354
Table 271. Data FIFO (DFIFO) access register map . . . . .	1357
Table 272. Power and clock gating control and status registers . . . . .	1357
Table 273. TRDT values (FS) . . . . .	1367
Table 274. TRDT values (HS) . . . . .	1367
Table 275. Minimum duration for soft disconnect . . . . .	1408
Table 276. OTG_FS/OTG_HS register map and reset values . . . . .	1438
Table 277. Alternate function mapping . . . . .	1519
Table 278. Management frame format . . . . .	1521
Table 279. Clock range . . . . .	1523
Table 280. TX interface signal encoding . . . . .	1525
Table 281. RX interface signal encoding . . . . .	1525
Table 282. Frame statuses . . . . .	1541
Table 283. Destination address filtering . . . . .	1547
Table 284. Source address filtering . . . . .	1548
Table 285. Receive descriptor 0 - encoding for bits 7, 5 and 0 (normal descriptor format only, EDFE=0) . . . . .	1579
Table 286. Time stamp snapshot dependency on registers bits . . . . .	1613
Table 287. Ethernet register map and reset values . . . . .	1634
Table 288. HDMI pin . . . . .	1639
Table 289. Error handling timing parameters . . . . .	1645
Table 290. TXERR timing parameters . . . . .	1646
Table 291. HDMI-CEC interrupts . . . . .	1647
Table 292. HDMI-CEC register map and reset values . . . . .	1655
Table 293. SWJ debug port pins . . . . .	1659
Table 294. Flexible SWJ-DP pin assignment . . . . .	1659
Table 295. JTAG debug port data registers . . . . .	1664
Table 296. 32-bit debug port registers addressed through the shifted value A[3:2] . . . . .	1666
Table 297. Packet request (8-bits) . . . . .	1667
Table 298. ACK response (3 bits) . . . . .	1667
Table 299. DATA transfer (33 bits) . . . . .	1667
Table 300. SW-DP registers . . . . .	1668
Table 301. Cortex®-M7 with FPU AHB-AP registers . . . . .	1670
Table 302. Core debug registers . . . . .	1671

Table 303. Main ITM registers . . . . .	1674
Table 304. Asynchronous TRACE pin assignment. . . . .	1683
Table 305. Synchronous TRACE pin assignment . . . . .	1683
Table 306. Flexible TRACE pin assignment. . . . .	1684
Table 307. Important TPIU registers. . . . .	1687
Table 308. DBG register map and reset values . . . . .	1688
Table 309. Document revision history . . . . .	1693

# List of figures

Figure 1.	System architecture for STM32F75xxx and STM32F74xxx devices .....	65
Figure 2.	Memory map .....	70
Figure 3.	Flash memory interface connection inside system architecture (STM32F75xxx and STM32F74xxx) .....	77
Figure 4.	RDP levels .....	90
Figure 5.	Power supply overview .....	101
Figure 6.	VDDUSB connected to VDD power supply .....	102
Figure 7.	VDDUSB connected to external independent power supply .....	103
Figure 8.	Backup domain .....	105
Figure 9.	Power-on reset/power-down reset waveform .....	108
Figure 10.	BOR thresholds .....	109
Figure 11.	PVD thresholds .....	110
Figure 12.	Simplified diagram of the reset circuit .....	134
Figure 13.	Clock tree .....	135
Figure 14.	HSE/ LSE clock sources .....	139
Figure 15.	Frequency measurement with TIM5 in Input capture mode .....	144
Figure 16.	Frequency measurement with TIM11 in Input capture mode .....	145
Figure 17.	Basic structure of an I/O port bit .....	202
Figure 18.	Basic structure of a 5-Volt tolerant I/O port bit .....	202
Figure 19.	Input floating / pull up / pull down configurations .....	207
Figure 20.	Output configuration .....	208
Figure 21.	Alternate function configuration .....	209
Figure 22.	High impedance-analog configuration .....	209
Figure 23.	DMA block diagram .....	228
Figure 24.	Channel selection .....	230
Figure 25.	Peripheral-to-memory mode .....	233
Figure 26.	Memory-to-peripheral mode .....	234
Figure 27.	Memory-to-memory mode .....	235
Figure 28.	FIFO structure .....	240
Figure 29.	DMA2D block diagram .....	262
Figure 30.	External interrupt/event controller block diagram .....	295
Figure 31.	External interrupt/event GPIO mapping .....	297
Figure 32.	CRC calculation unit block diagram .....	303
Figure 33.	FMC block diagram .....	310
Figure 34.	FMC memory banks .....	313
Figure 35.	Mode 1 read access waveforms .....	324
Figure 36.	Mode 1 write access waveforms .....	324
Figure 37.	Mode A read access waveforms .....	326
Figure 38.	Mode A write access waveforms .....	326
Figure 39.	Mode 2 and mode B read access waveforms .....	328
Figure 40.	Mode 2 write access waveforms .....	329
Figure 41.	Mode B write access waveforms .....	329
Figure 42.	Mode C read access waveforms .....	331
Figure 43.	Mode C write access waveforms .....	332
Figure 44.	Mode D read access waveforms .....	334
Figure 45.	Mode D write access waveforms .....	334
Figure 46.	Muxed read access waveforms .....	336
Figure 47.	Muxed write access waveforms .....	337

Figure 48.	Asynchronous wait during a read access waveforms . . . . .	339
Figure 49.	Asynchronous wait during a write access waveforms . . . . .	340
Figure 50.	Wait configuration waveforms . . . . .	342
Figure 51.	Synchronous multiplexed read mode waveforms - NOR, PSRAM (CRAM) . . . . .	343
Figure 52.	Synchronous multiplexed write mode waveforms - PSRAM (CRAM) . . . . .	345
Figure 53.	NAND flash controller waveforms for common memory access . . . . .	356
Figure 54.	Access to non 'CE don't care' NAND-flash . . . . .	358
Figure 55.	Burst write SDRAM access waveforms . . . . .	367
Figure 56.	Burst read SDRAM access . . . . .	368
Figure 57.	Logic diagram of Read access with RBURST bit set (CAS=1, RPIPE=0) . . . . .	369
Figure 58.	Read access crossing row boundary . . . . .	371
Figure 59.	Write access crossing row boundary . . . . .	371
Figure 60.	Self-refresh mode . . . . .	374
Figure 61.	Power-down mode . . . . .	375
Figure 62.	QUADSPI block diagram when dual-flash mode is disabled . . . . .	385
Figure 63.	QUADSPI block diagram when dual-flash mode is enabled . . . . .	386
Figure 64.	Example of read command in quad-SPI mode . . . . .	387
Figure 65.	Example of a DDR command in quad-SPI mode . . . . .	390
Figure 66.	NCS when CKMODE = 0 (T = CLK period) . . . . .	398
Figure 67.	NCS when CKMODE = 1 in SDR mode (T = CLK period) . . . . .	398
Figure 68.	NCS when CKMODE = 1 in DDR mode (T = CLK period) . . . . .	399
Figure 69.	NCS when CKMODE = 1 with an abort (T = CLK period) . . . . .	399
Figure 70.	Single ADC block diagram . . . . .	414
Figure 71.	ADC1 connectivity . . . . .	416
Figure 72.	ADC2 connectivity . . . . .	417
Figure 73.	ADC3 connectivity . . . . .	418
Figure 74.	Timing diagram . . . . .	421
Figure 75.	Analog watchdog's guarded area . . . . .	421
Figure 76.	Injected conversion latency . . . . .	423
Figure 77.	Right alignment of 12-bit data . . . . .	425
Figure 78.	Left alignment of 12-bit data . . . . .	425
Figure 79.	Left alignment of 6-bit data . . . . .	425
Figure 80.	Multi ADC block diagram <sup>(1)</sup> . . . . .	430
Figure 81.	Injected simultaneous mode on 4 channels: dual ADC mode . . . . .	433
Figure 82.	Injected simultaneous mode on 4 channels: triple ADC mode . . . . .	433
Figure 83.	Regular simultaneous mode on 16 channels: dual ADC mode . . . . .	434
Figure 84.	Regular simultaneous mode on 16 channels: triple ADC mode . . . . .	434
Figure 85.	Interleaved mode on 1 channel in continuous conversion mode: dual ADC mode . . . . .	435
Figure 86.	Interleaved mode on 1 channel in continuous conversion mode: triple ADC mode . . . . .	436
Figure 87.	Alternate trigger: injected group of each ADC . . . . .	437
Figure 88.	Alternate trigger: 4 injected channels (each ADC) in discontinuous mode . . . . .	437
Figure 89.	Alternate trigger: injected group of each ADC . . . . .	438
Figure 90.	Alternate + regular simultaneous . . . . .	439
Figure 91.	Case of trigger occurring during injected conversion . . . . .	439
Figure 92.	Temperature sensor and VREFINT channel block diagram . . . . .	440
Figure 93.	DAC channel block diagram . . . . .	461
Figure 94.	DAC output buffer connection . . . . .	462
Figure 95.	Data registers in single DAC channel mode . . . . .	463
Figure 96.	Data registers in dual DAC channel mode . . . . .	463
Figure 97.	Timing diagram for conversion with trigger disabled TEN = 0 . . . . .	464
Figure 98.	DAC LFSR register calculation algorithm . . . . .	466
Figure 99.	DAC conversion (SW trigger enabled) with LFSR wave generation . . . . .	466

---

Figure 100. DAC triangle wave generation . . . . .	467
Figure 101. DAC conversion (SW trigger enabled) with triangle wave generation . . . . .	467
Figure 102. DCMI block diagram . . . . .	483
Figure 103. Top-level block diagram . . . . .	483
Figure 104. DCMI signal waveforms . . . . .	484
Figure 105. Timing diagram . . . . .	486
Figure 106. Frame capture waveforms in snapshot mode . . . . .	488
Figure 107. Frame capture waveforms in continuous grab mode . . . . .	489
Figure 108. Coordinates and size of the window after cropping . . . . .	489
Figure 109. Data capture waveforms . . . . .	490
Figure 110. Pixel raster scan order . . . . .	491
Figure 111. LTDC block diagram . . . . .	505
Figure 112. LCD-TFT synchronous timings . . . . .	508
Figure 113. Layer window programmable parameters . . . . .	511
Figure 114. Blending two layers with background . . . . .	514
Figure 115. Interrupt events . . . . .	515
Figure 116. RNG block diagram . . . . .	538
Figure 117. Entropy source model . . . . .	539
Figure 118. CRYP block diagram . . . . .	550
Figure 119. AES-ECB mode overview . . . . .	553
Figure 120. AES-CBC mode overview . . . . .	554
Figure 121. AES-CTR mode overview . . . . .	555
Figure 122. AES-GCM mode overview . . . . .	556
Figure 123. AES-GMAC mode overview . . . . .	556
Figure 124. AES-CCM mode overview . . . . .	557
Figure 125. Example of suspend mode management . . . . .	563
Figure 126. DES/TDES-ECB mode encryption . . . . .	564
Figure 127. DES/TDES-ECB mode decryption . . . . .	565
Figure 128. DES/TDES-CBC mode encryption . . . . .	566
Figure 129. DES/TDES-CBC mode decryption . . . . .	567
Figure 130. AES-ECB mode encryption . . . . .	569
Figure 131. AES-ECB mode decryption . . . . .	570
Figure 132. AES-CBC mode encryption . . . . .	571
Figure 133. AES-CBC mode decryption . . . . .	572
Figure 134. Message construction for the Counter mode . . . . .	574
Figure 135. AES-CTR mode encryption . . . . .	575
Figure 136. AES-CTR mode decryption . . . . .	576
Figure 137. Message construction for the Galois/counter mode . . . . .	578
Figure 138. Message construction for the Galois Message Authentication Code mode . . . . .	583
Figure 139. Message construction for the Counter with CBC-MAC mode . . . . .	584
Figure 140. 64-bit block construction according to the data type (IN FIFO) . . . . .	591
Figure 141. 128-bit block construction according to the data type . . . . .	592
Figure 142. HASH block diagram . . . . .	614
Figure 143. Message data swapping feature . . . . .	616
Figure 144. HASH suspend/resume mechanism . . . . .	622
Figure 145. Advanced-control timer block diagram . . . . .	636
Figure 146. Counter timing diagram with prescaler division change from 1 to 2 . . . . .	638
Figure 147. Counter timing diagram with prescaler division change from 1 to 4 . . . . .	638
Figure 148. Counter timing diagram, internal clock divided by 1 . . . . .	640
Figure 149. Counter timing diagram, internal clock divided by 2 . . . . .	640
Figure 150. Counter timing diagram, internal clock divided by 4 . . . . .	641
Figure 151. Counter timing diagram, internal clock divided by N . . . . .	641

---

Figure 152. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded) . . . . .	642
Figure 153. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) . . . . .	642
Figure 154. Counter timing diagram, internal clock divided by 1 . . . . .	644
Figure 155. Counter timing diagram, internal clock divided by 2 . . . . .	644
Figure 156. Counter timing diagram, internal clock divided by 4 . . . . .	645
Figure 157. Counter timing diagram, internal clock divided by N . . . . .	645
Figure 158. Counter timing diagram, update event when repetition counter is not used . . . . .	646
Figure 159. Counter timing diagram, internal clock divided by 1, TIMx_ARR = 0x6 . . . . .	647
Figure 160. Counter timing diagram, internal clock divided by 2 . . . . .	648
Figure 161. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36 . . . . .	648
Figure 162. Counter timing diagram, internal clock divided by N . . . . .	649
Figure 163. Counter timing diagram, update event with ARPE=1 (counter underflow) . . . . .	649
Figure 164. Counter timing diagram, Update event with ARPE=1 (counter overflow) . . . . .	650
Figure 165. Update rate examples depending on mode and TIMx_RCR register settings . . . . .	651
Figure 166. External trigger input block . . . . .	652
Figure 167. Control circuit in normal mode, internal clock divided by 1 . . . . .	653
Figure 168. TI2 external clock connection example . . . . .	654
Figure 169. Control circuit in external clock mode 1 . . . . .	655
Figure 170. External trigger input block . . . . .	655
Figure 171. Control circuit in external clock mode 2 . . . . .	656
Figure 172. Capture/compare channel (example: channel 1 input stage) . . . . .	657
Figure 173. Capture/compare channel 1 main circuit . . . . .	658
Figure 174. Output stage of capture/compare channel (channel 1, idem ch. 2 and 3) . . . . .	658
Figure 175. Output stage of capture/compare channel (channel 4) . . . . .	659
Figure 176. Output stage of capture/compare channel (channel 5, idem ch. 6) . . . . .	659
Figure 177. PWM input mode timing . . . . .	661
Figure 178. Output compare mode, toggle on OC1 . . . . .	663
Figure 179. Edge-aligned PWM waveforms (ARR=8) . . . . .	664
Figure 180. Center-aligned PWM waveforms (ARR=8) . . . . .	665
Figure 181. Generation of 2 phase-shifted PWM signals with 50% duty cycle . . . . .	667
Figure 182. Combined PWM mode on channel 1 and 3 . . . . .	668
Figure 183. 3-phase combined PWM signals with multiple trigger pulses per period . . . . .	669
Figure 184. Complementary output with dead-time insertion . . . . .	670
Figure 185. Dead-time waveforms with delay greater than the negative pulse . . . . .	670
Figure 186. Dead-time waveforms with delay greater than the positive pulse . . . . .	671
Figure 187. Break and Break2 circuitry overview . . . . .	673
Figure 188. Various output behavior in response to a break event on BRK (OSSI = 1) . . . . .	675
Figure 189. PWM output state following BRK and BRK2 pins assertion (OSSI=1) . . . . .	676
Figure 190. PWM output state following BRK assertion (OSSI=0) . . . . .	677
Figure 191. Clearing TIMx OCxREF . . . . .	678
Figure 192. 6-step generation, COM example (OSSR=1) . . . . .	679
Figure 193. Example of one pulse mode . . . . .	680
Figure 194. Retriggerable one pulse mode . . . . .	682
Figure 195. Example of counter operation in encoder interface mode . . . . .	683
Figure 196. Example of encoder interface mode with TI1FP1 polarity inverted . . . . .	684
Figure 197. Measuring time interval between edges on 3 signals . . . . .	685
Figure 198. Example of Hall sensor interface . . . . .	687
Figure 199. Control circuit in reset mode . . . . .	688
Figure 200. Control circuit in Gated mode . . . . .	689
Figure 201. Control circuit in trigger mode . . . . .	690
Figure 202. Control circuit in external clock mode 2 + trigger mode . . . . .	691
Figure 203. General-purpose timer block diagram . . . . .	732

---

Figure 204. Counter timing diagram with prescaler division change from 1 to 2 .....	734
Figure 205. Counter timing diagram with prescaler division change from 1 to 4 .....	734
Figure 206. Counter timing diagram, internal clock divided by 1 .....	735
Figure 207. Counter timing diagram, internal clock divided by 2 .....	736
Figure 208. Counter timing diagram, internal clock divided by 4 .....	736
Figure 209. Counter timing diagram, internal clock divided by N .....	737
Figure 210. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded) .....	737
Figure 211. Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded) .....	738
Figure 212. Counter timing diagram, internal clock divided by 1 .....	739
Figure 213. Counter timing diagram, internal clock divided by 2 .....	739
Figure 214. Counter timing diagram, internal clock divided by 4 .....	740
Figure 215. Counter timing diagram, internal clock divided by N .....	740
Figure 216. Counter timing diagram, Update event .....	741
Figure 217. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6 .....	742
Figure 218. Counter timing diagram, internal clock divided by 2 .....	743
Figure 219. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36 .....	743
Figure 220. Counter timing diagram, internal clock divided by N .....	744
Figure 221. Counter timing diagram, Update event with ARPE=1 (counter underflow) .....	744
Figure 222. Counter timing diagram, Update event with ARPE=1 (counter overflow) .....	745
Figure 223. Control circuit in normal mode, internal clock divided by 1 .....	746
Figure 224. TI2 external clock connection example .....	746
Figure 225. Control circuit in external clock mode 1 .....	747
Figure 226. External trigger input block .....	748
Figure 227. Control circuit in external clock mode 2 .....	749
Figure 228. Capture/Compare channel (example: channel 1 input stage) .....	750
Figure 229. Capture/Compare channel 1 main circuit .....	750
Figure 230. Output stage of Capture/Compare channel (channel 1) .....	751
Figure 231. PWM input mode timing .....	753
Figure 232. Output compare mode, toggle on OC1 .....	755
Figure 233. Edge-aligned PWM waveforms (ARR=8) .....	756
Figure 234. Center-aligned PWM waveforms (ARR=8) .....	757
Figure 235. Generation of 2 phase-shifted PWM signals with 50% duty cycle .....	758
Figure 236. Combined PWM mode on channels 1 and 3 .....	760
Figure 237. Clearing TIMx OCxREF .....	761
Figure 238. Example of one-pulse mode .....	762
Figure 239. Retriggerable one-pulse mode .....	764
Figure 240. Example of counter operation in encoder interface mode .....	765
Figure 241. Example of encoder interface mode with TI1FP1 polarity inverted .....	766
Figure 242. Control circuit in reset mode .....	767
Figure 243. Control circuit in gated mode .....	768
Figure 244. Control circuit in trigger mode .....	769
Figure 245. Control circuit in external clock mode 2 + trigger mode .....	770
Figure 246. Master/Slave timer example .....	771
Figure 247. Master/slave connection example with 1 channel only timers .....	771
Figure 248. Gating TIM with OC1REF of TIM3 .....	772
Figure 249. Gating TIM with Enable of TIM3 .....	773
Figure 250. Triggering TIM with update of TIM3 .....	774
Figure 251. Triggering TIM with Enable of TIM3 .....	774
Figure 252. Triggering TIM3 and TIM2 with TIM3 TI1 input .....	775
Figure 253. General-purpose timer block diagram (TIM9/TIM12) .....	804
Figure 254. General-purpose timer block diagram (TIM10/TIM11/TIM13/TIM14) .....	805
Figure 255. Counter timing diagram with prescaler division change from 1 to 2 .....	807

---

Figure 256. Counter timing diagram with prescaler division change from 1 to 4 .....	807
Figure 257. Counter timing diagram, internal clock divided by 1 .....	808
Figure 258. Counter timing diagram, internal clock divided by 2 .....	809
Figure 259. Counter timing diagram, internal clock divided by 4 .....	809
Figure 260. Counter timing diagram, internal clock divided by N .....	810
Figure 261. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded) .....	810
Figure 262. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) .....	811
Figure 263. Control circuit in normal mode, internal clock divided by 1 .....	812
Figure 264. TI2 external clock connection example .....	812
Figure 265. Control circuit in external clock mode 1 .....	813
Figure 266. Capture/compare channel (example: channel 1 input stage) .....	814
Figure 267. Capture/compare channel 1 main circuit .....	814
Figure 268. Output stage of capture/compare channel (channel 1) .....	815
Figure 269. PWM input mode timing .....	817
Figure 270. Output compare mode, toggle on OC1 .....	818
Figure 271. Edge-aligned PWM waveforms (ARR=8) .....	819
Figure 272. Combined PWM mode on channel 1 and 2 .....	820
Figure 273. Example of one pulse mode .....	821
Figure 274. Retriggerable one pulse mode .....	823
Figure 275. Control circuit in reset mode .....	824
Figure 276. Control circuit in gated mode .....	825
Figure 277. Control circuit in trigger mode .....	825
Figure 278. Basic timer block diagram .....	853
Figure 279. Counter timing diagram with prescaler division change from 1 to 2 .....	855
Figure 280. Counter timing diagram with prescaler division change from 1 to 4 .....	855
Figure 281. Counter timing diagram, internal clock divided by 1 .....	856
Figure 282. Counter timing diagram, internal clock divided by 2 .....	857
Figure 283. Counter timing diagram, internal clock divided by 4 .....	857
Figure 284. Counter timing diagram, internal clock divided by N .....	858
Figure 285. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded) .....	858
Figure 286. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) .....	859
Figure 287. Control circuit in normal mode, internal clock divided by 1 .....	860
Figure 288. Low-power timer block diagram .....	867
Figure 289. Glitch filter timing diagram .....	869
Figure 290. LPTIM output waveform, single counting mode configuration .....	871
Figure 291. LPTIM output waveform, Single counting mode configuration and Set-once mode activated (WAVE bit is set) .....	871
Figure 292. LPTIM output waveform, Continuous counting mode configuration .....	872
Figure 293. Waveform generation .....	873
Figure 294. Encoder mode counting sequence .....	876
Figure 295. Independent watchdog block diagram .....	887
Figure 296. Watchdog block diagram .....	897
Figure 297. Window watchdog timing diagram .....	898
Figure 298. RTC block diagram .....	904
Figure 299. Block diagram .....	949
Figure 300. I <sup>2</sup> C-bus protocol .....	951
Figure 301. Setup and hold timings .....	952
Figure 302. I <sup>2</sup> C initialization flow .....	955

---

Figure 303. Data reception . . . . .	956
Figure 304. Data transmission . . . . .	956
Figure 305. Target initialization flow . . . . .	960
Figure 306. Transfer sequence flow for I2C target transmitter, NOSTRETCH = 0 . . . . .	961
Figure 307. Transfer sequence flow for I2C target transmitter, NOSTRETCH = 1 . . . . .	962
Figure 308. Transfer bus diagrams for I2C target transmitter (mandatory events only) . . . . .	963
Figure 309. Transfer sequence flow for I2C target receiver, NOSTRETCH = 0 . . . . .	964
Figure 310. Transfer sequence flow for I2C target receiver, NOSTRETCH = 1 . . . . .	965
Figure 311. Transfer bus diagrams for I2C target receiver (mandatory events only) . . . . .	965
Figure 312. Controller clock generation . . . . .	967
Figure 313. Controller initialization flow . . . . .	969
Figure 314. 10-bit address read access with HEAD10R = 0 . . . . .	969
Figure 315. 10-bit address read access with HEAD10R = 1 . . . . .	970
Figure 316. Transfer sequence flow for I2C controller transmitter, $N \leq 255$ bytes . . . . .	971
Figure 317. Transfer sequence flow for I2C controller transmitter, $N > 255$ bytes . . . . .	972
Figure 318. Transfer bus diagrams for I2C controller transmitter (mandatory events only) . . . . .	973
Figure 319. Transfer sequence flow for I2C controller receiver, $N \leq 255$ bytes . . . . .	975
Figure 320. Transfer sequence flow for I2C controller receiver, $N > 255$ bytes . . . . .	976
Figure 321. Transfer bus diagrams for I2C controller receiver (mandatory events only) . . . . .	977
Figure 322. Timeout intervals for $t_{LOW:SEXT}$ , $t_{LOW:MEXT}$ . . . . .	981
Figure 323. Transfer sequence flow for SMBus target transmitter $N$ bytes + PEC . . . . .	985
Figure 324. Transfer bus diagram for SMBus target transmitter (SBC = 1) . . . . .	985
Figure 325. Transfer sequence flow for SMBus target receiver $N$ bytes + PEC . . . . .	987
Figure 326. Bus transfer diagrams for SMBus target receiver (SBC = 1) . . . . .	988
Figure 327. Bus transfer diagrams for SMBus controller transmitter . . . . .	989
Figure 328. Bus transfer diagrams for SMBus controller receiver . . . . .	991
Figure 329. USART block diagram . . . . .	1014
Figure 330. Word length programming . . . . .	1016
Figure 331. Configurable stop bits . . . . .	1018
Figure 332. TC/TXE behavior when transmitting . . . . .	1019
Figure 333. Start bit detection when oversampling by 16 or 8 . . . . .	1020
Figure 334. Data sampling when oversampling by 16 . . . . .	1023
Figure 335. Data sampling when oversampling by 8 . . . . .	1024
Figure 336. Mute mode using Idle line detection . . . . .	1031
Figure 337. Mute mode using address mark detection . . . . .	1032
Figure 338. Break detection in LIN mode (11-bit break length - LBDL bit is set) . . . . .	1035
Figure 339. Break detection in LIN mode vs. Framing error detection . . . . .	1036
Figure 340. USART example of synchronous transmission . . . . .	1037
Figure 341. USART data clock timing diagram ( $M$ bits = 00) . . . . .	1037
Figure 342. USART data clock timing diagram ( $M$ bits = 01) . . . . .	1038
Figure 343. RX data setup/hold time . . . . .	1038
Figure 344. ISO 7816-3 asynchronous protocol . . . . .	1040
Figure 345. Parity error detection using the 1.5 stop bits . . . . .	1041
Figure 346. IrDA SIR ENDEC- block diagram . . . . .	1045
Figure 347. IrDA data modulation (3/16) - normal mode . . . . .	1045
Figure 348. Transmission using DMA . . . . .	1047
Figure 349. Reception using DMA . . . . .	1048
Figure 350. Hardware flow control between 2 USARTs . . . . .	1048
Figure 351. RS232 RTS flow control . . . . .	1049

---

Figure 352. RS232 CTS flow control . . . . .	1050
Figure 353. USART interrupt mapping diagram . . . . .	1052
Figure 354. SPI block diagram . . . . .	1076
Figure 355. Full-duplex single master/ single slave application . . . . .	1077
Figure 356. Half-duplex single master/ single slave application . . . . .	1078
Figure 357. Simplex single master/single slave application (master in transmit-only/ slave in receive-only mode) . . . . .	1079
Figure 358. Master and three independent slaves . . . . .	1080
Figure 359. Multimaster application . . . . .	1081
Figure 360. Hardware/software slave select management . . . . .	1082
Figure 361. Data clock timing diagram . . . . .	1083
Figure 362. Data alignment when data length is not equal to 8-bit or 16-bit . . . . .	1084
Figure 363. Packing data in FIFO for transmission and reception . . . . .	1088
Figure 364. Master full-duplex communication . . . . .	1091
Figure 365. Slave full-duplex communication . . . . .	1092
Figure 366. Master full-duplex communication with CRC . . . . .	1093
Figure 367. Master full-duplex communication in packed mode . . . . .	1094
Figure 368. NSSP pulse generation in Motorola SPI master mode . . . . .	1097
Figure 369. TI mode transfer . . . . .	1098
Figure 370. I <sup>2</sup> S block diagram . . . . .	1101
Figure 371. Full-duplex communication . . . . .	1103
Figure 372. I <sup>2</sup> S Philips protocol waveforms (16/32-bit full accuracy) . . . . .	1104
Figure 373. I <sup>2</sup> S Philips standard waveforms (24-bit frame) . . . . .	1104
Figure 374. Transmitting 0x8EAA33 . . . . .	1105
Figure 375. Receiving 0x8EAA33 . . . . .	1105
Figure 376. I <sup>2</sup> S Philips standard (16-bit extended to 32-bit packet frame) . . . . .	1105
Figure 377. Example of 16-bit data frame extended to 32-bit channel frame . . . . .	1105
Figure 378. MSB Justified 16-bit or 32-bit full-accuracy length . . . . .	1106
Figure 379. MSB justified 24-bit frame length . . . . .	1106
Figure 380. MSB justified 16-bit extended to 32-bit packet frame . . . . .	1107
Figure 381. LSB justified 16-bit or 32-bit full-accuracy . . . . .	1107
Figure 382. LSB justified 24-bit frame length . . . . .	1107
Figure 383. Operations required to transmit 0x3478AE . . . . .	1108
Figure 384. Operations required to receive 0x3478AE . . . . .	1108
Figure 385. LSB justified 16-bit extended to 32-bit packet frame . . . . .	1108
Figure 386. Example of 16-bit data frame extended to 32-bit channel frame . . . . .	1109
Figure 387. PCM standard waveforms (16-bit) . . . . .	1109
Figure 388. PCM standard waveforms (16-bit extended to 32-bit packet frame) . . . . .	1110
Figure 389. Start sequence in master mode . . . . .	1111
Figure 390. Audio sampling frequency definition . . . . .	1112
Figure 391. I <sup>2</sup> S clock generator architecture . . . . .	1112
Figure 392. SAI functional block diagram . . . . .	1135
Figure 393. Audio frame . . . . .	1139
Figure 394. FS role is start of frame + channel side identification (FSDEF = TRIS = 1) . . . . .	1141
Figure 395. FS role is start of frame (FSDEF = 0) . . . . .	1142
Figure 396. Slot size configuration with FBOFF = 0 in SAI_xSLOTR . . . . .	1143
Figure 397. First bit offset . . . . .	1143
Figure 398. Audio block clock generator overview . . . . .	1144
Figure 399. AC'97 audio frame . . . . .	1148
Figure 400. Example of typical AC'97 configuration on devices featuring at least two embedded SAIs (three external AC'97 decoders) . . . . .	1149
Figure 401. SPDIF format . . . . .	1150

---

Figure 402. SAI_xDR register ordering . . . . .	1151
Figure 403. Data companding hardware in an audio block in the SAI . . . . .	1154
Figure 404. Tristate strategy on SD output line on an inactive slot . . . . .	1156
Figure 405. Tristate on output data line in a protocol like I2S . . . . .	1157
Figure 406. Overrun detection error . . . . .	1158
Figure 407. FIFO underrun event . . . . .	1158
Figure 408. SPDIFRX block diagram . . . . .	1189
Figure 409. S/PDIF sub-frame format . . . . .	1190
Figure 410. S/PDIF block format . . . . .	1190
Figure 411. S/PDIF Preambles . . . . .	1191
Figure 412. Channel coding example . . . . .	1191
Figure 413. SPDIFRX decoder . . . . .	1192
Figure 414. Noise filtering and edge detection . . . . .	1193
Figure 415. Thresholds . . . . .	1194
Figure 416. Synchronization flowchart . . . . .	1196
Figure 417. Synchronization process scheduling . . . . .	1197
Figure 418. SPDIFRX States . . . . .	1198
Figure 419. SPDIFRX_FMTx_DR register format . . . . .	1200
Figure 420. Channel/user data format . . . . .	1201
Figure 421. S/PDIF overrun error when RXSTEO = 0 . . . . .	1203
Figure 422. S/PDIF overrun error when RXSTEO = 1 . . . . .	1204
Figure 423. SPDIFRX interface interrupt mapping diagram . . . . .	1205
Figure 424. "No response" and "no data" operations . . . . .	1220
Figure 425. (Multiple) block read operation . . . . .	1220
Figure 426. (Multiple) block write operation . . . . .	1220
Figure 427. Sequential read operation . . . . .	1221
Figure 428. Sequential write operation . . . . .	1221
Figure 429. SDMMC block diagram . . . . .	1221
Figure 430. SDMMC adapter . . . . .	1223
Figure 431. Control unit . . . . .	1224
Figure 432. SDMMC_CK clock dephasing (BYPASS = 0) . . . . .	1225
Figure 433. SDMMC adapter command path . . . . .	1225
Figure 434. Command path state machine (SDMMC) . . . . .	1226
Figure 435. SDMMC command transfer . . . . .	1227
Figure 436. Data path . . . . .	1229
Figure 437. Data path state machine (DPSM) . . . . .	1230
Figure 438. CAN network topology . . . . .	1280
Figure 439. Dual-CAN block diagram . . . . .	1281
Figure 440. bxCAN operating modes . . . . .	1283
Figure 441. bxCAN in silent mode . . . . .	1284
Figure 442. bxCAN in Loop back mode . . . . .	1284
Figure 443. bxCAN in combined mode . . . . .	1285
Figure 444. Transmit mailbox states . . . . .	1287
Figure 445. Receive FIFO states . . . . .	1288
Figure 446. Filter bank scale configuration - Register organization . . . . .	1290
Figure 447. Example of filter numbering . . . . .	1291
Figure 448. Filtering mechanism example . . . . .	1292
Figure 449. CAN error state diagram . . . . .	1294
Figure 450. Bit timing . . . . .	1296
Figure 451. CAN frames . . . . .	1297
Figure 452. Event flags and interrupt generation . . . . .	1298
Figure 453. CAN mailbox registers . . . . .	1310

---

Figure 454. OTG_FS full-speed block diagram . . . . .	1328
Figure 455. OTG_HS high-speed block diagram . . . . .	1329
Figure 456. OTG_FS/OTG_HS A-B device connection . . . . .	1332
Figure 457. OTG_FS/OTG_HS peripheral-only connection . . . . .	1334
Figure 458. OTG_FS/OTG_HS host-only connection . . . . .	1338
Figure 459. SOF connectivity (SOF trigger output to TIM and ITR1 connection) . . . . .	1342
Figure 460. Updating OTG_HFIR dynamically (RLDCTRL = 1) . . . . .	1344
Figure 461. Device-mode FIFO address mapping and AHB FIFO access mapping . . . . .	1345
Figure 462. Host-mode FIFO address mapping and AHB FIFO access mapping . . . . .	1346
Figure 463. Interrupt hierarchy . . . . .	1350
Figure 464. Transmit FIFO write task . . . . .	1456
Figure 465. Receive FIFO read task . . . . .	1457
Figure 466. Normal bulk/control OUT/SETUP . . . . .	1459
Figure 467. Bulk/control IN transactions . . . . .	1463
Figure 468. Normal interrupt OUT . . . . .	1466
Figure 469. Normal interrupt IN . . . . .	1471
Figure 470. Isochronous OUT transactions . . . . .	1473
Figure 471. Isochronous IN transactions . . . . .	1476
Figure 472. Normal bulk/control OUT/SETUP transactions - DMA . . . . .	1478
Figure 473. Normal bulk/control IN transaction - DMA . . . . .	1480
Figure 474. Normal interrupt OUT transactions - DMA mode . . . . .	1481
Figure 475. Normal interrupt IN transactions - DMA mode . . . . .	1482
Figure 476. Normal isochronous OUT transaction - DMA mode . . . . .	1483
Figure 477. Normal isochronous IN transactions - DMA mode . . . . .	1484
Figure 478. Receive FIFO packet read . . . . .	1490
Figure 479. Processing a SETUP packet . . . . .	1492
Figure 480. Bulk OUT transaction . . . . .	1499
Figure 481. TRDT max timing case . . . . .	1509
Figure 482. A-device SRP . . . . .	1510
Figure 483. B-device SRP . . . . .	1511
Figure 484. A-device HNP . . . . .	1512
Figure 485. B-device HNP . . . . .	1514
Figure 486. ETH block diagram . . . . .	1520
Figure 487. SMI interface signals . . . . .	1521
Figure 488. MDIO timing and frame structure - Write cycle . . . . .	1522
Figure 489. MDIO timing and frame structure - Read cycle . . . . .	1523
Figure 490. Media independent interface signals . . . . .	1524
Figure 491. MII clock sources . . . . .	1526
Figure 492. Reduced media-independent interface signals . . . . .	1526
Figure 493. RMII clock sources- . . . . .	1527
Figure 494. Clock scheme . . . . .	1527
Figure 495. Address field format . . . . .	1529
Figure 496. MAC frame format . . . . .	1531
Figure 497. Tagged MAC frame format . . . . .	1531
Figure 498. Transmission bit order . . . . .	1538
Figure 499. Transmission with no collision . . . . .	1538
Figure 500. Transmission with collision . . . . .	1539
Figure 501. Frame transmission in MMI and RMII modes . . . . .	1539
Figure 502. Receive bit order . . . . .	1543
Figure 503. Reception with no error . . . . .	1544
Figure 504. Reception with errors . . . . .	1544
Figure 505. Reception with false carrier indication . . . . .	1544

---

Figure 506. MAC core interrupt masking scheme . . . . .	1545
Figure 507. Wakeup frame filter register . . . . .	1550
Figure 508. Networked time synchronization . . . . .	1553
Figure 509. System time update using the Fine correction method . . . . .	1555
Figure 510. PTP trigger output to TIM2 ITR1 connection . . . . .	1557
Figure 511. PPS output . . . . .	1558
Figure 512. Descriptor ring and chain structure . . . . .	1559
Figure 513. TxDMA operation in default mode . . . . .	1563
Figure 514. TxDMA operation in OSF mode . . . . .	1565
Figure 515. Normal transmit descriptor . . . . .	1566
Figure 516. Enhanced transmit descriptor . . . . .	1572
Figure 517. Receive DMA operation . . . . .	1574
Figure 518. Normal Rx DMA descriptor structure . . . . .	1576
Figure 519. Enhanced receive descriptor field format with IEEE1588 time stamp enabled . . . . .	1582
Figure 520. Interrupt scheme . . . . .	1585
Figure 521. Ethernet MAC remote wakeup frame filter register (ETH_MACRWUFFR) . . . . .	1596
Figure 522. HDMI-CEC block diagram . . . . .	1639
Figure 523. Message structure . . . . .	1640
Figure 524. Blocks . . . . .	1640
Figure 525. Bit timings . . . . .	1641
Figure 526. Signal free time . . . . .	1641
Figure 527. Arbitration phase . . . . .	1642
Figure 528. SFT of three nominal bit periods . . . . .	1642
Figure 529. Error bit timing . . . . .	1643
Figure 530. Error handling . . . . .	1644
Figure 531. TXERR detection . . . . .	1646
Figure 532. Block diagram of STM32 MCU and Cortex®-M7 with FPU -level debug support . . . . .	1656
Figure 533. SWJ debug port . . . . .	1658
Figure 534. JTAG Debug Port connections . . . . .	1662
Figure 535. TPIU block diagram . . . . .	1682

# 1 Documentation conventions

## 1.1 General information

The STM32F75xxx and STM32F74xxx devices have an Arm®<sup>(a)</sup> Cortex®-M7 core.



## 1.2 List of abbreviations for registers

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

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

## 1.3 Register reset value

Bits (binary notation) or bits nibbles (hexadecimal notation) of which the reset value is undefined are marked as X.

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

Bits (binary notation) or bits nibbles (hexadecimal notation) of which the reset value is unmodified are marked as U.

## 1.4 Glossary

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

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

## 1.5 Availability of peripherals

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

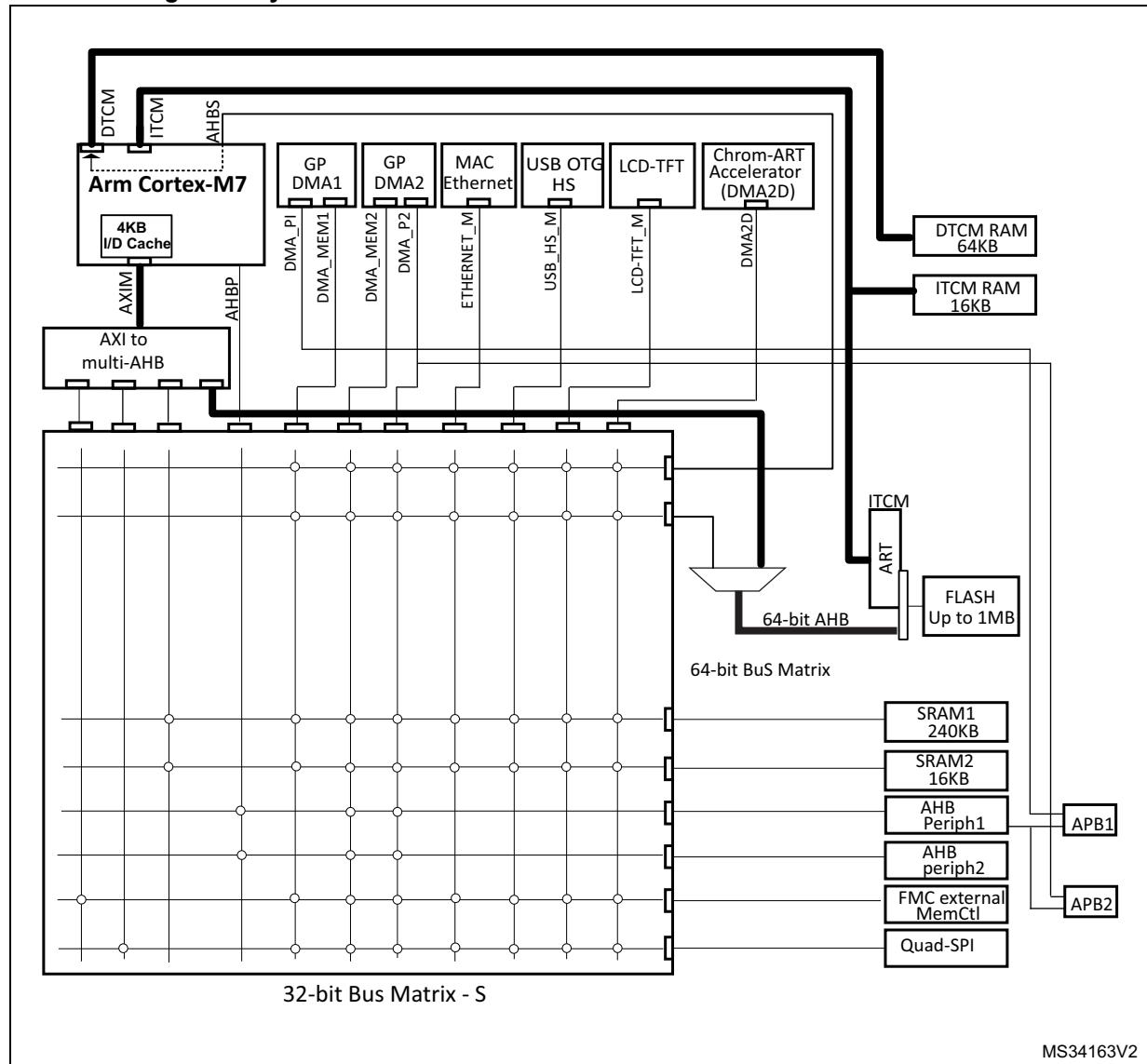
## 2 System and memory overview

### 2.1 System architecture

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

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

**Figure 1. System architecture for STM32F75xxx and STM32F74xxx devices**



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

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

The multi AHB bus matrix interconnects:

- 12 bus masters:
  - 3x32-bit AHB bus Cortex®-M7 AXI Master bus 64-bits, splitted 4 masters through the AXI to AHB bridge.
  - 1x64-bit AHB bus connected to the embedded flash
  - Cortex® -M7 AHB Peripherals bus
  - DMA1 memory bus
  - DMA2 memory bus
  - DMA2 peripheral bus
  - Ethernet DMA bus
  - USB OTG HS DMA bus
  - LCD Controller DMA-bus
  - Chrom-Art Accelerator™ (DMA2D) memory bus
- Eight bus slaves:
  - the embedded Flash on AHB bus (for Flash read/write access, for code execution and data access)
  - Cortex®-M7 AHBS slave interface for DMAs data transfer on DTCM RAM only.
  - Main internal SRAM1 (240 KB)
  - Auxiliary internal SRAM2 (16 KB)
  - AHB1peripherals including AHB to APB bridges and APB peripherals
  - AHB2 peripherals including AHB to APB bridges and APB peripherals
  - FMC
  - Quad SPI

## 2.1.1 Multi AHB BusMatrix

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

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

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

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

The instruction TCM RAM is reserved only for CPU. it is accessed at CPU clock speed with 0 wait states. The architecture is shown in [Figure 1](#).

## 2.1.2 AHB/APB bridges (APB)

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

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

After each device reset, all peripheral clocks are disabled (except for the SRAM, DTCM, ITCM RAM and Flash memory interface). Before using a peripheral its clock has to be enabled in the RCC\_AHBxENR or RCC\_APBxENR register.

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

### 2.1.3 CPU AXIM bus

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

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

### 2.1.4 ITCM bus

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

### 2.1.5 DTCM bus

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

### 2.1.6 CPU AHBS bus

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

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

### 2.1.7 AHB peripheral bus

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

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

### 2.1.8 DMA memory bus

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

### 2.1.9 DMA peripheral bus

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

### 2.1.10 Ethernet DMA bus

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

### 2.1.11 USB OTG HS DMA bus

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

### 2.1.12 LCD-TFT controller DMA bus

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

### 2.1.13 DMA2D bus

This bus connects the DMA2D master interface to the BusMatrix. This bus is used by the DMA2D graphic Accelerator to load/store data to a memory. The targets of this bus are data memories: internal SRAM1, SRAM2 and DTCM (through the AHBS bus of Cortex®-M7), external memories through FMC or Quad SPI, and internal Flash memory.

## 2.2 Memory organization

### 2.2.1 Introduction

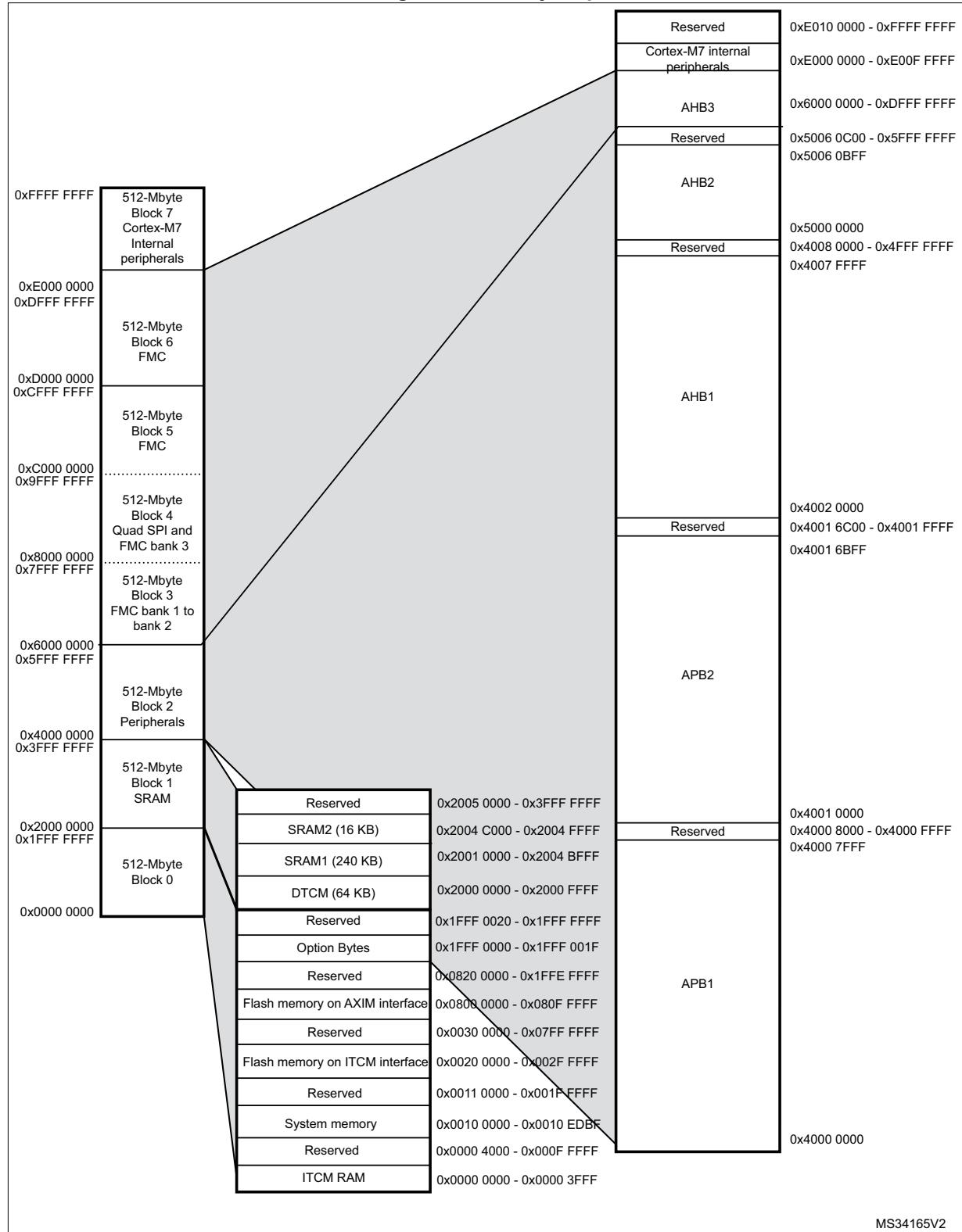
Program memory, data memory, registers and I/O ports are organized within the same linear 4-Gbyte address space.

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

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

## 2.2.2 Memory map and register boundary addresses

Figure 2. Memory map



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

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

**Table 1. STM32F75xxx and STM32F74xxx register boundary addresses**

Boundary address	Peripheral	Bus	Register map
0xA000 1000 - 0xA0001FFF	QuadSPI Control Register	AHB3	<a href="#">Section 14.5.14: QUADSPI register map on page 411</a>
0xA000 0000 - 0xA000 0FFF	FMC control register		<a href="#">Section 13.8.6: FMC register map on page 382</a>
0x5006 0800 - 0x5006 0BFF	RNG	AHB2	<a href="#">Section 19.7.4: RNG register map on page 547</a>
0x5006 0400 - 0x5006 07FF	HASH		<a href="#">Section 21.6.8: HASH register map on page 633</a>
0x5006 0000 - 0x5006 03FF	CRYP		<a href="#">Section 20.7.21: CRYP register map on page 611</a>
0x5005 0000 - 0x5005 03FF	DCMI		<a href="#">Section 17.5.12: DCMI register map on page 503</a>
0x5000 0000 - 0x5003 FFFF	USB OTG FS		<a href="#">Section 37.15.64: OTG_FS/OTG_HS register map on page 1438</a>
0x4004 0000 - 0x4007 FFFF	USB OTG HS	AHB1	<a href="#">Section 37.15.64: OTG_FS/OTG_HS register map on page 1438</a>
0x4002 B000 - 0x4002 BBFF	Chrom-ART (DMA2D)		<a href="#">Section 9.5.26: DMA2D register map on page 288</a>
0x4002 8000 - 0x4002 93FF	ETHERNET MAC		<a href="#">Section 38.8.5: Ethernet register maps on page 1634</a>
0x4002 6400 - 0x4002 67FF	DMA2		<a href="#">Section 8.5.11: DMA register map on page 257</a>
0x4002 6000 - 0x4002 63FF	DMA1		<a href="#">Section 5.3.27: RCC register map on page 198</a>
0x4002 4000 - 0x4002 4FFF	BKPSRAM		<a href="#">Section 3.7.8: Flash interface register map</a>
0x4002 3C00 - 0x4002 3FFF	Flash interface register		<a href="#">Section 5.3.27: RCC register map on page 198</a>
0x4002 3800 - 0x4002 3BFF	RCC		<a href="#">Section 12.4.6: CRC register map on page 308</a>
0x4002 3000 - 0x4002 33FF	CRC		<a href="#">Section 6.4.11: GPIO register map on page 218</a>
0x4002 2800 - 0x4002 2BFF	GPIOK		<a href="#">Section 6.4.11: GPIO register map on page 218</a>
0x4002 2400 - 0x4002 27FF	GPIOJ		
0x4002 2000 - 0x4002 23FF	GPIOI		
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

**Table 1. STM32F75xxx and STM32F74xxx register boundary addresses (continued)**

Boundary address	Peripheral	Bus	Register map
0x4001 6800 - 0x4001 6BFF	LCD-TFT	APB2	<a href="#">Section 18.7.26: LTDC register map on page 534</a>
0x4001 5C00 - 0x4001 5FFF	SAI2		<a href="#">Section 33.5.18: SAI register map on page 1186</a>
0x4001 5800 - 0x4001 5BFF	SAI1		<a href="#">Section 33.5.18: SAI register map on page 1186</a>
0x4001 5400 - 0x4001 57FF	SPI6		<a href="#">Section 32.9.10: SPI/I2S register map on page 1132</a>
0x4001 5000 - 0x4001 53FF	SPI5		<a href="#">Section 24.5.13: TIM10/TIM11/TIM13/TIM14 register map on page 851</a>
0x4001 4800 - 0x4001 4BFF	TIM11		<a href="#">Section 24.4.14: TIM9/TIM12 register map on page 840</a>
0x4001 4400 - 0x4001 47FF	TIM10		<a href="#">Section 11.9.7: EXTI register map on page 301</a>
0x4001 4000 - 0x4001 43FF	TIM9		<a href="#">Section 7.2.8: SYSCFG register maps on page 225</a>
0x4001 3C00 - 0x4001 3FFF	EXTI		<a href="#">Section 32.9.10: SPI/I2S register map on page 1132</a>
0x4001 3800 - 0x4001 3BFF	SYSCFG		<a href="#">Section 32.9.10: SPI/I2S register map on page 1132</a>
0x4001 3400 - 0x4001 37FF	SPI4		<a href="#">Section 35.8.16: SDMMC register map on page 1277</a>
0x4001 3000 - 0x4001 33FF	SPI1		<a href="#">Section 15.13.18: ADC register map on page 457</a>
0x4001 2C00 - 0x4001 2FFF	SDMMC1		<a href="#">Section 31.8.12: USART register map on page 1072</a>
0x4001 2000 - 0x4001 23FF	ADC1 - ADC2 - ADC3		<a href="#">Section 22.4.26: TIM1 register map on page 726</a>
0x4001 1400 - 0x4001 17FF	USART6		
0x4001 1000 - 0x4001 13FF	USART1		
0x4001 0400 - 0x4001 07FF	TIM8		
0x4001 0000 - 0x4001 03FF	TIM1		

**Table 1. STM32F75xxx and STM32F74xxx register boundary addresses (continued)**

Boundary address	Peripheral	Bus	Register map
0x4000 7C00 - 0x4000 7FFF	UART8	APB1	<a href="#">Section 31.8.12: USART register map on page 1072</a>
0x4000 7800 - 0x4000 7BFF	UART7		<a href="#">Section 16.5.15: DAC register map on page 481</a>
0x4000 7400 - 0x4000 77FF	DAC		<a href="#">Section 4.4.4: PWR power control register 2 (PWR_CSR2) on page 130</a>
0x4000 7000 - 0x4000 73FF	PWR		<a href="#">Section 39.7.7: HDMI-CEC register map on page 1655</a>
0x4000 6C00 - 0x4000 6FFF	HDMI-CEC		<a href="#">Section 36.9.5: bxCAN register map on page 1319</a>
0x4000 6800 - 0x4000 6BFF	CAN2		<a href="#">Section 30.9.12: I2C register map on page 1009</a>
0x4000 6400 - 0x4000 67FF	CAN1		<a href="#">Section 30.9.12: I2C register map on page 1009</a>
0x4000 6000 - 0x4000 63FF	I2C4		<a href="#">Section 30.9.12: I2C register map on page 1009</a>
0x4000 5C00 - 0x4000 5FFF	I2C3		<a href="#">Section 30.9.12: I2C register map on page 1009</a>
0x4000 5800 - 0x4000 5BFF	I2C2		<a href="#">Section 30.9.12: I2C register map on page 1009</a>
0x4000 5400 - 0x4000 57FF	I2C1		<a href="#">Section 30.9.12: I2C register map on page 1009</a>
0x4000 5000 - 0x4000 53FF	UART5		<a href="#">Section 31.8.12: USART register map on page 1072</a>
0x4000 4C00 - 0x4000 4FFF	UART4		<a href="#">Section 34.5.10: SPDIFRX interface register map on page 1218</a>
0x4000 4800 - 0x4000 4BFF	USART3		<a href="#">Section 32.9.10: SPI/I2S register map on page 1132</a>
0x4000 4400 - 0x4000 47FF	USART2		<a href="#">Section 27.4.6: IWDG register map on page 895</a>
0x4000 4000 - 0x4000 43FF	SPDIFRX		<a href="#">Section 28.5.4: WWDG register map on page 901</a>
0x4000 3C00 - 0x4000 3FFF	SPI3 / I2S3		<a href="#">Section 29.6.21: RTC register map on page 945</a>
0x4000 3800 - 0x4000 3BFF	SPI2 / I2S2		<a href="#">Section 26.7.9: LPTIM register map on page 886</a>
0x4000 3000 - 0x4000 33FF	IWDG		<a href="#">Section 24.5.13: TIM10/TIM11/TIM13/TIM14 register map on page 851</a>
0x4000 2C00 - 0x4000 2FFF	WWDG		<a href="#">Section 24.4.14: TIM9/TIM12 register map on page 840</a>
0x4000 2800 - 0x4000 2BFF	RTC & BKP Registers		<a href="#">Section 25.4.9: TIMx register map on page 865</a>
0x4000 2400 - 0x4000 27FF	LPTIM1		<a href="#">Section 23.4.24: TIMx register map on page 800</a>
0x4000 2000 - 0x4000 23FF	TIM14		
0x4000 1C00 - 0x4000 1FFF	TIM13		
0x4000 1800 - 0x4000 1BFF	TIM12		
0x4000 1400 - 0x4000 17FF	TIM7		
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		

## 2.3 Embedded SRAM

The STM32F75xxx and STM32F74xxx feature:

- System SRAM up to 320 Kbytes including Data TCM RAM 64 Kbytes
- Instruction RAM (ITCM-RAM) 16 Kbytes.
- 4 Kbytes of backup SRAM (see section 5.1.2: Battery backup domain)

The embedded SRAM is divided into up to four blocks:

- System SRAM:
  - SRAM1 mapped at address 0x2001 0000 and accessible by all AHB masters from AHB bus Matrix.
  - SRAM2 mapped at address 0x2004 C000 and accessible by all AHB masters from AHB bus Matrix.
  - DTCM-RAM on TCM interface (Tightly Coupled Memory interface) mapped at address 0x2000 0000 and accessible by all AHB masters from AHB bus Matrix but through a specific AHB slave bus of the CPU.
- Instruction SRAM:
  - Instruction RAM (ITCM-RAM) mapped at address 0x0000 0000 and accessible only by CPU.

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

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

## 2.4 Flash memory overview

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

The Flash memory is organized as follows:

- A main memory block divided into sectors.
- An information block:
  - System memory from which the device boots in System memory boot mode
  - 1024 OTP (one-time programmable) bytes for user data.
  - Option bytes to configure read and write protection, BOR level, software/hardware watchdog, boot memory base address and reset when the device is in Standby or Stop mode.

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

## 2.5 Boot configuration

In the STM32F75xxx and STM32F74xxx, two different boot areas can be selected through the BOOT pin and the boot base address programmed in the BOOT\_ADD0 and BOOT\_ADD1 option bytes as shown in the [Table 2](#).

**Table 2. Boot modes**

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

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

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

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

The BOOT\_ADD0 and BOOT\_ADD1 address option bytes allows to program any boot memory address from 0x0000 0000 to 0x2004 FFFF which includes:

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

The BOOT\_ADD0 / BOOT\_ADD1 option bytes can be modified after reset in order to boot from any other boot address after next reset.

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

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

When flash level 2 protection is enabled, only boot from Flash (on ITCM or AXIM interface) or system bootloader will be available. If the already programmed boot address in the BOOT\_ADD0 and/or BOOT\_ADD1 option bytes is out of the memory range or RAM address (on ITCM or AXIM) the default fetch will be forced from Flash on ITCM interface at address 0x00200000.

### Embedded bootloader

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

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

## 3 Embedded Flash memory (FLASH)

### 3.1 Introduction

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

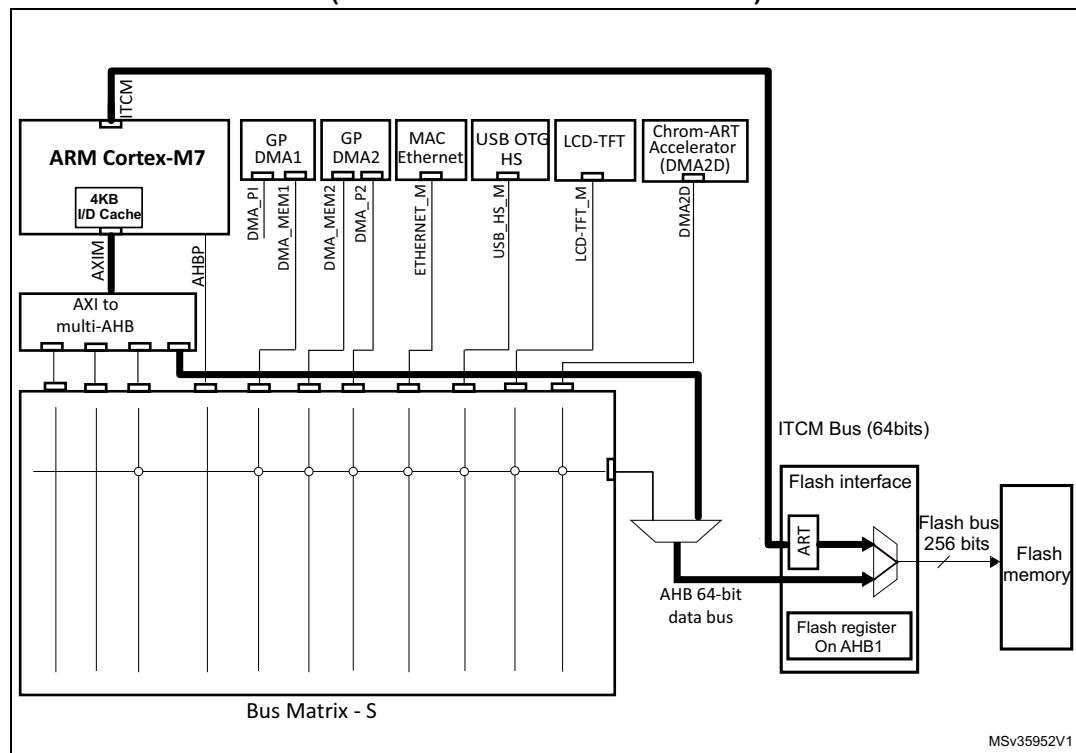
The Flash memory interface accelerates code execution with a system of instruction prefetch and cache lines on ITCM interface (ART Accelerator™).

### 3.2 Flash main features

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- 64 cache lines of 256 bits on ITCM interface (ART Accelerator™)
- Prefetch on TCM instruction code

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

**Figure 3. Flash memory interface connection inside system architecture  
(STM32F75xxx and STM32F74xxx)**



### 3.3 Flash functional description

#### 3.3.1 Flash memory organization

The Flash memory has the following main features:

- Capacity up to 1 Mbyte on STM32F756xx and STM32F74xx devices and 64 Kbytes on STM32F750xx devices
- 256 bits wide data read
- Byte, half-word, word and double word write
- Sector and mass erase

The Flash memory is organized as follows:

- On STM32F756xx and STM32F74xx devices, a main memory block divided into 4 sectors of 32 Kbytes, 1 sector of 128 Kbytes and 3 sectors of 256 Kbytes
- On STM32F750xx devices, a main memory block divided into 2 sectors of 32 Kbytes.
- Information blocks containing:
  - System memory from which the device boots in System memory boot mode
  - 1024 OTP (one-time programmable) for user data
  - The OTP area contains 16 additional bytes used to lock the corresponding OTP data block.
  - Option bytes to configure read and write protection, BOR level, software/hardware watchdog, boot memory base address and reset when the device is in Standby or Stop mode.

The embedded flash has three main interfaces:

- 64-bits ITCM interface:
  - It is connected to the ITCM bus of Cortex-M7 and used for instruction execution and data read access.
  - Write accesses are not supported on ITCM interface
  - Supports a unified 64 cache lines of 256 bits (ART accelerator)
- 64-bits AHB interface:
  - It is connected to the AXI bus of Cortex-M7 through the AHB bus matrix and used for code execution, read and write accesses.
  - DMAs and peripherals DMAs data transfer on Flash are done through the AHB interface whatever the addressed flash interface TCM or AHB.
- 32-bits AHB register interface:
  - It is used for control and status register accesses.

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

**Table 3. STM32F756xx and STM32F74xxx Flash memory organization**

Block	Name	Bloc base address on AXIM interface	Block base address on ICTM interface	Sector size
Main memory block	Sector 0	0x0800 0000 - 0x0800 7FFF	0x0020 0000 - 0x0020 7FFF	32 Kbytes
	Sector 1	0x0800 8000 - 0x0800 FFFF	0x0020 8000 - 0x0020 FFFF	32 Kbytes
	Sector 2	0x0801 0000 - 0x0801 7FFF	0x0021 0000 - 0x0021 7FFF	32 Kbytes
	Sector 3	0x0801 8000 - 0x0801 FFFF	0x0021 8000 - 0x0021 FFFF	32 Kbytes
	Sector 4	0x0802 0000 - 0x0803 FFFF	0x0022 0000 - 0x0023 FFFF	128 Kbytes
	Sector 5	0x0804 0000 - 0x0807 FFFF	0x0024 0000 - 0x0027 FFFF	256 Kbytes
	Sector 6	0x0808 0000 - 0x080B FFFF	0x0028 0000 - 0x002B FFFF	256 Kbytes
	Sector 7	0x080C 0000 - 0x080F FFFF	0x002C 0000 - 0x02F FFFF	256 Kbytes
Information block	System memory	0x1FF0 0000 - 0x1FF0 EDBF	0x0010 0000 - 0x0010 EDBF	60 Kbytes
	OTP	0x1FF0 F000 - 0x1FF0 F41F	0x0010 F000 - 0x0010 F41F	1024 bytes
	Option bytes	0x1FFF 0000 - 0x1FFF 001F	-	32 bytes

**Table 4. STM32F750xx Flash memory organization**

Block	Name	Bloc base address on AXIM interface	Block base address on ICTM interface	Sector size
Main memory block	Sector 0	0x0800 0000 - 0x0800 7FFF	0x0020 0000 - 0x0020 7FFF	32 Kbytes
	Sector 1	0x0800 8000 - 0x0800 FFFF	0x0020 8000 - 0x0020 FFFF	32 Kbytes
Information block	System memory	0x1FF0 0000 - 0x1FF0 EDBF	0x0010 0000 - 0x0010 EDBF	60 Kbytes
	OTP	0x1FF0 F000 - 0x1FF0 F41F	0x0010 F000 - 0x0010 F41F	1024 bytes
	Option bytes	0x1FFF 0000 - 0x1FFF 001F	-	32 bytes

### 3.3.2 Read access latency

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

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

Note:

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

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

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

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

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

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

### Increasing the CPU frequency

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

### Decreasing the CPU frequency

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

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

### Instruction prefetch

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

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

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

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

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

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

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

### 3.3.3 Flash program and erase operations

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

Any attempt to read the Flash memory while it is being written or erased, causes the bus to stall. Read operations are processed correctly once the program operation has completed.

This means that code or data fetches cannot be performed while a write/erase operation is ongoing.

### 3.3.4 Unlocking the Flash control register

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

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

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

The FLASH\_CR register can be locked again by software by setting the LOCK bit in the FLASH\_CR register.

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

### 3.3.5 Maximum program/erase parallelism

The maximum 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, bank or for the whole Flash memory (mass erase). The erase time depends on PSIZE programmed value. For more details on the erase time, refer to the electrical characteristics section of the device datasheet.

*Table 6* provides the correct PSIZE values.

**Table 6. Maximum 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 1.8 V - 2.1 V
Maximum 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.*

### 3.3.6 Flash erase sequences

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

#### Sector Erase

To erase a sector, follow the procedure below:

1. Check that no Flash memory operation is ongoing by checking the BSY bit in the FLASH\_SR register
2. Set the SER bit and select the sector number of the user memory block to erase (SNB) in the FLASH\_CR register.
3. Set the STRT bit in the FLASH\_CR register
4. Wait for the BSY bit to be cleared

#### Mass Erase

To perform Mass Erase, the following sequence is recommended:

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

*Note: If MERx and SER bits are both set in the FLASH\_CR register, mass erase is performed.*

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

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

### 3.3.7 Flash programming sequences

#### Standard programming

The Flash memory programming sequence is as follows:

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

- Note:** Successive write operations are possible without the need of an erase operation when changing bits from '1' to '0'. Writing '1' requires a Flash memory erase operation.  
If an erase and a program operation are requested simultaneously, the erase operation is performed first.
- Note:** After performing a data write operation and before polling the BSY bit to be cleared, the software can issue a DSB instruction to guarantee the completion of a previous data write operation.

### Programming errors

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

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

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

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

### 3.3.8 Flash Interrupts

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

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

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

In this case, if the error interrupt enable bit (ERRIE) is set in the FLASH\_CR register, an interrupt is generated and the operation error bit (OPERR) is set in the FLASH\_SR register.

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

**Table 7. Flash interrupt request**

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

## 3.4 FLASH Option bytes

### 3.4.1 Option bytes description

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

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

- *Flash option control register (FLASH\_OPTCR)*
- *Flash option control register (FLASH\_OPTCR1)*

**Table 8. Option byte organization**

AXI address	[63:16]	[15:0]
0x1FFF 0000	Reserved	ROP & user option bytes ( <b>RDP &amp; USER</b> )
0x1FFF 0008	Reserved	IWDG_STOP, IWDG_STBY and Write protection nWRP (sector 0 to 7) and user option bytes
0x1FFF 0010	Reserved	BOOT_ADD0
0x1FFF 0018	Reserved	BOOT_ADD1

### User and read protection option bytes

Memory address: 0x1FFF 0000

ST programmed value: 0x5500AAFF

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

Bits 31:13 Not used.

#### Bits 15:8 **RDP**: Read Out Protection

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

0xAA: Level0, no Protection

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

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

#### Bit 7 **nRST\_STDBY**

0: Reset generated when entering Standby mode.

1: No reset generated.

#### Bit 6 **nRST\_STOP**

0: Reset generated when entering Stop mode.

1: No reset generated.

#### Bit 5 **IWDG\_SW**: Independant watchdog selection

0: Hardware independant watchdog.

1: Software independant watchdog.

#### Bit 4 **WWDG\_SW**: Window watchdog selection

0: Hardware window watchdog.

1: Software window watchdog.

#### Bits 3:2 **BOR\_LEV**: BOR reset Level

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

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

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

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

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

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

Bits 1:0 Not used

### User and write protection option bytes

Memory address: 0x1FFF 0008

ST programmed value: 0x0000FFFF

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

Bits 31:16 Not used.

Bit 15 **IWDG\_STOP:** Independent watchdog counter freeze in stop mode

0: Freeze IWDG counter in stop mode.

1: IWDG counter active in stop mode.

Bit 14 **IWDG\_STDBY:** Independent watchdog counter freeze in Standby mode

1: IWDG counter active in standby mode.

Bits 13:8 Not used

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

0: Write protection active on sector i.

1: Write protection not active on sector i.

### Boot address option bytes when Boot pin =0

Memory address: 0x1FFF 0010

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

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

Bits 31:16 Not used.

Bits 15:0 **BOOT\_ADD0[15:0]:** Boot memory base address when Boot pin =0

BOOT\_ADD0[15:0] correspond to address [29:14],

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

Example:

BOOT\_ADD0 = 0x0000: Boot from ITCM RAM (0x0000 0000)

BOOT\_ADD0 = 0x0040: Boot from system memory bootloader (0x0010 0000)

BOOT\_ADD0 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)

BOOT\_ADD0 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)

BOOT\_ADD0 = 0x8000: Boot from DTCM RAM (0x2000 0000)

BOOT\_ADD0 = 0x8004: Boot from SRAM1 (0x2001 0000)

BOOT\_ADD0 = 0x8013: Boot from SRAM2 (0x2004 C000)

### Boot address option bytes when Boot pin =1

Memory address: 0x1FFF 0018

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

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

Bits 31:16    **Not used**

Bits 15:0    **BOOT\_ADD1[15:0]**: Boot memory base address when Boot pin =1

BOOT\_ADD1[15:0] correspond to address [29:14],

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

Example:

BOOT\_ADD1 = 0x0000: Boot from ITCM RAM(0x0000 0000)

BOOT\_ADD1 = 0x0040: Boot from system memory bootloader (0x0010 0000)

BOOT\_ADD1 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)

BOOT\_ADD1 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)

BOOT\_ADD1 = 0x8000: Boot from DTCM RAM (0x2000 0000)

BOOT\_ADD1 = 0x8004: Boot from SRAM1 (0x2001 0000)

BOOT\_ADD1 = 0x8013: Boot from SRAM2 (0x2004 C000)

### 3.4.2 Option bytes programming

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

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

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

#### Modifying user option bytes

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

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

Note:

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

Note:

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

## 3.5 FLASH memory protection

### 3.5.1 Read protection (RDP)

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

- Level 0: no read protection

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

- Level 1: read protection enabled

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

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

When Level 1 is active, programming the protection option byte (RDP) to Level 0 causes the Flash memory and the backup SRAM to be mass-erased. As a result the user code area is cleared before the read protection is removed. The mass erase only erases the user code area. The other option bytes including write protections remain unchanged from before the mass-erase operation. 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 disabled

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

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

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

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

**Note:** *If the read protection is set while the debugger is still connected (or had been connected since the last power on) through JTAG/SWD, apply a POR (power-on reset) instead of a system reset. If the read protection is programmed through software, don't set the*

*OBL\_LAUNCH bit (FLASH\_CR register) but perform a POR to reload the option byte. This can be done with a transition Standby (or Shutdown) mode followed by a wake-up.*

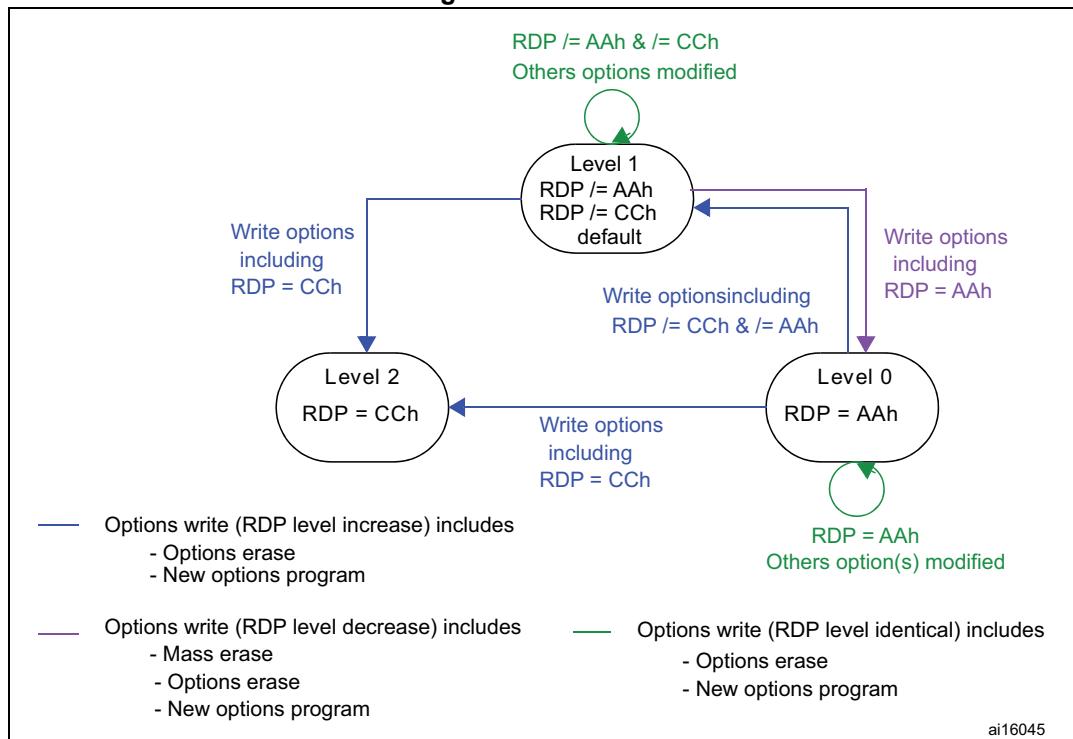
**Table 9. Access versus read protection level**

Memory area	Protection Level	Debug features, Boot from RAM or from System memory bootloader			Booting from Flash memory		
		Read	Write	Erase	Read	Write	Erase
Main Flash Memory and Backup SRAM	Level 1	NO		NO <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 and backup SRAM are only erased when the RDP changes from level 1 to 0. The OTP area remains unchanged.

*Figure 4* shows how to go from one RDP level to another.

**Figure 4. RDP levels**



### 3.5.2 Write protections

Up to 8 user sectors in Flash memory can be protected against unwanted write operations due to loss of program counter contexts. When the non-write protection nWRPi bit ( $0 \leq i \leq 7$ ) in the FLASH\_OPTCR or FLASH\_OPTCR1 registers is low, the corresponding sector

cannot be erased or programmed. Consequently, a mass erase cannot be performed if one of the sectors is write-protected.

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

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

### Write protection error flag

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

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

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

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

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

## 3.6 One-time programmable bytes

Table 10 shows the organization of the one-time programmable (OTP) part of the OTP area.

Table 10. OTP area organization

OTP Block	[255:224]	[223:193]	[192:161]	[160:128]	[127:96]	[95:64]	[63:32]	[31:0]	Address byte 0
0	OTP0	OTP0	OTP0	OTP0	OTP0	OTP0	OTP0	OTP0	0x1FF0 F000
	OTP0	OTP0	OTP0	OTP0	OTP0	OTP0	OTP0	OTP0	0x1FF0 F020
1	OTP1	OTP1	OTP1	OTP1	OTP1	OTP1	OTP1	OTP1	0x1FF0 F040
	OTP1	OTP1	OTP1	OTP1	OTP1	OTP1	OTP1	OTP1	0x1FF0 F060
-	-	-	-	-	-	-	-	-	-

Table 10. OTP area organization (continued)

OTP Block	[255:224]	[223:193]	[192:161]	[160:128]	[127:96]	[95:64]	[63:32]	[31:0]	Address byte 0
14	OTP14	OTP14	OTP14	OTP14	OTP14	OTP14	OTP14	OTP14	0x1FF0 F380
	OTP14	OTP14	OTP14	OTP14	OTP14	OTP14	OTP14	OTP14	0x1FF0 F3A0
15	OTP15	OTP15	OTP15	OTP15	OTP15	OTP15	OTP15	OTP15	0x1FF0 F3C0
	OTP15	OTP15	OTP15	OTP15	OTP15	OTP15	OTP15	OTP15	0x1FF0 F3E0
Lock block	reserved	reserved	reserved	reserved	LOCK15... LOCKB12	LOCK11... LOCKB8	LOCK7... LOCKB4	LOCK3... LOCKB0	0x1FF0 F400

The OTP area is divided into 16 OTP data blocks of 64 bytes and one lock OTP block of 16 bytes. The OTP data and lock blocks cannot be erased. The lock block contains 16 bytes LOCKBi ( $0 \leq i \leq 15$ ) to lock the corresponding OTP data block (blocks 0 to 15). Each OTP data block can be programmed until the value 0x00 is programmed in the corresponding OTP lock byte. The lock bytes must only contain 0x00 and 0xFF values, otherwise the OTP bytes might not be taken into account correctly.

## 3.7 FLASH registers

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

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

Address offset: 0x000

Reset value: 0x0000 0000

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

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

Bits 31:12 Reserved, must be kept cleared.

Bit 11 **ARTRST**: ART Accelerator reset

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

Bit 10 Reserved, must be kept cleared.

Bit 9 **ARTEN**: ART Accelerator Enable

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

Bit 8 **PRFTEN**: Prefetch enable

0: Prefetch is disabled

1: Prefetch is enabled

Bits 7:4 Reserved, must be kept cleared.

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

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

0000: Zero wait state

0001: One wait state

0010: Two wait states

-

-

-

1110: Fourteen wait states

1111: Fifteen wait states

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

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

Address offset: 0x04

Reset value: 0x0000 0000

Access: no wait state, word access

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

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

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

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

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

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

Address offset: 0x08

Reset value: 0x0000 0000

Access: no wait state, word access

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

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

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

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

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

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

Address offset: 0x0C

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	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.	ERSERR	PGPERR	PGAERR	WRPERR	Res.	Res.	OPERR	EOP							
								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 15:8 Reserved, must be kept cleared.

Bit 7 **ERSERR**: Erase Sequence Error

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

Cleared by writing 1.

Bit 6 **PGPERR**: Programming parallelism error

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

Cleared by writing 1.

Bit 5 **PGAERR**: Programming alignment error

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

Cleared by writing 1.

Bit 4 **WRPERR**: Write protection error

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

Cleared by writing 1.

Bits 3:2 Reserved, must be kept cleared.

Bit 1 **OPERR**: Operation error

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

Bit 0 **EOP**: End of operation

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

Cleared by writing a 1.

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

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

Address offset: 0x10

Reset value: 0x8000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOCK	Res.	Res.	Res.	Res.	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[3:0]**: Sector number

These bits select the sector to erase.

- 0000 sector 0
- 0001 sector 1
- ...

0111 sector 7

Others not allowed

Bit 2 **MER**: Mass Erase

Erase activated for all user sectors.

Bit 1 **SER**: Sector Erase

Sector Erase activated.

Bit 0 **PG**: Programming

Flash programming activated.

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

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

Address offset: 0x14

Reset value: 0xC0FFAAFD. The option bytes are loaded with values from Flash memory at reset release.

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

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

- Bit 31 **IWDG\_STOP**: Independent watchdog counter freeze in Stop mode  
 0: Freeze IWDG counter in STOP mode.  
 1: IWDG counter active in STOP mode.
- Bit 30 **IWDG\_STDBY**: Independent watchdog counter freeze in standby mode  
 0: Freeze IWDG counter in standby mode.  
 1: IWDG counter active in standby mode.
- Bits 29:24 Reserved, must be kept cleared.
- Bits 23:16 **nWRP[7:0]**: Not write protect  
 These bits contain the value of the write-protection option bytes for sectors 0 to 7 after reset. They can be written to program a new write-protect into Flash memory.  
 0: Write protection active on sector i  
 1: Write protection not active on sector i
- Bits 15:8 **RDP[7:0]**: Read protect  
 These bits contain the value of the read-protection option level after reset. They can be written to program a new read protection value into Flash memory.  
 0xAA: Level 0, read protection not active  
 0xCC: Level 2, chip read protection active  
 Others: Level 1, read protection of memories active
- Bits 7:4 **USER**: User option bytes  
 These bits contain the value of the user option byte after reset. They can be written to program a new user option byte value into Flash memory.  
 Bit 7: nRST\_STDBY  
 Bit 6: nRST\_STOP  
 Bit 5: IWDG\_SW  
 Bit 4: WWGD\_SW
- Bits 3:2 **BORLEV[1:0]**: BOR reset Level  
 These bits contain the supply level threshold that activates/releases the reset. They can be written to program a new BOR level. By default, BOR is off. When the supply voltage ( $V_{DD}$ ) drops below the selected BOR level, a device reset is generated.  
 00: BOR Level 3 (VBOR3), brownout threshold level 3  
 01: BOR Level 2 (VBOR2), brownout threshold level 2  
 10: BOR Level 1 (VBOR1), brownout threshold level 1  
 11: BOR off, POR/PDR reset threshold level is applied
- Note: For full details on BOR characteristics, refer to the “Electrical characteristics” section of the product datasheet.*
- Bit 1 **OPTSTRT**: Option start  
 This bit triggers a user option operation when set. It is set only by software and cleared when the BSY bit is cleared.
- Bit 0 **OPTLOCK**: Option lock  
 Write to 1 only. When this bit is set, it indicates that the FLASH\_OPTCR register is locked. This bit is cleared by hardware after detecting the unlock sequence.  
 In the event of an unsuccessful unlock operation, this bit remains set until the next reset.
- Note: When modifying the IWDG\_SW, IWDG\_STOP or IWDG\_STDBY option byte, a system reset is required to make the change effective.*

### 3.7.7 Flash option control register (FLASH\_OPTCR1)

The FLASH\_OPTCR1 register is used to modify the user option bytes.

Address offset: 0x18

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

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

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

Bits 31:16 **BOOT\_ADD1[15:0]**: Boot base address when Boot pin =1

BOOT\_ADD1[15:0] correspond to address [29:14],

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

Example:

BOOT\_ADD1 = 0x0000: Boot from ITCM RAM (0x0000 0000)

BOOT\_ADD1 = 0x0040: Boot from System memory bootloader (0x0010 0000)

BOOT\_ADD1 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)

BOOT\_ADD1 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)

BOOT\_ADD1 = 0x8000: Boot from DTCM RAM (0x2000 0000)

BOOT\_ADD1 = 0x8004: Boot from SRAM1 (0x2001 0000)

BOOT\_ADD1 = 0x8013: Boot from SRAM2 (0x2004 C000)

Bits 15:0 **BOOT\_ADD0[15:0]**: Boot base address when Boot pin =0

BOOT\_ADD0[15:0] correspond to address [29:14],

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

Example:

BOOT\_ADD0 = 0x0000: Boot from ITCM RAM (0x0000 0000)

BOOT\_ADD0 = 0x0040: Boot from System memory bootloader (0x0010 0000)

BOOT\_ADD0 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)

BOOT\_ADD0 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)

BOOT\_ADD0 = 0x8000: Boot from DTCM RAM (0x2000 0000)

BOOT\_ADD0 = 0x8004: Boot from SRAM1 (0x2001 0000)

BOOT\_ADD0 = 0x8013: Boot from SRAM2 (0x2004 C000)

### 3.7.8 Flash interface register map

Table 11. Flash register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<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.	LATENCY[3:0]			
	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			
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			
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.			
	Reset value																																
0x10	<b>FLASH_CR</b>	1	IWDG_STOP	LOCK	Res.	0	STRT	0	BSY	Res.																							
	Reset value	1	1	IWDG_STDBY	Res.	0	0	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	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x18	<b>FLASH_OPTCR1</b>	BOOT_ADD1[15:0]												BOOT_ADD0[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 4 Power controller (PWR)

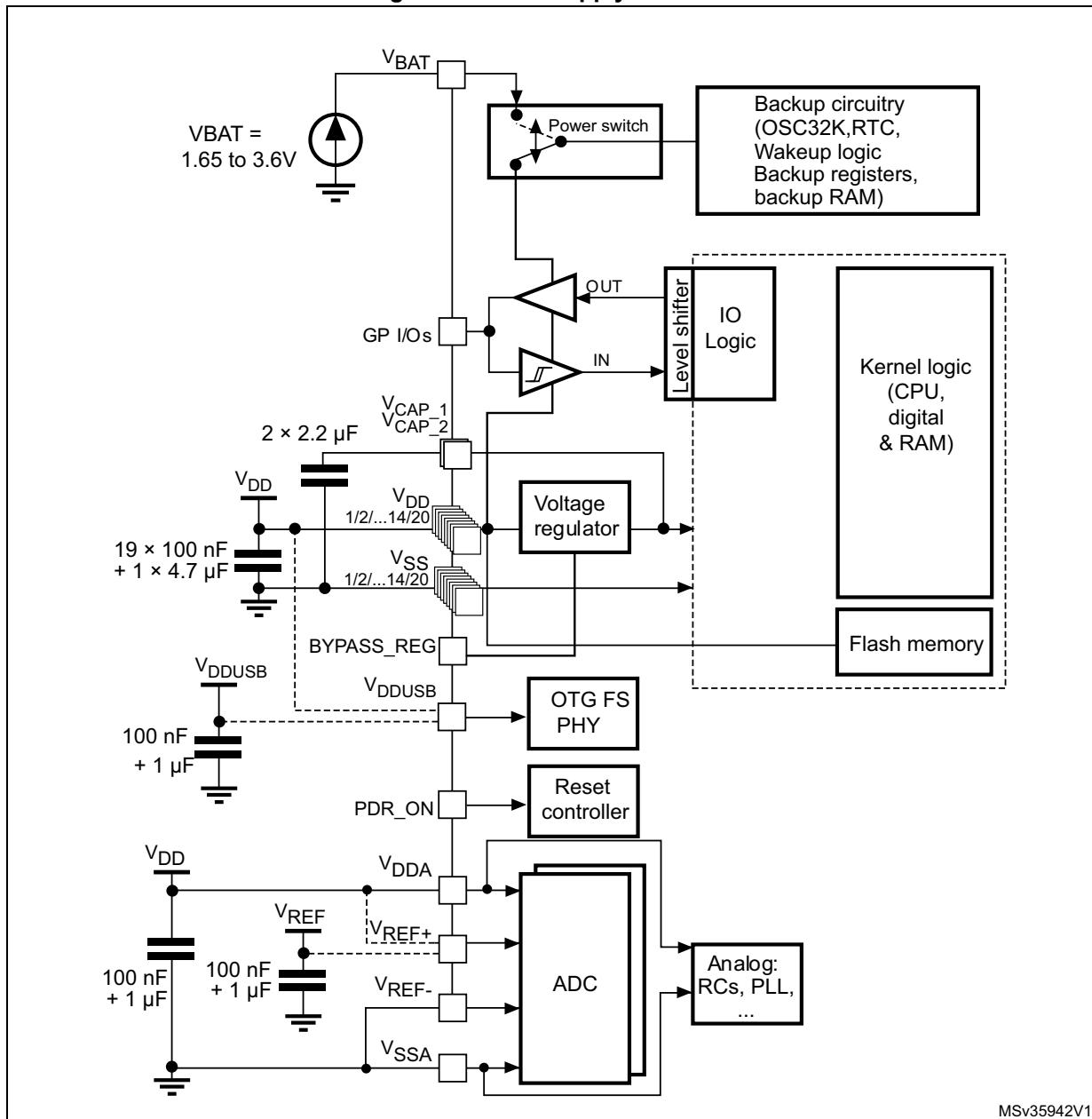
### 4.1 Power supplies

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

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

*Note:* *Depending on the operating power supply range, some peripheral may be used with limited functionality and performance. For more details refer to section "General operating conditions" in STM32F75xxx and STM32F74xxx datasheets.*

Figure 5. Power supply overview



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

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

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

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

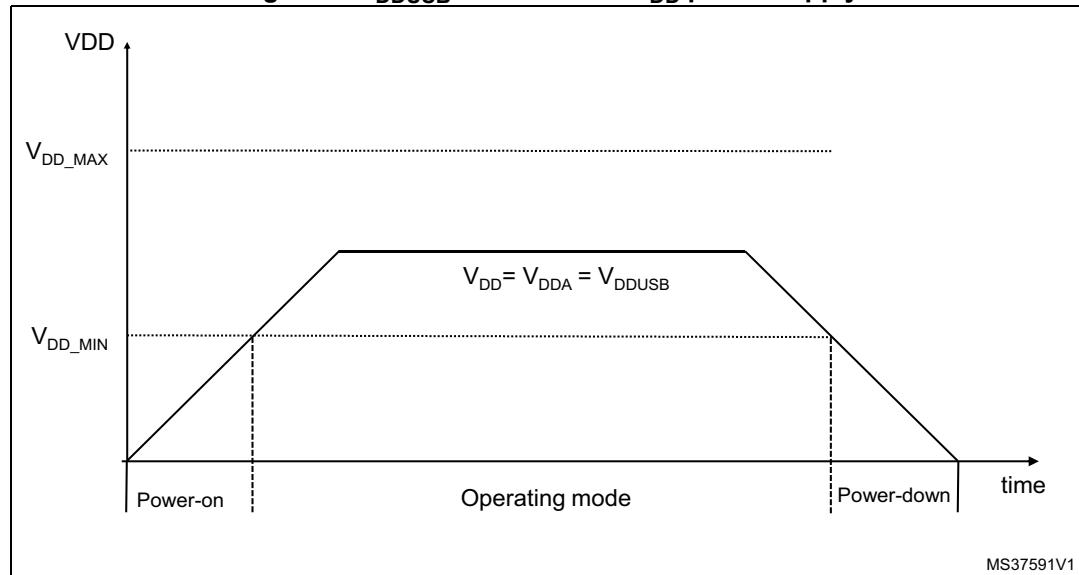
To ensure a better accuracy of low voltage inputs, the user can connect a separate external reference voltage ADC input on  $V_{REF}$ . The voltage on  $V_{REF}$  ranges from 1.8 V to  $V_{DDA}$ .

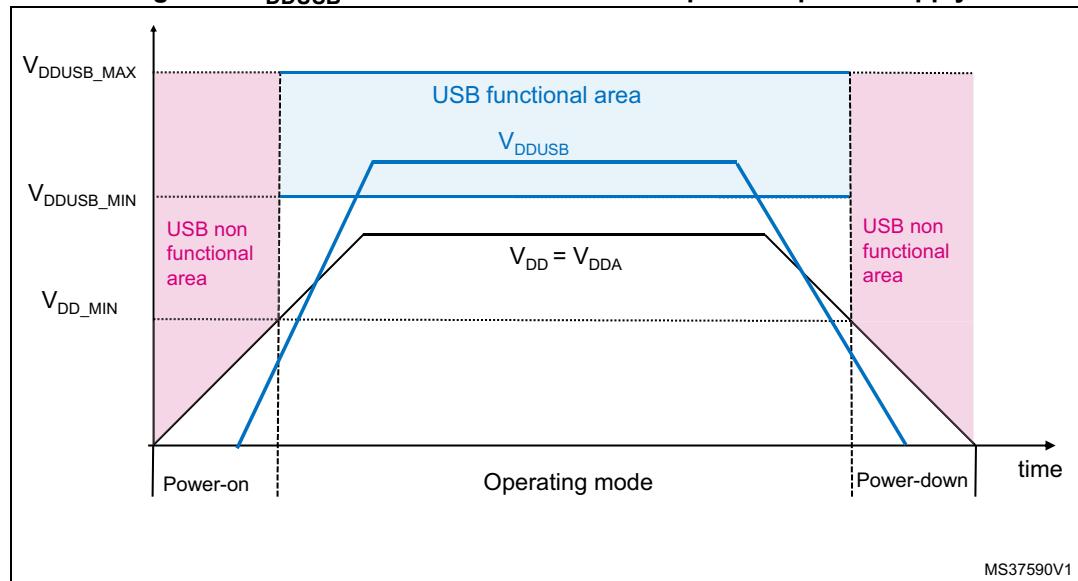
#### 4.1.2 Independent USB transceivers supply

The  $V_{DDUSB}$  is an independent USB power supply for full speed transceivers (USB OTG FS and USB OTG HS in FS mode). It can be connected either to  $V_{DD}$  or an external independent power supply (3.0 to 3.6V) for USB transceivers (refer [Figure 6](#) and [Figure 7](#)). For example, when the device is powered at 1.8V, an independent power supply 3.3V can be connected to  $V_{DDUSB}$ . When the  $V_{DDUSB}$  is connected to a separated power supply, it is independent from  $V_{DD}$  or  $V_{DDA}$  but it must be the last supply to be provided and the first to disappear. The following conditions  $V_{DDUSB}$  must be respected:

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

**Figure 6.  $V_{DDUSB}$  connected to  $V_{DD}$  power supply**



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

#### 4.1.3 Battery backup domain

##### Backup domain description

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

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

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

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

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

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

If the power supply/battery connected to the  $V_{BAT}$  pin cannot support this current injection, it is strongly recommended to

---

**connect an external low-drop diode between this power supply and the V<sub>BAT</sub> pin.**

---

If no external battery is used in the application, it is recommended to connect the V<sub>BAT</sub> externally to V<sub>DD</sub> through a 100 nF external ceramic capacitor.

When the backup domain is supplied by V<sub>DD</sub> (analog switch connected to V<sub>DD</sub>), the following functions are available:

- PC14 and PC15 can be used as either GPIO or LSE pins
- PC13 and PI8 can be used as a GPIO pin (refer to [Table 172: RTC pin PC13 configuration](#) and [Table 173: RTC pin PI8 configuration](#) for more details about these pins configuration)

**Note:** *Due to the fact that the switch only sinks a limited amount of current (3 mA), the use of GPIOs PI8 and PC13 to PC15 are restricted: the speed has to be limited to 2 MHz with a maximum load of 30 pF and these I/Os must not be used as a current source (e.g. to drive an LED).*

When the backup domain is supplied by V<sub>BAT</sub> (analog switch connected to V<sub>BAT</sub> because V<sub>DD</sub> is not present), the following functions are available:

- PC14 and PC15 can be used as LSE pins only
- PC13 can be used as tamper pin (TAMP1)
- PI8 can be used as tamper pin (TAMP2)

### Backup domain access

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

- Access to the RTC and RTC backup registers
- 1. Enable the power interface clock by setting the PWREN bits in the RCC\_APB1ENR register (see [Section 5.3.13](#))
- 2. Set the DBP bit in the [PWR power control register \(PWR\\_CR1\)](#) to enable access to the backup domain
- 3. Select the RTC clock source: see [Section 5.2.8: RTC/AWU clock](#)
- 4. Enable the RTC clock by programming the RTCEN [15] bit in the [RCC backup domain control register \(RCC\\_BDCR\)](#)
- Access to the backup SRAM
- 1. Enable the power interface clock by setting the PWREN bits in the RCC\_APB1ENR register (see [Section 5.3.13](#))
- 2. Set the DBP bit in the [PWR power control register \(PWR\\_CR1\)](#) to enable access to the backup domain
- 3. Enable the backup SRAM clock by setting BKPSRAMEN bit in the [RCC AHB1 peripheral clock register \(RCC\\_AHB1ENR\)](#).

### RTC and RTC backup registers

The real-time clock (RTC) is an independent BCD timer/counter. The RTC provides a time-of-day clock/calendar, two programmable alarm interrupts, and a periodic programmable wakeup flag with interrupt capability. The RTC contains 32 backup data registers (128 bytes)

which are reset when a tamper detection event occurs. For more details refer to [Section 29: Real-time clock \(RTC\)](#).

### Backup SRAM

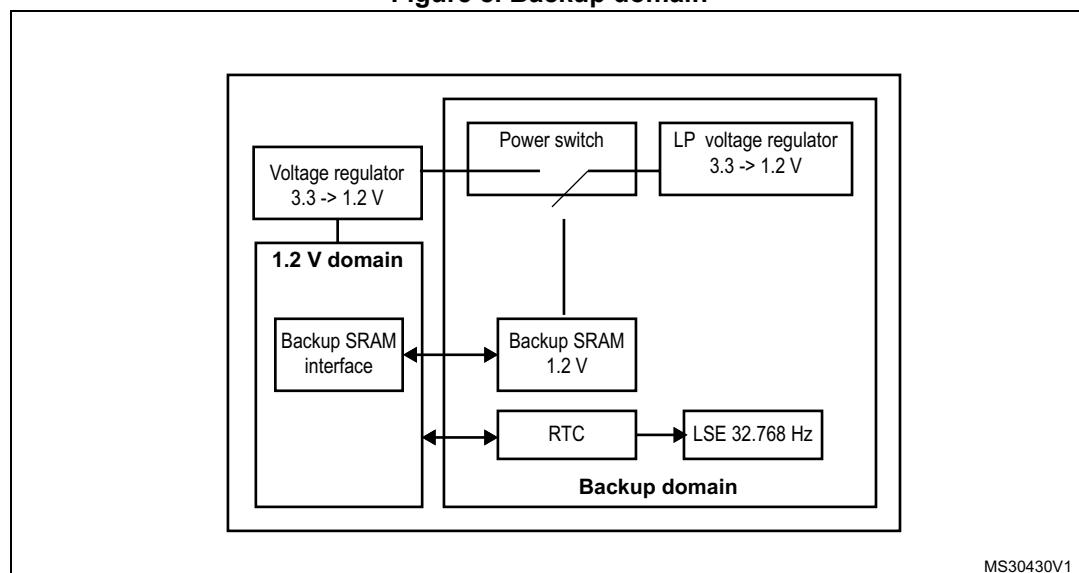
The backup domain includes 4 Kbytes of backup SRAM addressed in 32-bit, 16-bit or 8-bit mode. Its content is retained even in Standby or  $V_{BAT}$  mode when the low-power backup regulator is enabled. It can be considered as an internal EEPROM when  $V_{BAT}$  is always present.

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

When the backup domain is supplied by  $V_{BAT}$  (analog switch connected to  $V_{BAT}$  because  $V_{DD}$  is not present), the backup SRAM is powered by a dedicated low-power regulator. This regulator can be ON or OFF depending whether the application needs the backup SRAM function in Standby and  $V_{BAT}$  modes or not. The power-down of this regulator is controlled by a dedicated bit, the BRE control bit of the PWR\_CSR1 register (see [Section 4.4.2: PWR power control/status register \(PWR\\_CSR1\)](#)).

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

**Figure 8. Backup domain**



MS30430V1

### 4.1.4 Voltage regulator

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

This voltage regulator requires two external capacitors to be connected to two dedicated pins,  $V_{CAP\_1}$  and  $V_{CAP\_2}$  available in all packages. Specific pins must be connected either to  $V_{SS}$  or  $V_{DD}$  to activate or deactivate the voltage regulator. These pins depend on the package.

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

- In **Run mode**, the main regulator supplies full power to the 1.2 V domain (core, memories and digital peripherals). In this mode, the regulator output voltage (around 1.2 V) can be scaled by software to different voltage values (scale 1, scale 2, and scale 3 can be configured through VOS[1:0] bits of the PWR\_CR1 register). The scale can be modified only when the PLL is OFF and the HSI or HSE clock source is selected as system clock source. The new value programmed is active only when the PLL is ON. When the PLL is OFF, the voltage scale 3 is automatically selected.

The voltage scaling allows optimizing the power consumption when the device is clocked below the maximum system frequency. After exit from Stop mode, the voltage scale 3 is automatically selected.(see [Section 4.4.1: PWR power control register \(PWR\\_CR1\)](#)).

2 operating modes are available:

- **Normal mode**: The CPU and core logic operate at maximum frequency at a given voltage scaling (scale 1, scale 2 or scale 3)
- **Over-drive mode**: This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for the voltage scaling scale 1 and scale 2.
- In **Stop mode**: the main regulator or low-power regulator supplies a low-power voltage to the 1.2V domain, thus preserving the content of registers and internal SRAM. The voltage regulator can be put either in main regulator mode (MR) or in low-power mode (LPR). Both modes can be configured by software as follows:
  - **Normal mode**: the 1.2 V domain is preserved in nominal leakage mode. It is the default mode when the main regulator (MR) or the low-power regulator (LPR) is enabled.
  - **Under-drive mode**: the 1.2 V domain is preserved in reduced leakage mode. This mode is only available with the main regulator or the low-power regulator mode (see [Table 12](#)).
- In **Standby mode**: the regulator is powered down. The content of the registers and SRAM are lost except for the Standby circuitry and the backup domain.

*Note:*

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

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

**Table 12. 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	-
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\_CR1 register to enable the Over-drive mode and wait for the ODRDY flag to be set in the PWR\_CSR1 register.
4. Set the ODSW bit in the PWR\_CR1 register to switch the voltage regulator from Normal mode to Over-drive mode. The System will be stalled during the switch but the PLL clock system will be still running during locking phase.
5. Wait for the ODSWRDY flag in the PWR\_CSR1 to be set.
6. Select the required Flash latency as well as AHB and APB prescalers.
7. Wait for PLL lock.
8. Switch the system clock to the PLL.
9. Enable the peripherals that are not generated by the System PLL (I2S clock, LCD-TFT clock, SAI1 clock, USB\_48MHz clock....).

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

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

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

## Exiting from Over-drive mode

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

- By resetting simultaneously the ODEN and ODSW bits in the PWR\_CR1 register (sequence 1)
- By resetting first the ODSW bit to switch the voltage regulator to Normal mode and then resetting the ODEN bit to disable the Over-drive mode (sequence 2).

Example of sequence 1:

1. Select HSI or HSE as system clock source.
2. Disable the peripheral clocks that are not generated by the System PLL (I2S clock, LCD-TFT clock, SAI1 clock, USB\_48MHz clock,...)
3. Reset simultaneously the ODEN and the ODSW bits in the PWR\_CR1 register to switch back the voltage regulator to Normal mode and disable the Over-drive mode.
4. Wait for the ODWRDY flag of PWR\_CSR1 to be reset.

Example of sequence 2:

1. Select HSI or HSE as system clock source.
2. Disable the peripheral clocks that are not generated by the System PLL (I2S clock, LCD-TFT clock, SAI1 clock, USB\_48MHz clock,...).
3. Reset the ODSW bit in the PWR\_CR1 register to switch back the voltage regulator to Normal mode. The system clock is stalled during voltage switching.
4. Wait for the ODWRDY flag of PWR\_CSR1 to be reset.
5. Reset the ODEN bit in the PWR\_CR1 register to disable the Over-drive mode.

Note:

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

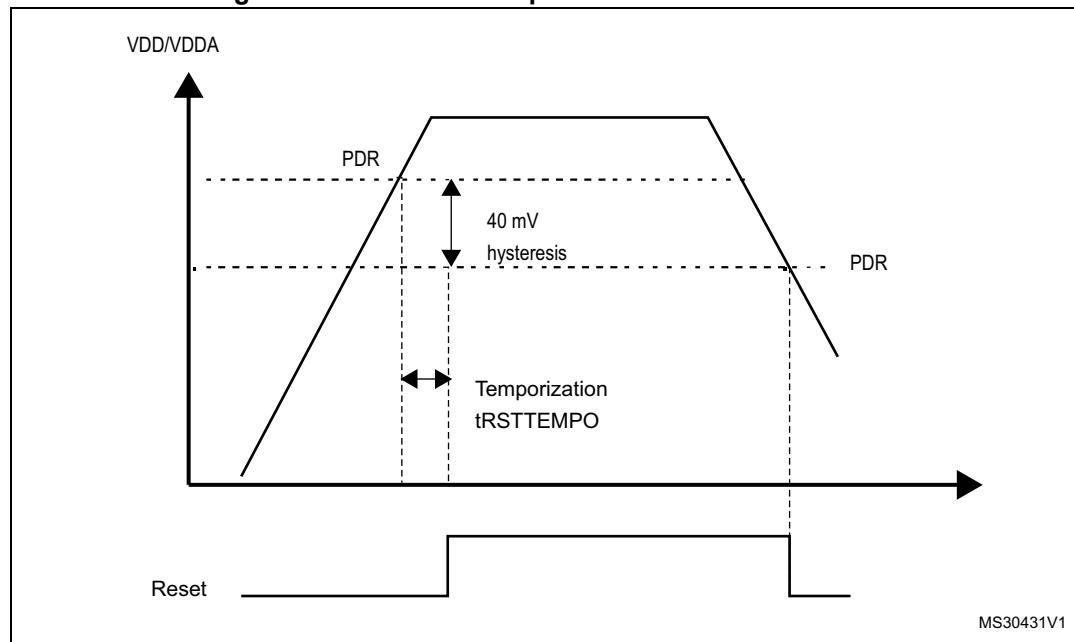
## 4.2 Power supply supervisor

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

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

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

**Figure 9. Power-on reset/power-down reset waveform**



#### 4.2.2 Brownout reset (BOR)

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

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

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

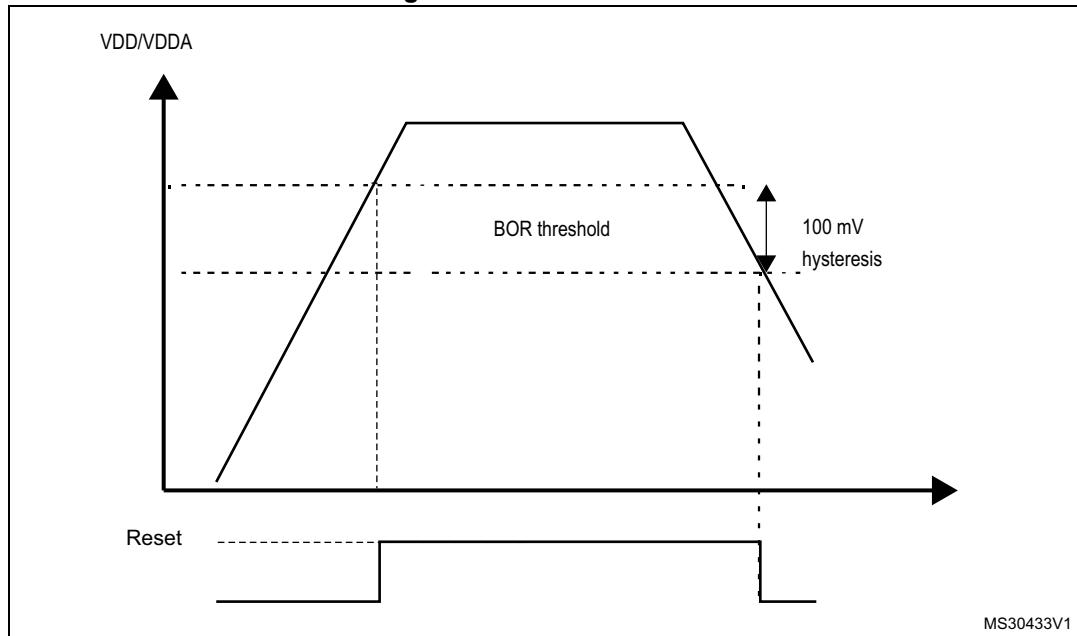
**Note:** *For full details about BOR characteristics, refer to the "Electrical characteristics" section in the device datasheet.*

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

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

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

**Figure 10. BOR thresholds**



#### 4.2.3 Programmable voltage detector (PVD)

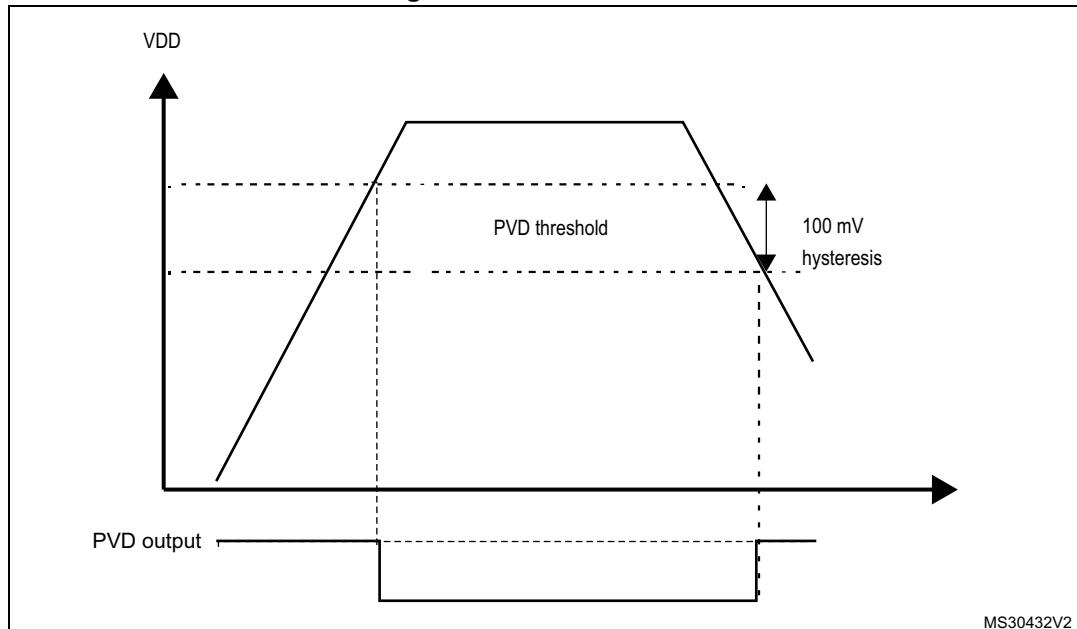
The user can use the PVD to monitor the  $V_{DD}$  power supply by comparing it to a threshold selected by the PLS[2:0] bits in the [PWR power control register \(PWR\\_CR1\)](#).

The PVD is enabled by setting the PVDE bit.

A PVDO flag is available, in the [PWR power control/status register \(PWR\\_CSR1\)](#), to indicate if  $V_{DD}$  is higher or lower than the PVD threshold. This event is internally connected

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

**Figure 11. PVD thresholds**



## 4.3 Low-power modes

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

The devices feature three low-power modes:

- Sleep mode (Cortex<sup>®</sup>-M7 core stopped, peripherals kept running)
- Stop mode (all clocks are stopped)
- Standby mode (1.2 V domain powered off)

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

- Slowing down the system clocks
- Gating the clocks to the APBx and AHBx peripherals when they are unused.

### Entering low-power mode

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

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

### Exiting low-power mode

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

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

- NVIC IRQ interrupt:

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

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

- Event

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

The MCU exits from Standby low-power mode through an external reset (NRST pin), an IWDG reset, a rising or falling on one of the enabled WKUPx pins or a RTC event occurs (see [Figure 298: RTC block diagram](#)).

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

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

**Table 13. 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	Any interrupt	CPU CLK OFF no effect on other clocks or analog clock sources	None	ON
	WFE	Wakeup event			

**Table 13. Low-power mode summary (continued)**

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

**Table 14. Features over all modes<sup>(1)</sup>**

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

**Table 14. Features over all modes (continued)<sup>(1)</sup>**

Peripheral	Run	Sleep	Stop		Standby		VBAT
			-	Wakeup	-	Wakeup	
Number of RTC tamper pins	3	3	3	3	3	3	2
CRC calculation unit	O	O	-	-	-	-	-
GPIOs	Y	Y	Y	Y	-	6 pins	2 tamper
DMA	O	O	-	-	-	-	-
Chrom-Art Accelerator (DMA2D)	O	O	-	-	-	-	-
LCD-TFT	O	O	-	-	-	-	-
DCMI	O	O	-	-	-	-	-
USARTx (x=1..8)	O	O	-	-	-	-	-
I2Cx (x=1,2,3,4)	O	O	-	-	-	-	-
SPIx (x=1..6)	O	O	-	-	-	-	-
SAIx (x=1,2)	O	O	-	-	-	-	-
SPDIFRX	O	O	-	-	-	-	-
ADCx (x=1,2,3)	O	O	-	-	-	-	-
DACx (x=1,2)	O	O	-	-	-	-	-
Temperature sensor	O	O	-	-	-	-	-
Timers (TIMx)	O	O	-	-	-	-	-
Low-power timer 1 (LPTIM1)	O	O	O	O	-	-	-
Independent watchdog (IWDG)	O	O	O	O	O	O	-
Window watchdog (WWDG)	O	O	-	-	-	-	-
Systick timer	O	O	-	-	-	-	-
Random number generator (RNG)	O	O	-	-	-	-	-
Cryptographic processor (CRYP)	O	O	-	-	-	-	-
Hash processor (HASH)	O	O	-	-	-	-	-
SDMMC	O	O	-	-	-	-	-
CANx (x=1,2)	O	O	-	-	-	-	-
USB OTG FS	O	O	-	O	-	-	-
USB OTG HS	O	O	-	O	-	-	-
Ethernet	O	O	-	O	-	-	-
HDMI-CEC	O	O	-	-	-	-	-

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

### 4.3.1 Debug mode

By default, the debug connection is lost when the devices enters in Stop or Standby mode while the debug features are used. This is due to the fact that the Cortex<sup>®</sup>-M7 core is no longer clocked.

However, by setting some configuration bits in the DBGMCU\_CR register, the software can be debugged even when using the low-power modes extensively. For more details, refer to [Section 40.16.1: Debug support for low-power modes](#).

### 4.3.2 Run mode

#### Slowing down system clocks

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

For more details refer to [Section 5.3.3: RCC clock configuration register \(RCC\\_CFGR\)](#).

#### Peripheral clock gating

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

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

Peripheral clock gating is controlled by the AHB1 peripheral clock enable register (RCC\_AHB1ENR), AHB2 peripheral clock enable register (RCC\_AHB2ENR), AHB3 peripheral clock enable register (RCC\_AHB3ENR) (see [Section 5.3.10: RCC AHB1 peripheral clock register \(RCC\\_AHB1ENR\)](#), [Section 5.3.11: RCC AHB2 peripheral clock enable register \(RCC\\_AHB2ENR\)](#), and [Section 5.3.12: RCC AHB3 peripheral clock enable register \(RCC\\_AHB3ENR\)](#)).

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

### 4.3.3 Low-power mode

#### Entering low-power mode

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

### Exiting low-power mode

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

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

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

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

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

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

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

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

- Event

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

From Standby mode the MCU exits Low-power mode through an external reset (NRST pin), an IWDG reset, a rising edge on one of the enabled WKUPx pins or a RTC event (see [Figure 298: RTC block diagram](#)).

### 4.3.4 Sleep mode

#### I/O states in Sleep mode

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

#### Entering Sleep mode

The Sleep mode is entered by executing the WFI (Wait For Interrupt) or WFE (Wait for Event) instructions. Two options are available to select the Sleep mode entry mechanism, depending on the SLEEPONEXIT bit in the Cortex®-M7 System Control register:

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

Refer to [Table 15](#) and [Table 16](#) for details on how to enter Sleep mode.

### Exiting Sleep mode

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

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

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

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

Refer to [Table 15](#) and [Table 16](#) for more details on how to exit Sleep mode.

**Table 15. Sleep-now entry and exit**

Sleep-now 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>– SLEEPDEEP = 0, and</li> <li>– No interrupt (for WFI) or event (for WFE) is pending.</li> </ul> <p>Refer to the Cortex®-M7 System Control register.</p>
<b>Mode exit</b>	<p>On Return from ISR while:</p> <ul style="list-style-type: none"> <li>– SLEEPDEEP = 0 and</li> <li>– SLEEPONEXIT = 1,</li> <li>– No interrupt is pending.</li> </ul> <p>Refer to the Cortex®-M7 System Control register.</p>
<b>Wakeup latency</b>	None

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

Sleep-on-exit	Description
<b>Mode entry</b>	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0, and – No interrupt (for WFI) or event (for WFE) is pending. Refer to the Cortex®-M7 System Control register.
	On Return from ISR while: – SLEEPDEEP = 0, and – SLEEPONEXIT = 1, and – No interrupt is pending. Refer to the Cortex®-M7 System Control register.
<b>Mode exit</b>	Interrupt: refer to <a href="#">Table 44: STM32F75xxx and STM32F74xxx vector table</a>
<b>Wakeup latency</b>	None

#### 4.3.5 Stop mode

The Stop mode is based on the Cortex®-M7 deepsleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the 1.2 V domain are stopped, the PLLs, the HSI and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved.

In Stop mode, the power consumption can be further reduced by using additional settings in the PWR\_CR1 register. However this will induce an additional startup delay when waking up from Stop mode (see [Table 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

**Table 17. Stop operating modes (continued)**

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

### I/O states in Stop mode

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

### Entering Stop mode

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

Refer to [Table 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 \(PWR\\_CR1\)](#).

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

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

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

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

- Independent watchdog (IWDG): the IWDG is started by writing to its Key register or by hardware option. Once started it cannot be stopped except by a Reset. See [Section 27.3 in Section 27: Independent watchdog \(IWDG\)](#).
- Real-time clock (RTC): this is configured by the RTCEN bit in the [RCC backup domain control register \(RCC\\_BDCR\)](#)
- Internal RC oscillator (LSI RC): this is configured by the LSION bit in the [RCC clock control & status register \(RCC\\_CSR\)](#).
- External 32.768 kHz oscillator (LSE OSC): this is configured by the LSEON bit in the [RCC backup domain control register \(RCC\\_BDCR\)](#).

The ADC or DAC can also consume power during the Stop mode, unless they are disabled before entering it. To disable them, the ADON bit in the ADC\_CR2 register and the ENx bit in the DAC\_CR register must both be written to 0.

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

### Exiting Stop mode

Refer to [Table 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 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 (STM32F75xxx and STM32F74xxx)**

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>– No interrupt (for WFI) or event (for WFE) is pending.</li> <li>– SLEEPDEEP bit is set in Cortex®-M7 System Control register,</li> <li>– PDSS bit is cleared in Power Control register (PWR_CR1),</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>– No interrupt is pending,</li> <li>– SLEEPDEEP bit is set in Cortex®-M7 System Control register, and</li> <li>– SLEEPONEXIT = 1, and</li> <li>– PDSS is cleared in Power Control register (PWR_CR1).</li> </ul>
	<p><i>Note: To enter Stop mode, all EXTI Line pending bits (in Pending register (EXTI_PR)), all peripheral interrupts pending bits, the RTC Alarm (Alarm A and Alarm B), RTC wakeup, RTC tamper, and RTC time stamp flags, must be reset. Otherwise, the Stop mode entry procedure is ignored and program execution continues.</i></p>

**Table 18. Stop mode entry and exit (STM32F75xxx and STM32F74xxx) (continued)**

Stop mode	Description
<b>Mode exit</b>	If WFI or Return from ISR was used for entry: All EXTI lines configured in Interrupt mode (the corresponding EXTI Interrupt vector must be enabled in the NVIC). The interrupt source can be external interrupts or peripherals with wakeup capability. Refer to <a href="#">Table 44: STM32F75xxx and STM32F74xxx vector table on page 290</a> . If WFE was used for entry and SEVONPEND = 0: All EXTI Lines configured in event mode. Refer to <a href="#">Section 11.3: Wakeup event management on page 296</a> If WFE was used for entry and SEVONPEND = 1: – Any EXTI lines configured in Interrupt mode (even if the corresponding EXTI Interrupt vector is disabled in the NVIC). The interrupt source can be external interrupts or peripherals with wakeup capability. Refer to <a href="#">Table 44: STM32F75xxx and STM32F74xxx vector table on page 290</a> . – Wakeup event: refer to <a href="#">Section 11.3: Wakeup event management on page 296</a> .
<b>Wakeup latency</b>	Refer to <a href="#">Table 17: Stop operating modes</a>

### 4.3.6 Standby mode

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

#### Entering Standby mode

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

Refer to [Table 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 27.3 in Section 27: Independent watchdog \(IWDG\)](#).
- Real-time clock (RTC): this is configured by the RTCEN bit in the backup domain control register (RCC\_BDCR)
- Internal RC oscillator (LSI RC): this is configured by the LSION bit in the Control/status register (RCC\_CSR).
- External 32.768 kHz oscillator (LSE OSC): this is configured by the LSEON bit in the backup domain control register (RCC\_BDCR)

#### Exiting Standby mode

The microcontroller exits Standby mode when an external Reset (NRST pin), an IWDG Reset, a rising or falling edge on WKUP pin, an RTC alarm, a tamper event, or a time stamp

event is detected. All registers are reset after wakeup from Standby except for [PWR power control/status register \(PWR\\_CSR1\)](#).

After waking up from Standby mode, program execution restarts in the same way as after a Reset (boot pin sampling, vector reset is fetched, etc.). The SBF status flag in the [PWR power control/status register \(PWR\\_CSR1\)](#) indicates that the MCU was in Standby mode.

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

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

Standby 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>– SLEEPDEEP is set in Cortex®-M7 System Control register,</li> <li>– PDDS bit is set in Power Control register (PWR_CR1),</li> <li>– No interrupt (for WFI) or event (for WFE) is pending,</li> <li>– WUIF bit is cleared in Power Control and Status register (PWR_CSR1),</li> <li>– the RTC flag corresponding to the chosen wakeup source (RTC Alarm A, RTC Alarm B, RTC wakeup, Tamper or Timestamp flags) is cleared.</li> <li>– EIWUP bit is set in Power Control register (PWR_CSR1).</li> </ul>
	<p>On return from ISR while:</p> <ul style="list-style-type: none"> <li>– SLEEPDEEP bit is set in Cortex®-M7 System Control register, and</li> <li>– SLEEPONEXIT = 1, and</li> <li>– PDDS bit is set in Power Control register (PWR_CR1), and</li> <li>– No interrupt is pending,</li> <li>– WUIF bit is cleared in Power Control and Status register (PWR_CSR1),</li> <li>– The RTC flag corresponding to the chosen wakeup source (RTC Alarm A, RTC Alarm B, RTC wakeup, Tamper or Timestamp flags) is cleared.</li> </ul>
<b>Mode exit</b>	WKUP pin rising or falling edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.
<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)
- PC13 if configured for tamper, time stamp, RTC Alarm out, or RTC clock calibration out
- WKUP pins (PA0/PA2/PC1/PC13/PI8/PI11), if enabled

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

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

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

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

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

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

For this purpose, two of the three alternate RTC clock sources can be selected by programming the RTCSEL[1:0] bits in the *RCC backup domain control register (RCC\_BDCR)*:

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

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

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

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

- To wake up the device from the Standby mode with an RTC alarm event, it is necessary to:
  - a) Enable the RTC alarm interrupt in the RTC\_CR register
  - b) Configure the RTC to generate the RTC alarm
  - c) Enable the RTC internal wakeup event from standby by setting the EIWUP bit in the PWR\_CSR1 register
- To wake up the device from the Standby mode with an RTC tamper or time stamp event, it is necessary to:
  - a) Enable the RTC time stamp interrupt in the RTC\_CR register or the RTC tamper interrupt in the RTC\_TAFCR register
  - b) Configure the RTC to detect the tamper or time stamp event
  - c) Enable the RTC internal wakeup event from standby by setting the EIWUP bit in the PWR\_CSR1 register
- To wake up the device from the Standby mode with an RTC wakeup event, it is necessary to:
  - a) Enable the RTC wakeup interrupt in the RTC\_CR register
  - b) Configure the RTC to generate the RTC wakeup event
  - c) Enable the RTC internal wakeup event from standby by setting the EIWUP bit in the PWR\_CSR1 register

### Safe RTC alternate function wakeup flag clearing sequence

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

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

## 4.4 Power control registers

### 4.4.1 PWR power control register (PWR\_CR1)

Address offset: 0x00

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

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

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

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

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

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

00: Under-drive disable

01: Reserved

10: Reserved

11:Under-drive enable

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

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

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

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

0: Over-drive switching disabled

1: Over-drive switching enabled

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

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

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

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

0: Over-drive disabled

1: Over-drive enabled

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

These bits control the main internal voltage regulator output voltage to achieve a trade-off between performance and power consumption when the device does not operate at the maximum frequency (refer to the STM32F75xxx and STM32F74xxx datasheets for more details).

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

00: Reserved (Scale 3 mode selected)

01: Scale 3 mode

10: Scale 2 mode

11: Scale 1 mode (reset value)

**Bit 13 **ADCDC1**:**

0: No effect.

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

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

**Bit 12 Reserved, must be kept at reset value.**

**Bit 11 **MRUDS**:** Main regulator in deepsleep under-drive mode

This bit is set and cleared by software.

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

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

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

This bit is set and cleared by software.

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

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

**Bit 9 **FPDS**:** Flash power-down in Stop mode

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

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

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

**Bit 8 **DBP**:** Disable backup domain write protection

In reset state, the RCC\_BDCR register, the RTC registers (including the backup registers), and the BRE bit of the PWR\_CSR1 register, are protected against parasitic write access. This bit must be set to enable write access to these registers.

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

1: Access to RTC and RTC Backup registers and backup SRAM enabled

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

These bits are written by software to select the voltage threshold detected by the programmable 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**: Programmable voltage detector enable

This bit is set and cleared by software.

- 0: PVD disabled
- 1: PVD enabled

Bit 3 **CSBF**: Clear standby flag

This bit is always read as 0.

- 0: No effect
- 1: Clear the SDF Standby Flag (write)

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

Bit 1 **PDSS**: Power-down deepsleep

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

- 0: Enter Stop mode when the CPU enters deepsleep. The regulator status depends on the LPDS bit.
- 1: Enter Standby mode when the CPU enters deepsleep.

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

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

- 0: Main voltage regulator ON during Stop mode
- 1: Low-power voltage regulator ON during Stop mode

**4.4.2 PWR power control/status register (PWR\_CSR1)**

Address offset: 0x04

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UDRDY[1:0]	ODSWRDY	ODRDY	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	VOS RDY	Res.	Res.	Res.	Res.	BRE	EIWUP	Res.	Res.	Res.	Res.	BRR	PVDO	SBF	WUIF
	r					rw	rw					r	r	r	r

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

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

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

00: Under-drive is disabled

01: Reserved

10: Reserved

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

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

0: Over-drive mode is not active.

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

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

0: Over-drive mode not ready.

1: Over-drive mode ready

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

0: Not ready

1: Ready

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

**Bit 9 BRE: Backup regulator enable**

When set, the Backup regulator (used to maintain backup SRAM content in Standby and V<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 or by a system reset or by a power reset. It is reset by a backup domain reset.*

*The DBP bit of PWR\_CR register must be set to 1 before PWR\_CSR.BRE can be written.*

**Bit 8 EIWUP: Enable internal wakeup**

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

This bit is always read as 0.

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

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

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

**Bit 3 BRR: Backup regulator ready**

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

0: Backup Regulator not ready

1: Backup Regulator ready

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

Bit 2 **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<sub>DD</sub> is higher than the PVD threshold selected with the PLS[2:0] bits.

1: V<sub>DD</sub> is lower than the PVD threshold selected with the PLS[2:0] bits.

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

Bit 1 **SBF**: Standby flag

This bit is set by hardware and cleared only by a POR/PDR (power-on reset/power-down reset) or by setting the CSBF bit in the [PWR power control register \(PWR\\_CR1\)](#)

0: Device has not been in Standby mode

1: Device has been in Standby mode

Bit 0 **WUIF**: Wakeup internal flag

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

0: No wakeup internal event occurred

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

**4.4.3 PWR power control/status register 2 (PWR\_CR2)**

Address offset: 0x08

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

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

Bits 31:14 Reserved, always read as 0.

Bit 13 **WUPP6**: Wakeup pin polarity bit for PI11

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

0: Detection on rising edge

1: Detection on falling edge

Bit 12 **WUPP5**: Wakeup pin polarity bit for PI8

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

0: Detection on rising edge

1: Detection on falling edge

Bit 11 **WUPP4**: Wakeup pin polarity bit for PC13

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

0: Detection on rising edge

1: Detection on falling edge

Bit 10 **WUPP3**: Wakeup pin polarity bit for PC1

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

0: Detection on rising edge

1: Detection on falling edge

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

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

0: Detection on rising edge

1: Detection on falling edge

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

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

0: Detection on rising edge

1: Detection on falling edge

Bits 7:6 Reserved, always read as 0

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

These bits are always read as 0

0: No effect

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

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

These bits are always read as 0

0: No effect

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

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

These bits are always read as 0

0: No effect

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

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

These bits are always read as 0

0: No effect

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

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

These bits are always read as 0

0: No effect

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

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

These bits are always read as 0

0: No effect

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

#### 4.4.4 PWR power control register 2 (PWR\_CSR2)

Address offset: 0x0C

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

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

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

Bits 31:14 Reserved, always read as 0.

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

This bit is set and cleared by software.

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

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

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

This bit is set and cleared by software.

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

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

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

This bit is set and cleared by software.

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

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

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

This bit is set and cleared by software.

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

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

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

This bit is set and cleared by software.

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

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

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

This bit is set and cleared by software.

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

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

Bits 7:6 Reserved, always read as 0

**Bit 5 WUPF6:** Wakeup Pin flag for PI11

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF6 bit in the PWR power control register 2 (PWR\_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PI11

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

**Bit 4 WUPF5:** Wakeup Pin flag for PI8

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF5 bit in the PWR power control register 2 (PWR\_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PI8

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

**Bit 3 WUPF4:** Wakeup Pin flag for PC13

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF4 bit in the PWR power control register 2 (PWR\_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PC13

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

**Bit 2 WUPF3:** Wakeup Pin flag for PC1

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF3 bit in the PWR power control register 2 (PWR\_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PC1

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

**Bit 1 WUPF2:** Wakeup Pin flag for PA2

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF2 bit in the PWR power control register 2 (PWR\_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PA2

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

**Bit 0 WUPF1:** Wakeup Pin flag for PA0

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF1 bit in the PWR power control register 2 (PWR\_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PA0

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

## 4.5 PWR register map

The following table summarizes the PWR registers.

**Table 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	4	3	2	1	0														
0x000	PWR_CR1	Res.	UDEN[1:0]	1	UDEN[1:0]	1	UDEN[1:0]	1	ADCDC1	0	ADCDC1	13	12	11	10	9	8	7	6	5	4	3	2	1	0																						
	Reset value	Res.	UDRDY[1:0]	0	UDRDY[1:0]	0	UDRDY[1:0]	0	ODSWRDY	1	ODSWRDY	1	ODSWEN	17	ODEN	16	VOS[1:9]	1	VOS[1:9]	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
0x004	PWR_CSR1	Res.	UDRDY[1:0]	0	UDRDY[1:0]	0	UDRDY[1:0]	0	ODSWRDY	1	ODSWRDY	1	ODSWEN	17	ODEN	16	VOS[1:9]	1	VOS[1:9]	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
	Reset value	Res.	UDRDY[1:0]	0	UDRDY[1:0]	0	UDRDY[1:0]	0	ODSWRDY	1	ODSWRDY	1	ODSWEN	17	ODEN	16	VOS[1:9]	1	VOS[1:9]	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
0x008	PWR_CR2	Res.	UDRDY[1:0]	0	UDRDY[1:0]	0	UDRDY[1:0]	0	ODSWRDY	1	ODSWRDY	1	ODSWEN	17	ODEN	16	VOS[1:9]	1	VOS[1:9]	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
	Reset value	Res.	UDRDY[1:0]	0	UDRDY[1:0]	0	UDRDY[1:0]	0	ODSWRDY	1	ODSWRDY	1	ODSWEN	17	ODEN	16	VOS[1:9]	1	VOS[1:9]	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
0x00C	PWR_CSR2	Res.	UDRDY[1:0]	0	UDRDY[1:0]	0	UDRDY[1:0]	0	ODSWRDY	1	ODSWRDY	1	ODSWEN	17	ODEN	16	VOS[1:9]	1	VOS[1:9]	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
	Reset value	Res.	UDRDY[1:0]	0	UDRDY[1:0]	0	UDRDY[1:0]	0	ODSWRDY	1	ODSWRDY	1	ODSWEN	17	ODEN	16	VOS[1:9]	1	VOS[1:9]	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 5 Reset and clock control (RCC)

### 5.1 Reset

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

#### 5.1.1 System reset

A system reset sets all registers to their reset values unless specified otherwise in the register description (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>-M7 Application Interrupt and Reset Control Register must be set to force a software reset on the device. Refer to the Cortex<sup>®</sup>-M7 technical reference manual for more details.

#### Low-power management reset

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

1. Reset generated when entering the Standby mode:

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

2. Reset when entering the Stop mode:

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

#### 5.1.2 Power reset

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

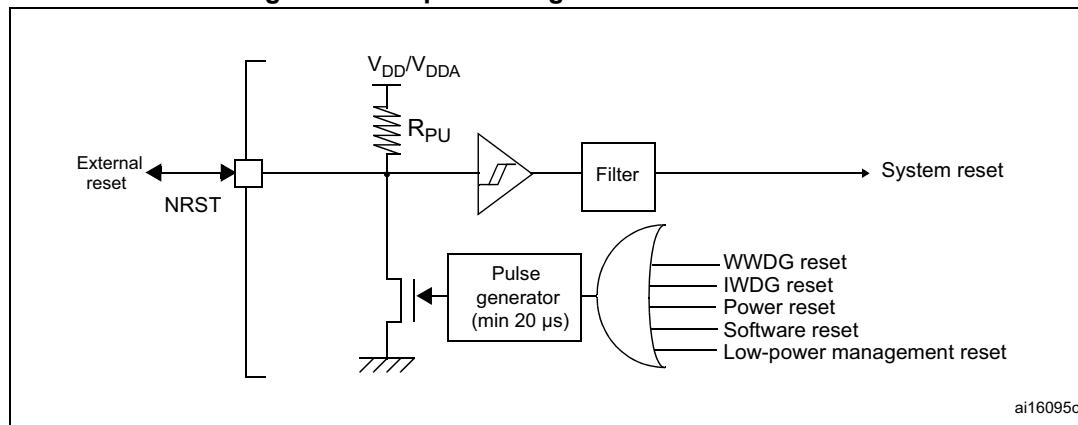
1. Power-on/power-down reset (POR/PDR reset) or brownout (BOR) reset
2. When exiting the Standby mode

A power reset sets all registers to their reset values except the Backup domain (see [Figure 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 µs for each internal reset source. In case of an external reset, the reset pulse is generated while the NRST pin is asserted low.

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



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

### 5.1.3 Backup domain reset

The backup domain reset sets all RTC registers, the RCC\_BDCR register and the bit BRE of PWR\_CSR 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.

*Note:* The bit DBP of the register PWR\_CR must be set to 1 in order to generate the backup domain reset.

## 5.2 Clocks

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

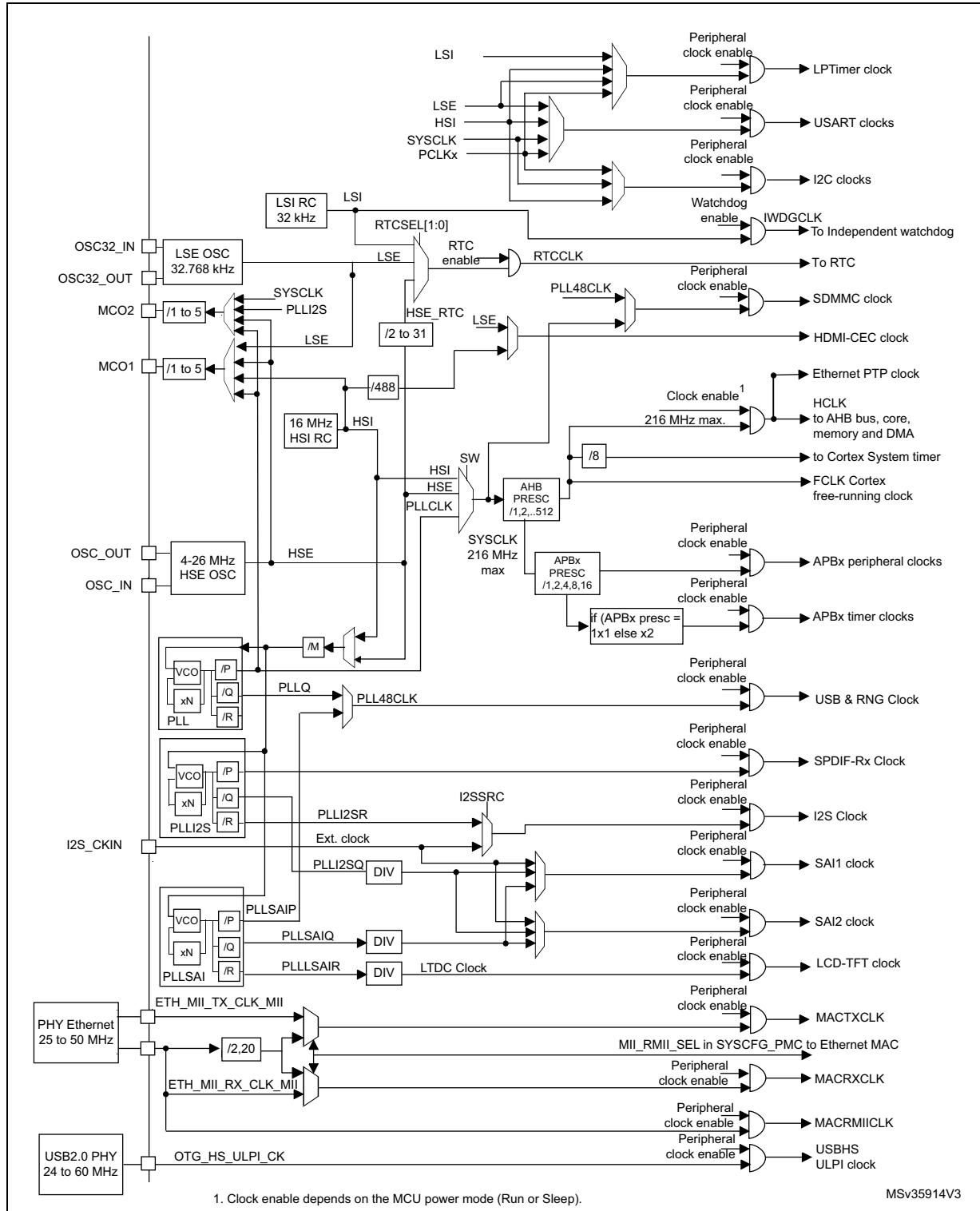
- HSI oscillator clock
- HSE oscillator clock
- Main PLL (PLL) clock

The devices have the two following secondary clock sources:

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

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

**Figure 13. Clock tree**



- For full details about the internal and external clock source characteristics, refer to the Electrical characteristics section in

- the device datasheet.
2. When TIMPRE bit of the RCC\_DCKCFGR1 register is reset, if APBx prescaler is 1, then TIMxCLK = PCLKx, otherwise TIMxCLK = 2x PCLKx.
  3. When TIMPRE bit in the RCC\_DCKCFGR1 register is set, if APBx prescaler is 1,2 or 4, then TIMxCLK = HCLK, otherwise TIMxCLK = 4x PCLKx.

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

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

- The 48MHz clock, used for USB OTG FS, SDMMC and RNG. This clock is derived from one of the following sources:
  - main PLL VCO (PLLQ Clock)
  - PLLSAI VCO (PLLSAI clock)
- The U(S)ARTs clocks which are derived from one of the following sources:
  - System clock (SYSCLK)
  - HSI clock
  - LSE clock
  - APB1 or APB2 clock (PCLK1 or PCLK2 depending on which APB is mapped the U(S)ART)
- The I2Cs clocks which are derived from one of the following sources:
  - System clock (SYSCLK)
  - HSI clock
  - APB1 clock (PCLK1)
- I2S clock

To achieve high-quality audio performance, the I2S clock can be derived either from a specific PLL (PLLI2S) or from an external clock mapped on the I2S\_CKIN pin. For more information about I2S clock frequency and precision, refer to [Section 32.7.5: Clock generator](#).
- The SAI1 and SAI2 clocks which are derived from one of the following sources:
  - PLLSAI VCO (PLLSAIQ)
  - PLLI2S VCO (PLLI2SQ)
  - External clock mapped on the I2S\_CKIN pin.
- LTDC clock

The LTDC clock is generated from a specific PLL (PLLSAI).
- The low-power timer (LPTIM1) clock which is derived from one of the following sources:
  - LSI clock
  - LSE clock
  - HSI clock
  - APB1 clock (PCLK1)
  - External clock mapped on LPTIM1\_IN1
- The USB OTG HS (60 MHz) clock which is provided from the external PHY
- The Ethernet MAC clocks (TX, RX and RMII) which are provided from the external PHY. For further information on the Ethernet configuration, refer to [Section 38.4.4: MII/RMII selection](#) in the Ethernet peripheral description. When the Ethernet is used,

the AHB clock frequency must be at least 25 MHz.

- SPDIFRX clock: which is generated from PLLI2SP VCO.
- The HDMI-CEC clock which is derived from one of the following sources:
  - LSE clock
  - HSI clock divided by 488

- The RTC clock which is derived from one of the following sources:
  - LSE clock
  - LSI clock
  - HSE clock divided by 32
- The IWDG clock which is always the LSI clock.
- The timer clock frequencies are automatically set by hardware. There are two cases depending on the value of TIMPRE bit in RCC\_CFGR register:
  - If TIMPRE bit in RCC\_DCKCFGR1 register is reset:  
If the APB prescaler is configured to a division factor of 1, the timer clock frequencies (TIMxCLK) are set to PCLKx. Otherwise, the timer clock frequencies are twice the frequency of the APB domain to which the timers are connected:  $TIMxCLK = 2 \times PCLKx$ .
  - If TIMPRE bit in RCC\_DCKCFGR1 register is set:  
If the APB prescaler is configured to a division factor of 1, 2 or 4, the timer clock frequencies (TIMxCLK) are set to HCLK. Otherwise, the timer clock frequencies are four times the frequency of the APB domain to which the timers are connected:  $TIMxCLK = 4 \times PCLKx$ .

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

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

## 5.2.1 HSE clock

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

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

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

Figure 14. HSE/ LSE clock sources

-	Hardware configuration
External clock	<p>External clock source</p> <p>MSv42094V1</p>
Crystal/ceramic resonators	<p>Load capacitors</p> <p>MSv42081V1</p>

### External source (HSE bypass)

In this mode, an external clock source must be provided. This mode is selected 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 can be used as GPIO. 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 \(RCC\\_CR\)](#).

## 5.2.2 HSI clock

The HSI clock signal is generated from an internal 16 MHz RC oscillator and can be used directly as a system clock, or used as PLL input.

The HSI RC oscillator has the advantage of providing a clock source at low cost (no external components). It also has a faster startup time than the HSE crystal oscillator however, even

with calibration the frequency is less accurate than an external crystal oscillator or ceramic resonator.

### Calibration

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

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. The HSI frequency can be trimmed in the application using the HSITRIM[4:0] bits in the [RCC clock control register \(RCC\\_CR\)](#).

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

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

The HSI signal can also be used as a backup source (Auxiliary clock) if the HSE crystal oscillator fails. Refer to [Section 5.2.7: Clock security system \(CSS\) on page 141](#).

## 5.2.3 PLL

The devices feature three PLLs:

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

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

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

The three PLLs are disabled by hardware when entering Stop and Standby modes, or when an HSE failure occurs when HSE or PLL (clocked by HSE) are used as system clock. [RCC PLL configuration register \(RCC\\_PLLCFGR\)](#), [RCC clock configuration register \(RCC\\_CFGR\)](#), and [RCC dedicated clocks configuration register \(RCC\\_DCKCFGR1\)](#) can be used to configure PLL, PLLI2S, and PLLSAI.

### 5.2.4 LSE clock

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

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

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

#### External source (LSE bypass)

In this mode, an external clock source must be provided. It must have a frequency up to 1 MHz. This mode is selected 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 can be used as GPIO. See [Figure 14](#).

### 5.2.5 LSI clock

The LSI RC acts as an low-power clock source that can be kept running in Stop and Standby mode for the independent watchdog (IWDG) and Auto-wakeup unit (AWU). The clock frequency is around 32 kHz. For more details, refer to the electrical characteristics section of the datasheets.

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

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

### 5.2.6 System clock (SYSCLK) selection

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

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

### 5.2.7 Clock security system (CSS)

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

If a failure is detected on the HSE clock, this oscillator is automatically disabled, a clock failure event is sent to the break inputs of advanced-control timers TIM1 and TIM8, and an interrupt is generated to inform the software about the failure (clock security system).

interrupt CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex®-M7 NMI (non-maskable interrupt) exception vector.

**Note:** When the CSS is enabled, if the HSE clock happens to fail, the CSS generates an interrupt, which causes the automatic generation of an NMI. The NMI is executed indefinitely unless the CSS interrupt pending bit is cleared. As a consequence, the application has to clear the CSS interrupt in the NMI ISR by setting the CSSC bit in the Clock interrupt register (RCC\_CIR).

If the HSE oscillator is used directly or indirectly as the system clock (indirectly meaning that it is directly used as PLL input clock, and that PLL clock is the system clock) and a failure is detected, then the system clock switches to the HSI oscillator and the HSE oscillator is disabled.

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

### 5.2.8 RTC/AWU clock

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

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

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

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

- If LSE is selected as the RTC clock:
  - The RTC continues to work even if the V<sub>DD</sub> supply is switched off, provided the V<sub>BAT</sub> supply is maintained.
  - The RTC remains clocked and functional under system reset.
- If LSI is selected as the Auto-wakeup unit (AWU) clock:
  - The AWU state is not guaranteed if the V<sub>DD</sub> supply is powered off. Refer to [Section 5.2.5: LSI clock on page 141](#) 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<sub>DD</sub> supply is powered off or if the internal voltage regulator is powered off (removing power from the 1.2 V domain) and also when entering in Stop mode

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

### 5.2.9 Watchdog clock

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

### 5.2.10 Clock-out capability

Two microcontroller clock output (MCO) pins are available:

- MCO1

The user 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

The user 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.

### 5.2.11 Internal/external clock measurement using TIM5/TIM11

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

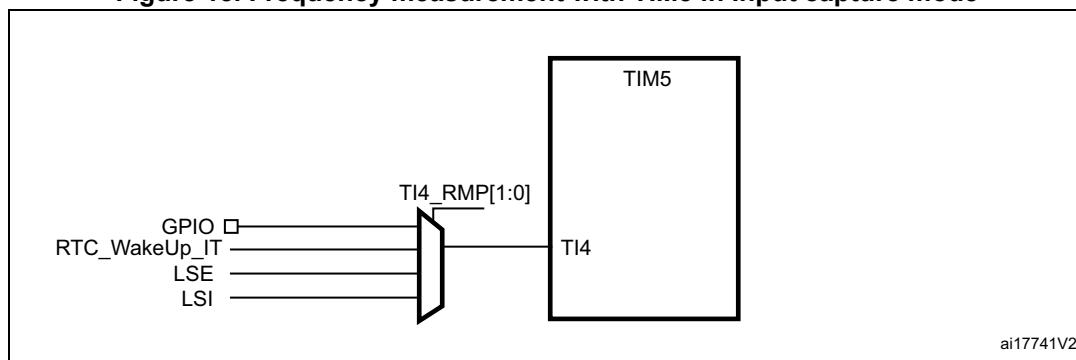
The basic concept consists in providing a relative measurement (e.g. HSI/LSE ratio): the precision is therefore tightly linked to the ratio between the two clock sources. The greater the ratio, the better the measurement.

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

Use the following procedure to measure the LSI frequency:

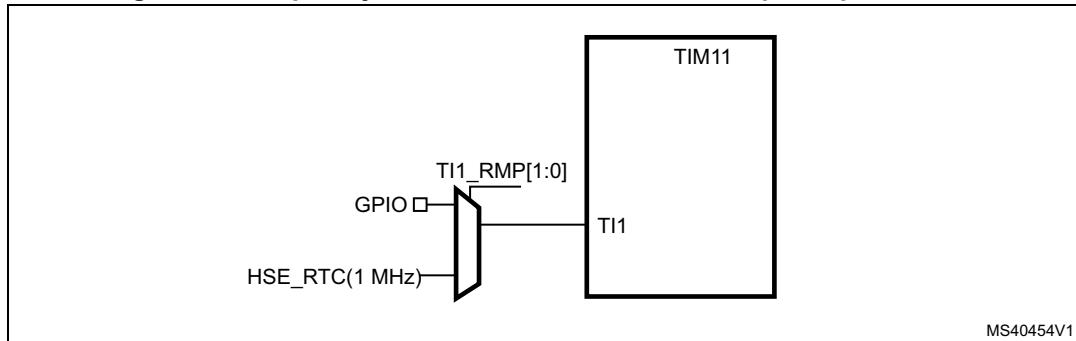
1. Enable the TIM5 timer and configure channel4 in Input capture mode.
2. This bit is set the TI4\_RMP bits in the TIM5\_OR register to 0x01 to connect the LSI clock internally to TIM5 channel4 input capture for calibration purposes.
3. Measure the LSI clock frequency using the TIM5 capture/compare 4 event or interrupt.
4. Use the measured LSI frequency to update the prescaler of the RTC depending on the desired time base and/or to compute the IWDG timeout.

**Figure 15. Frequency measurement with TIM5 in Input capture mode**



### Internal/external clock measurement using TIM11 channel1

TIM11 has an input multiplexer which allows choosing whether the input capture is triggered by the I/O or by an internal clock. This selection is performed through TI1\_RMP [1:0] bits in the TIM11\_OR register. The HSE\_RTC clock (HSE divided by a programmable prescaler) is connected to channel 1 input capture to have a rough indication of the external crystal frequency. This requires that the HSI is the system clock source. This can be useful for instance to ensure compliance with the IEC 60730/IEC 61335 standards which require to be able to determine harmonic or subharmonic frequencies (-50/+100% deviations).

**Figure 16. Frequency measurement with TIM11 in Input capture mode**

### 5.2.12 Peripheral clock enable register (RCC\_AHBxENR, RCC\_APBxENRy)

Each peripheral clock can be enabled by the xxxxEN bit of the RCC\_AHBxENR or RCC\_APBxENRy registers.

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

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

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

## 5.3 RCC registers

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

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

Address offset: 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	PLL12S RDY	PLL12S ON	PLL1RD Y	PLLON	Res.	Res.	Res.	Res.	CSS ON	HSE BYP	HSE RDY	HSE ON
		r	rw	r	rw	r	rw					rw	rw	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]						Res.	HSI RDY
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

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

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

Set by hardware to indicate that the PLLSAI is locked.

0: PLLSAI unlocked

1: PLLSAI locked

Bit 28 **PLLSAION**: PLLSAI enable

Set and cleared by software to enable PLLSAI.

Cleared by hardware when entering Stop or Standby mode.

0: PLLSAI OFF

1: PLLSAI ON

Bit 27 **PLL12SRDY**: PLLI2S clock ready flag

Set by hardware to indicate that the PLLI2S is locked.

0: PLLI2S unlocked

1: PLLI2S locked

Bit 26 **PLL12SON**: PLLI2S enable

Set and cleared by software to enable PLLI2S.

Cleared by hardware when entering Stop or Standby mode.

0: PLLI2S OFF

1: PLLI2S ON

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

Set by hardware to indicate that PLL is locked.

0: PLL unlocked

1: PLL locked

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

Set and cleared by software to enable PLL.

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

0: PLL OFF

1: PLL ON

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

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

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

0: Clock security system OFF (Clock detector OFF)

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

Bit 18 **HSEBYP**: HSE clock bypass

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

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

0: HSE oscillator not bypassed

1: HSE oscillator bypassed with an external clock

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

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

0: HSE oscillator not ready

1: HSE oscillator ready

Bit 16 **HSEON**: HSE clock enable

Set and cleared by software.

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

0: HSE oscillator OFF

1: HSE oscillator ON

Bits 15:8 **HSICAL[7:0]**: Internal high-speed clock calibration

These bits are initialized automatically at startup.

Bits 7:3 **HSITRIM[4:0]**: Internal high-speed clock trimming

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

Bit 2 Reserved, must be kept at reset value.

Bit 1 **HSIRDY**: Internal high-speed clock ready flag

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

0: HSI oscillator not ready

1: HSI oscillator ready

Bit 0 **HSION**: Internal high-speed clock enable

Set and cleared by software.

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

0: HSI oscillator OFF

1: HSI oscillator ON

### 5.3.2 RCC PLL configuration register (RCC\_PLLCFGR)

Address offset: 0x04

Reset value: 0x2400 3010

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

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

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

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

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

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

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

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

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

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 **PLLSRC:** 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<sub>P</sub>[1:0]**: Main PLL (PLL) division factor for main system clock

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

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

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

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

Bits 14:6 **PLL<sub>N</sub>[8:0]**: Main PLL (PLL) multiplication factor for VCO

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

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

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

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

...

- 000110010: PLLN = 50

...

- 001100011: PLLN = 99

- 001100100: PLLN = 100

...

- 110110000: PLLN = 432

- 110110001: PLLN = 433, wrong configuration

...

- 111111111: PLLN = 511, wrong configuration

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

Bits 5:0 **PLL<sub>M</sub>[5:0]**: Division factor for the main PLLs (PLL, PLLI2S and PLLSAI) input clock

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

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

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

- 00000: PLLM = 0, wrong configuration

- 000001: PLLM = 1, wrong configuration

- 000010: PLLM = 2

- 000011: PLLM = 3

- 000100: PLLM = 4

...

- 111110: PLLM = 62

- 111111: PLLM = 63

### 5.3.3 RCC clock configuration register (RCC\_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

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

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

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

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

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

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

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

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

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

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

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

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

Bit 23 **I2SSRC:** I2S clock selection

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

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

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

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

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

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

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

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

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

**Bits 15:13 PPREG2:** APB high-speed prescaler (APB2)

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

**Caution:** The software has to set these bits correctly not to exceed 108 MHz on this domain. The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after PPREG2 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 PPREG1:** APB Low-speed prescaler (APB1)

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

**Caution:** The software has to set these bits correctly not to exceed 54 MHz on this domain. The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after PPREG1 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:** System clock switch status

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

00: HSI oscillator used as the system clock

01: HSE oscillator used as the system clock

10: PLL used as the system clock

11: not applicable

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

Set and cleared by software to select the system clock source.

Set by hardware to force the HSI selection when leaving the Stop or Standby mode or in case of failure of the HSE oscillator used directly or indirectly as the system clock.

00: HSI oscillator selected as system clock

01: HSE oscillator selected as system clock

10: PLL selected as system clock

11: not allowed

### 5.3.4 RCC clock interrupt register (RCC\_CIR)

Address offset: 0x0C

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CSSC	PLLSAI RDYC	PLL2S RDYC	PLL RDYC	HSE RDYC	HSI RDYC	LSE RDYC	LSI RDYC
								w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PLLSAI RDYIE	PLL2S RDYIE	PLL RDYIE	HSE RDYIE	HSI RDYIE	LSE RDYIE	LSI RDYIE	CSSF	PLLSAI RDYF	PLL2S RDYF	PLL RDYF	HSE RDYF	HSI RDYF	LSE RDYF	LSI RDYF
rw	rw	rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r

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

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

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

0: No effect

1: Clear CSSF flag

Bit 22 **PLLSAIRDYC:** PLLSAI Ready Interrupt Clear

This bit is set by software to clear PLLSAIRDYF flag. It is reset by hardware when the PLLSAIRDYF is cleared.

0: PLLSAIRDYF not cleared

1: PLLSAIRDYF cleared

Bit 21 **PLL12SRDYC:** 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 **PLL12SRDYIE:** 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 PLLSAIRDYIE is set.  
It is cleared by software by setting the PLLSAIRDYC bit.  
0: No clock ready interrupt caused by PLLSAI lock  
1: Clock ready interrupt caused by PLLSAI lock
- Bit 5 **PLLI2SRDYF:** PLLI2S ready interrupt flag  
This bit is set by hardware when the PLLI2S is locked and PLLI2SRDYIE is set.  
It is cleared by software by setting the PLLI2SDYC bit.  
0: No clock ready interrupt caused by PLLI2S lock  
1: Clock ready interrupt caused by PLLI2S lock
- Bit 4 **PLLRDYF:** Main PLL (PLL) ready interrupt flag  
This bit is set by hardware when PLL is locked and PLLRDYIE is set.  
It is cleared by software setting the PLLRDYC bit.  
0: No clock ready interrupt caused by PLL lock  
1: Clock ready interrupt caused by PLL lock

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

This bit is set by hardware when External High Speed clock becomes stable and HSERDYIE 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 HSIRDYIE 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 LSERDYIE 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 LSIRDYIE is set.

It is cleared by software by setting the LSIRDYC bit.

0: No clock ready interrupt caused by the LSI oscillator

1: Clock ready interrupt caused by the LSI oscillator

### 5.3.5 RCC AHB1 peripheral reset register (RCC\_AHB1RSTR)

Address offset: 0x10

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	OTGH S RST	Res.	Res.	Res.	ETHMAC RST	Res.	DMA2D RST	DMA2 RST	DMA1 RST	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
Res.	Res.	Res.	CRCR ST	Res.	GPIOK RST	GPIOJ RST	GPIOI RST	GPIOH RST	GPIOGG RST	GPIOF RST	GPIOE RST	GPIOD RST	GPIOC RST	GPIOB RST	GPIOA RST
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

**Bit 29 OTGHSRST: USB OTG HS module reset**

This bit is set and cleared by software.

0: does not reset the USB OTG HS module

1: resets the USB OTG HS module

Bits 28:26 Reserved, must be kept at reset value.

- Bit 25 **ETHMACRST:** Ethernet MAC reset  
This bit is set and cleared by software.  
0: does not reset Ethernet MAC  
1: resets Ethernet MAC
- Bit 24 Reserved, must be kept at reset value.
- Bit 23 **DMA2DRST:** DMA2D reset  
This bit is set and reset by software.  
0: does not reset DMA2D  
1: resets DMA2D
- 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
- Bit 11 Reserved, must be kept at reset value.
- Bit 10 **GPIOKRST:** IO port K reset  
This bit is set and cleared by software.  
0: does not reset IO port K  
1: resets IO port K
- Bit 9 **GPIOJRST:** IO port J reset  
This bit is set and cleared by software.  
0: does not reset IO port J  
1: resets IO port J
- Bit 8 **GPIOIRST:** IO port I reset  
This bit is set and cleared by software.  
0: does not reset IO port I  
1: resets IO port I
- Bit 7 **GPIOHRST:** IO port H reset  
This bit is set and cleared by software.  
0: does not reset IO port H  
1: resets IO port H
- Bit 6 **GPIOGRST:** IO port G reset  
This bit is set and cleared by software.  
0: does not reset IO port G  
1: resets IO port G

**Bit 5 **GPIOFRST:** IO port F reset**

This bit is set and cleared by software.

- 0: does not reset IO port F
- 1: resets IO port F

**Bit 4 **GPIOERST:** IO port E reset**

This bit is set and cleared by software.

- 0: does not reset IO port E
- 1: resets IO port E

**Bit 3 **GPIODRST:** IO port D reset**

This bit is set and cleared by software.

- 0: does not reset IO port D
- 1: resets IO port D

**Bit 2 **GPIOCRST:** IO port C reset**

This bit is set and cleared by software.

- 0: does not reset IO port C
- 1: resets IO port C

**Bit 1 **GPIOBRST:** IO port B reset**

This bit is set and cleared by software.

- 0: does not reset IO port B
- 1: resets IO port B

**Bit 0 **GPIOARST:** IO port A reset**

This bit is set and cleared by software.

- 0: does not reset IO port A
- 1: resets IO port A

### 5.3.6 RCC AHB2 peripheral reset register (RCC\_AHB2RSTR)

Address offset: 0x14

Reset value: 0x0000 0000

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

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

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

Bit 7 **OTGFSRST:** USB OTG FS module reset

Set and cleared by software.

0: does not reset the USB OTG FS module

1: resets the USB OTG FS module

Bit 6 **RNGRST:** Random number generator module reset

Set and cleared by software.

0: does not reset the random number generator module

1: resets the random number generator module

Bit 5 **HASHRST:** Hash module reset

Set and cleared by software.

0: does not reset the HASH module

1: resets the HASH module

Bit 4 **CRYPRST:** Cryptographic module reset

Set and cleared by software.

0: does not reset the cryptographic module

1: resets the cryptographic module

Bits 3:1 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

### 5.3.7 RCC AHB3 peripheral reset register (RCC\_AHB3RSTR)

Address offset: 0x18

Reset value: 0x0000 0000

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

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

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

Bit 1 **QSPIRST**: Quad SPI memory controller reset

Set and cleared by software.

0: does not reset the QUADSPI memory controller

1: resets the QUADSPI memory controller

Bit 0 **FMCRST**: Flexible memory controller module reset

Set and cleared by software.

0: does not reset the FMC module

1: resets the FMC module

### 5.3.8 RCC APB1 peripheral reset register (RCC\_APB1RSTR)

Address offset: 0x20

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UART8R ST	UART7R ST	DACRST	PWR RST	CECRST	CAN2 RST	CAN1 RST	I2C4RST	I2C3 RST	I2C2 RST	I2C1 RST	UART5 RST	UART4 RST	UART3 RST	UART2 RST	SPDIFR XRST
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
SPI3 RST	SPI2 RST	Res.	Res.	WWDG RST	Res.	LPTIM1 RST	TIM14 RST	TIM13 RST	TIM12 RST	TIM7 RST	TIM6 RST	TIM5 RST	TIM4 RST	TIM3 RST	TIM2 RST
rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Bit 31 **UART8RST**: UART8 reset  
Set and cleared by software.  
0: does not reset UART8  
1: resets UART8
- Bit 30 **UART7RST**: UART7 reset  
Set and cleared by software.  
0: does not reset UART7  
1: resets UART7
- Bit 29 **DACRST**: DAC reset  
Set and cleared by software.  
0: does not reset the DAC interface  
1: resets the DAC interface
- Bit 28 **PWRRST**: Power interface reset  
Set and cleared by software.  
0: does not reset the power interface  
1: resets the power interface
- Bit 27 **CECRST**: HDMI-CEC reset  
Set and cleared by software.  
0: does not reset HDMI-CEC  
1: resets HDMI-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 **I2C4RST**: I2C4 reset  
Set and cleared by software.  
0: does not reset I2C4  
1: resets I2C4
- 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:** SPDIFRX reset  
Set and cleared by software.  
0: does not reset SPDIFRX  
1: resets SPDIFRX
- Bit 15 **SPI3RST:** SPI3 reset  
Set and cleared by software.  
0: does not reset SPI3  
1: resets SPI3
- Bit 14 **SPI2RST:** SPI2 reset  
Set and cleared by software.  
0: does not reset SPI2  
1: resets SPI2
- Bits 13:12 Reserved, must be kept at reset value.
- Bit 11 **WWDGRST:** Window watchdog reset  
Set and cleared by software.  
0: does not reset the window watchdog  
1: resets the window watchdog
- Bit 10 Reserved, must be kept at reset value.
- Bit 9 **LPTIM1RST:** Low-power timer 1 reset  
Set and cleared by software.  
0: does not reset LPTM11  
1: resets LPTM11
- Bit 8 **TIM14RST:** TIM14 reset  
Set and cleared by software.  
0: does not reset TIM14  
1: resets TIM14
- Bit 7 **TIM13RST:** TIM13 reset  
Set and cleared by software.  
0: does not reset TIM13  
1: resets TIM13

Bit 6 **TIM12RST:** TIM12 reset

Set and cleared by software.  
0: does not reset TIM12  
1: resets TIM12

Bit 5 **TIM7RST:** TIM7 reset

Set and cleared by software.  
0: does not reset TIM7  
1: resets TIM7

Bit 4 **TIM6RST:** TIM6 reset

Set and cleared by software.  
0: does not reset TIM6  
1: resets TIM6

Bit 3 **TIM5RST:** TIM5 reset

Set and cleared by software.  
0: does not reset TIM5  
1: resets TIM5

Bit 2 **TIM4RST:** TIM4 reset

Set and cleared by software.  
0: does not reset TIM4  
1: resets TIM4

Bit 1 **TIM3RST:** TIM3 reset

Set and cleared by software.  
0: does not reset TIM3  
1: resets TIM3

Bit 0 **TIM2RST:** TIM2 reset

Set and cleared by software.  
0: does not reset TIM2  
1: resets TIM2

### 5.3.9 RCC APB2 peripheral reset register (RCC\_APB2RSTR)

Address offset: 0x24

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	LTDC RST	Res.	Res.	SAI2 RST	SAI1 RST	SPI6 RST	SPI5 RST	Res.	TIM11 RST	TIM10 RST	TIM9 RST
					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.	SYSCFG RST	SPI4 RST	SPI1 RST	SDMMC1 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 31:27 Reserved, must be kept at reset value.

Bit 26 **LTDCRST**: LTDC reset

This bit is set and reset by software.

- 0: does not reset LCD-TFT
- 1: resets LCD-TFT

Bits 27:24 Reserved, must be kept at reset value.

Bit 23 **SAI2RST**: SAI2 reset

This bit is set and cleared by software.

- 0: does not reset SAI2
- 1: resets SAI2

Bit 22 **SAI1RST**: SAI1 reset

This bit is set and reset by software.

- 0: does not reset SAI1
- 1: resets SAI1

Bit 21 **SPI6RST**: SPI6 reset

This bit is set and cleared by software.

- 0: does not reset SPI6
- 1: resets SPI6

Bit 20 **SPI5RST**: SPI5 reset

This bit is set and cleared by software.

- 0: does not reset SPI5
- 1: resets SPI5

Bit 19 Reserved, must be kept at reset value.

Bit 18 **TIM11RST**: TIM11 reset

This bit is set and cleared by software.

- 0: does not reset TIM11
- 1: resets TIM14

Bit 17 **TIM10RST**: TIM10 reset

This bit is set and cleared by software.

- 0: does not reset TIM10
- 1: resets TIM10

Bit 16 **TIM9RST:** TIM9 reset

This bit is set and cleared by software.

0: does not reset TIM9

1: resets TIM9

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

Bit 14 **SYSCFGRST:** System configuration controller reset

This bit is set and cleared by software.

0: does not reset the System configuration controller

1: resets the System configuration controller

Bit 13 **SPI4RST:** SPI4 reset

This bit is set and cleared by software.

0: does not reset SPI4

1: resets SPI4

Bit 12 **SPI1RST:** SPI1 reset

This bit is set and cleared by software.

0: does not reset SPI1

1: resets SPI1

Bit 11 **SDMMC1RST:** SDMMC1 reset

This bit is set and cleared by software.

0: does not reset the SDMMC1 module

1: resets the SDMMC1 module

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

Bit 8 **ADCRST:** ADC interface reset (common to all ADCs)

This bit is set and cleared by software.

0: does not reset the ADC interface

1: resets the ADC interface

## 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

### 5.3.10 RCC AHB1 peripheral clock register (RCC\_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OTGHS ULPIEN	OTGHS EN	ETHM ACPTP EN	ETHM ACRX EN	ETHM ACTX EN	ETHMA CEN	Res.	DMA2D EN	DMA2 EN	DMA1 EN	DTCMRA MEN	Res.	BKPSR AMEN	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
Res.	Res.	Res.	CRC EN	Res.	GPIOK EN	GPIOJ EN	GPIOI EN	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN	
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	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

Bit 28 **ETHMACPTPEN:** Ethernet PTP clock enable

This bit is set and cleared by software.

- 0: Ethernet PTP clock disabled
- 1: Ethernet PTP clock enabled

Bit 27 **ETHMACRXEN:** Ethernet Reception clock enable

This bit is set and cleared by software.

- 0: Ethernet Reception clock disabled
- 1: Ethernet Reception clock enabled

Bit 26 **ETHMACTXEN:** Ethernet Transmission clock enable

This bit is set and cleared by software.

- 0: Ethernet Transmission clock disabled
- 1: Ethernet Transmission clock enabled

Bit 25 **ETHMACEN:** Ethernet MAC clock enable

This bit is set and cleared by software.

- 0: Ethernet MAC clock disabled
- 1: Ethernet MAC clock enabled

Bit 24 Reserved, must be kept at reset value.

Bit 23 **DMA2DEN:** DMA2D clock enable

This bit is set and cleared by software.

- 0: DMA2D clock disabled
- 1: DMA2D clock enabled

- Bit 22 **DMA2EN:** DMA2 clock enable  
This bit is set and cleared by software.  
0: DMA2 clock disabled  
1: DMA2 clock enabled
- Bit 21 **DMA1EN:** DMA1 clock enable  
This bit is set and cleared by software.  
0: DMA1 clock disabled  
1: DMA1 clock enabled
- Bit 20 **DTCMRAMEN:** DTCM data RAM clock enable  
This bit is set and cleared by software.  
0: DTCM RAM clock disabled  
1: DTCM RAM clock enabled
- Bit 19 Reserved, must be kept at reset value.
- Bit 18 **BKPSRAMEN:** Backup SRAM interface clock enable  
This bit is set and cleared by software.  
0: Backup SRAM interface clock disabled  
1: Backup SRAM interface clock enabled
- Bits 17:13 Reserved, must be kept at reset value.
- Bit 12 **CRCEN:** CRC clock enable  
This bit is set and cleared by software.  
0: CRC clock disabled  
1: CRC clock enabled
- Bit 11 Reserved, must be kept at reset value.
- Bit 10 **GPIOKEN:** IO port K clock enable  
This bit is set and cleared by software.  
0: IO port K clock disabled  
1: IO port K clock enabled
- Bit 9 **GPIOJEN:** IO port J clock enable  
This bit is set and cleared by software.  
0: IO port J clock disabled  
1: IO port J clock enabled
- Bit 8 **GPIOIEN:** IO port I clock enable  
This bit is set and cleared by software.  
0: IO port I clock disabled  
1: IO port I clock enabled
- Bit 7 **GPIOHEN:** IO port H clock enable  
This bit is set and cleared by software.  
0: IO port H clock disabled  
1: IO port H clock enabled
- Bit 6 **GPIOGEN:** IO port G clock enable  
This bit is set and cleared by software.  
0: IO port G clock disabled  
1: IO port G clock enabled

Bit 5 **GPIOFEN**: IO port F clock enable

This bit is set and cleared by software.

0: IO port F clock disabled

1: IO port F clock enabled

Bit 4 **GPIOEEN**: IO port E clock enable

This bit is set and cleared by software.

0: IO port E clock disabled

1: IO port E clock enabled

Bit 3 **GPIODEN**: IO port D clock enable

This bit is set and cleared by software.

0: IO port D clock disabled

1: IO port D clock enabled

Bit 2 **GPIOCEN**: IO port C clock enable

This bit is set and cleared by software.

0: IO port C clock disabled

1: IO port C clock enabled

Bit 1 **GPIOBEN**: IO port B clock enable

This bit is set and cleared by software.

0: IO port B clock disabled

1: IO port B clock enabled

Bit 0 **GPIOAEN**: IO port A clock enable

This bit is set and cleared by software.

0: IO port A clock disabled

1: IO port A clock enabled

### 5.3.11 RCC AHB2 peripheral clock enable register (RCC\_AHB2ENR)

Address offset: 0x34

Reset value: 0x0000 0000

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

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

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

Bit 7 **OTGFSEN**: USB OTG FS clock enable

This bit is set and cleared by software.

0: USB OTG FS clock disabled

1: USB OTG FS clock enabled

Bit 6 **RNGEN**: Random number generator clock enable

This bit is set and cleared by software.

0: Random number generator clock disabled

1: Random number generator clock enabled

Bit 5 **HASHEN**: Hash modules clock enable

This bit is set and cleared by software.

0: Hash modules clock disabled

1: Hash modules clock enabled

Bit 4 **CRYPEN**: Cryptographic modules clock enable

This bit is set and cleared by software.

0: cryptographic module clock disabled

1: cryptographic module clock enabled

Bits 3:1 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

### 5.3.12 RCC AHB3 peripheral clock enable register (RCC\_AHB3ENR)

Address offset: 0x38

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	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**: Quad SPI memory controller clock enable

This bit is set and cleared by software.

0: QUASPI controller clock disabled

1: QUASPI controller clock enabled

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

This bit is set and cleared by software.

0: FMC clock disabled

1: FMC clock enabled

### 5.3.13 RCC APB1 peripheral clock enable register (RCC\_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UART8 EN	UART7 EN	DAC EN	PWR EN	CEC EN	CAN2 EN	CAN1 EN	I2C4 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	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.	LPTIM1 EN	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UART8EN**: UART8 clock enable

This bit is set and cleared by software.

- 0: UART8 clock disabled
- 1: UART8 clock enabled

Bit 30 **UART7EN**: UART7 clock enable

This bit is set and cleared by software.

- 0: UART7 clock disabled
- 1: UART7 clock enabled

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

This bit is set and cleared by software.

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

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

This bit is set and cleared by software.

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

Bit 27 **CECEN**: HDMI-CEC clock enable

This bit is set and cleared by software.

- 0: HDMI-CEC clock disabled
- 1: HDMI-CEC 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 **I2C4**: I2C4 clock enable

This bit is set and cleared by software.

- 0: I2C4 clock disabled
- 1: I2C4 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:** SPDIFRX clock enable  
This bit is set and cleared by software.  
0: SPDIFRX clock disabled  
1: SPDIFRX clock enable
- 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 by software to enable the window watchdog clock. It is reset by hardware system reset. This bit can also be set by hardware if the WWDG\_SW option bit is reset.  
0: Window watchdog clock disabled  
1: Window watchdog clock enabled
- Bit 10 Reserved, must be kept at reset value.
- Bit 9 **LPTM1EN:** Low-power timer 1 clock enable  
This bit is set and cleared by software.  
0: LPTIM1 clock disabled  
1: LPTIM1 clock enabled
- Bit 8 **TIM14EN:** TIM14 clock enable  
This bit is set and cleared by software.  
0: TIM14 clock disabled  
1: TIM14 clock enabled

**Bit 7 TIM13EN: TIM13 clock enable**

This bit is set and cleared by software.

0: TIM13 clock disabled

1: TIM13 clock enabled

**Bit 6 TIM12EN: TIM12 clock enable**

This bit is set and cleared by software.

0: TIM12 clock disabled

1: TIM12 clock enabled

**Bit 5 TIM7EN: TIM7 clock enable**

This bit is set and cleared by software.

0: TIM7 clock disabled

1: TIM7 clock enabled

**Bit 4 TIM6EN: TIM6 clock enable**

This bit is set and cleared by software.

0: TIM6 clock disabled

1: TIM6 clock enabled

**Bit 3 TIM5EN: TIM5 clock enable**

This bit is set and cleared by software.

0: TIM5 clock disabled

1: TIM5 clock enabled

**Bit 2 TIM4EN: TIM4 clock enable**

This bit is set and cleared by software.

0: TIM4 clock disabled

1: TIM4 clock enabled

**Bit 1 TIM3EN: TIM3 clock enable**

This bit is set and cleared by software.

0: TIM3 clock disabled

1: TIM3 clock enabled

**Bit 0 TIM2EN: TIM2 clock enable**

This bit is set and cleared by software.

0: TIM2 clock disabled

1: TIM2 clock enabled

### 5.3.14 RCC APB2 peripheral clock enable register (RCC\_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	LTDC EN	Res.	Res.	SAI2EN	SAI1EN	SPI6EN	SPI5EN	Res.	TIM11 EN	TIM10 EN	TIM9 EN
					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.	SYSCFG EN	SPI4 EN	SPI1 EN	SDMMC1 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 31:27 Reserved, must be kept at reset value.

Bit 26 **LTDCEN**: LTDC clock enable

This bit is set and cleared by software.

- 0: LTDC clock disabled
- 1: LTDC clock enabled

Bits 27: 24 Reserved, must be kept at reset value.

Bit 23 **SAI2EN**: SAI2 clock enable

This bit is set and cleared by software.

- 0: SAI2 clock disabled
- 1: SAI2 clock enabled

Bit 22 **SAI1EN**: SAI1 clock enable

This bit is set and cleared by software.

- 0: SAI1 clock disabled
- 1: SAI1 clock enabled

Bit 21 **SPI6EN**: SPI6 clock enable

This bit is set and cleared by software.

- 0: SPI6 clock disabled
- 1: SPI6 clock enabled

Bit 20 **SPI5EN**: SPI5 clock enable

This bit is set and cleared by software.

- 0: SPI5 clock disabled
- 1: SPI5 clock enabled

Bit 18 **TIM11EN**: TIM11 clock enable

This bit is set and cleared by software.

- 0: TIM11 clock disabled
- 1: TIM11 clock enabled

Bit 17 **TIM10EN**: TIM10 clock enable

This bit is set and cleared by software.

- 0: TIM10 clock disabled
- 1: TIM10 clock enabled

**Bit 16 **TIM9EN:** TIM9 clock enable**

This bit is set and cleared by software.

0: TIM9 clock disabled

1: TIM9 clock enabled

**Bit 15 Reserved, must be kept at reset value.****Bit 14 **SYSCKGEN:** System configuration controller clock enable**

This bit is set and cleared by software.

0: System configuration controller clock disabled

1: System configuration controller clock enabled

**Bit 13 **SPI4EN:** SPI4 clock enable**

This bit is set and cleared by software.

0: SPI4 clock disabled

1: SPI4 clock enabled

**Bit 12 **SPI1EN:** SPI1 clock enable**

This bit is set and cleared by software.

0: SPI1 clock disabled

1: SPI1 clock enabled

**Bit 11 **SDMMC1EN:** SDMMC1 clock enable**

This bit is set and cleared by software.

0: SDMMC1 module clock disabled

1: SDMMC1 module clock enabled

**Bit 10 **ADC3EN:** ADC3 clock enable**

This bit is set and cleared by software.

0: ADC3 clock disabled

1: ADC3 clock enabled

**Bit 9 **ADC2EN:** ADC2 clock enable**

This bit is set and cleared by software.

0: ADC2 clock disabled

1: ADC2 clock enabled

**Bit 8 **ADC1EN:** ADC1 clock enable**

This bit is set and cleared by software.

0: ADC1 clock disabled

1: ADC1 clock enabled

**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

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

Address offset: 0x50

Reset value: 0x7EF7 B7FFh

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OTGHS ULPI LPEN	OTGHS LPEN	ETHPTP LPEN	ETHRX LPEN	ETHTX LPEN	ETHMA C LPEN	Res.	DMA2D LPEN	DMA2 LPEN	DMA1 LPEN	DTCM LPEN	Res.	BKPS RAM LPEN	SRAM2 LPEN	SRAM1 LPEN
	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
FLITF LPEN	Res.	AXI LPEN	CRC LPEN	Res.	GPIOK LPEN	GPIOIJ LPEN	GPIOI LPEN	GPIOH LPEN	GPIOGG LPEN	GPIOF LPEN	GPIOE LPEN	GPIOD LPEN	GPIOC LPEN	GPIOB LPEN	GPIOA LPEN
	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	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

Bit 28 **ETHMACPTPLPEN:** Ethernet PTP clock enable during Sleep mode

This bit is set and cleared by software.

0: Ethernet PTP clock disabled during Sleep mode

1: Ethernet PTP clock enabled during Sleep mode

Bit 27 **ETHMACRXLPEN:** Ethernet reception clock enable during Sleep mode

This bit is set and cleared by software.

0: Ethernet reception clock disabled during Sleep mode

1: Ethernet reception clock enabled during Sleep mode

Bit 26 **ETHMACTXLPEN:** Ethernet transmission clock enable during Sleep mode

This bit is set and cleared by software.

0: Ethernet transmission clock disabled during sleep mode

1: Ethernet transmission clock enabled during sleep mode

Bit 25 **ETHMACLPEN:** Ethernet MAC clock enable during Sleep mode

This bit is set and cleared by software.

0: Ethernet MAC clock disabled during Sleep mode

1: Ethernet MAC clock enabled during Sleep mode

Bit 24 Reserved, must be kept at reset value.

Bit 23 **DMA2DLPEN:** DMA2D clock enable during Sleep mode

This bit is set and cleared by software.

0: DMA2D clock disabled during Sleep mode

1: DMA2D clock enabled during Sleep mode

Bit 22 **DMA2LPEN:** DMA2 clock enable during Sleep mode

This bit is set and cleared by software.

0: DMA2 clock disabled during Sleep mode

1: DMA2 clock enabled during Sleep mode

Bit 21 **DMA1LPEN:** DMA1 clock enable during Sleep mode

This bit is set and cleared by software.

0: DMA1 clock disabled during Sleep mode

1: DMA1 clock enabled during Sleep mode

Bit 20 **DTCMLPEN:** DTCM RAM interface clock enable during Sleep mode

This bit is set and cleared by software.

0: DTCM RAM interface clock disabled during Sleep mode

1: DTCM RAM interface clock enabled during Sleep mode

Bit 19 Reserved, must be kept at reset value.

Bit 18 **BKPSRAMLPEN:** Backup SRAM interface clock enable during Sleep mode

This bit is set and cleared by software.

0: Backup SRAM interface clock disabled during Sleep mode

1: Backup SRAM interface clock enabled during Sleep mode

Bit 17 **SRAM2LPEN:** SRAM2 interface clock enable during Sleep mode

This bit is set and cleared by software.

0: SRAM2 interface clock disabled during Sleep mode

1: SRAM2 interface clock enabled during Sleep mode

Bit 16 **SRAM1LPEN:** SRAM1 interface clock enable during Sleep mode

This bit is set and cleared by software.

0: SRAM1 interface clock disabled during Sleep mode

1: SRAM1 interface clock enabled during Sleep mode

Bit 15 **FLITFLPEN:** Flash interface clock enable during Sleep mode

This bit is set and cleared by software.

0: Flash interface clock disabled during Sleep mode

1: Flash interface clock enabled during Sleep mode

Bit 14 Reserved, must be kept at reset value.

Bit 13 **AXILPEN:** AXI to AHB bridge clock enable during Sleep mode

This bit is set and cleared by software.

0: AXI to AHB bridge clock disabled during Sleep mode

1: AXI to AHB bridge clock enabled during Sleep mode

Bit 12 **CRCLPEN:** CRC clock enable during Sleep mode

This bit is set and cleared by software.

0: CRC clock disabled during Sleep mode

1: CRC clock enabled during Sleep mode

Bit 11 Reserved, must be kept at reset value.

Bit 10 **GPIOKLPEN:** IO port K clock enable during Sleep mode

This bit is set and cleared by software.

0: IO port K clock disabled during Sleep mode

1: IO port K clock enabled during Sleep mode

Bit 9 **GPIOJLPEN:** IO port J clock enable during Sleep mode

This bit is set and cleared by software.

0: IO port J clock disabled during Sleep mode

1: IO port J clock enabled during Sleep mode

Bit 8 **GPIOILPEN:** IO port I clock enable during Sleep mode

This bit is set and cleared by software.

0: IO port I clock disabled during Sleep mode

1: IO port I clock enabled during Sleep mode

Bit 7 **GPIOHLPEN:** IO port H clock enable during Sleep mode

This bit is set and cleared by software.

0: IO port H clock disabled during Sleep mode

1: IO port H clock enabled during Sleep mode

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 **GPIOODLPEN:** 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

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

Address offset: 0x54

Reset value: 0x0000 00F1

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

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

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

Bit 7 **OTGFSLPEN:** USB OTG FS clock enable during Sleep mode

This bit is set and cleared by software.

0: USB OTG FS clock disabled during Sleep mode

1: USB OTG FS clock enabled during Sleep mode

Bit 6 **RNGLPEN:** Random number generator clock enable during Sleep mode

This bit is set and cleared by software.

0: Random number generator clock disabled during Sleep mode

1: Random number generator clock enabled during Sleep mode

Bit 5 **HASHPEN:** Hash modules clock enable during Sleep mode

This bit is set and cleared by software.

0: Hash modules clock disabled during Sleep mode

1: Hash modules clock enabled during Sleep mode

Bit 4 **CRYPLPEN:** Cryptography modules clock enable during Sleep mode

This bit is set and cleared by software.

0: cryptography modules clock disabled during Sleep mode

1: cryptography modules clock enabled during Sleep mode

Bits 3:1 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

### 5.3.17 RCC AHB3 peripheral clock enable in low-power mode register (RCC\_AHB3LPENR)

Address offset: 0x58

Reset value: 0x0000 0003

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

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

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

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

This bit is set and cleared by software.

0: QUADSPI controller clock disabled during Sleep mode

1: QUADSPI controller clock enabled during Sleep mode

Bit 0 **FMCCLPEN**: Flexible memory controller module clock enable during Sleep mode

This bit is set and cleared by software.

0: FMC module clock disabled during Sleep mode

1: FMC module clock enabled during Sleep mode

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

Address offset: 0x60

Reset value: 0xFFFF CBFFh

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UART8 LPEN	UART7 LPEN	DAC LPEN	PWR LPEN	CEC LPEN	CAN2 LPEN	CAN1 LPEN	I2C4 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.	LPTMI1 LPEN	TIM14 LPEN	TIM13 LPEN	TIM12 LPEN	TIM7 LPEN	TIM6 LPEN	TIM5 LPEN	TIM4 LPEN	TIM3 LPEN	TIM2 LPEN
rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UART8LPEN**: UART8 clock enable during Sleep mode

This bit is set and cleared by software.

0: UART8 clock disabled during Sleep mode

1: UART8 clock enabled during Sleep mode

Bit 30 **UART7LPEN**: UART7 clock enable during Sleep mode

This bit is set and cleared by software.

0: UART7 clock disabled during Sleep mode

1: UART7 clock enabled during Sleep mode

Bit 29 **DACLPE**N: 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 **PWRLPE**N: 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 **CECLPE**N: HDMI-CEC clock enable during Sleep mode

This bit is set and cleared by software.

0: HDMI-CEC clock disabled during Sleep mode

1: HDMI-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 **I2C4LPEN:** I2C4 clock enable during Sleep mode

This bit is set and cleared by software.

0: I2C4 clock disabled during Sleep mode

1: I2C4 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 **SPDIFRXLPEN:** SPDIFRX clock enable during Sleep mode

This bit is set and cleared by software.

0: SPDIFRX clock disabled during Sleep mode

1: SPDIFRX 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
- Bit 10 Reserved, must be kept at reset value.
- Bit 9 **LPTIM1LPEN:** low-power timer 1 clock enable during Sleep mode  
This bit is set and cleared by software.  
0: LPTIM1 clock disabled during Sleep mode  
1: LPTIM1 clock enabled during Sleep mode
- Bit 8 **TIM14LPEN:** TIM14 clock enable during Sleep mode  
This bit is set and cleared by software.  
0: TIM14 clock disabled during Sleep mode  
1: TIM14 clock enabled during Sleep mode
- Bit 7 **TIM13LPEN:** TIM13 clock enable during Sleep mode  
This bit is set and cleared by software.  
0: TIM13 clock disabled during Sleep mode  
1: TIM13 clock enabled during Sleep mode
- Bit 6 **TIM12LPEN:** TIM12 clock enable during Sleep mode  
This bit is set and cleared by software.  
0: TIM12 clock disabled during Sleep mode  
1: TIM12 clock enabled during Sleep mode
- Bit 5 **TIM7LPEN:** TIM7 clock enable during Sleep mode  
This bit is set and cleared by software.  
0: TIM7 clock disabled during Sleep mode  
1: TIM7 clock enabled during Sleep mode
- Bit 4 **TIM6LPEN:** TIM6 clock enable during Sleep mode  
This bit is set and cleared by software.  
0: TIM6 clock disabled during Sleep mode  
1: TIM6 clock enabled during Sleep mode
- Bit 3 **TIM5LPEN:** TIM5 clock enable during Sleep mode  
This bit is set and cleared by software.  
0: TIM5 clock disabled during Sleep mode  
1: TIM5 clock enabled during Sleep mode

Bit 2 **TIM4LPEN:** TIM4 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM4 clock disabled during Sleep mode

1: TIM4 clock enabled during Sleep mode

Bit 1 **TIM3LPEN:** TIM3 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM3 clock disabled during Sleep mode

1: TIM3 clock enabled during Sleep mode

Bit 0 **TIM2LPEN:** TIM2 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM2 clock disabled during Sleep mode

1: TIM2 clock enabled during Sleep mode

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

Address offset: 0x64

Reset value: 0x04F7 7F33h

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	LTDC LPEN	Res.	Res.	SAI2 LPEN	SAI1 LPEN	SPI6 LPEN	SPI5 LPEN	Res.	TIM11 LPEN	TIM10 LPEN	TIM9 LPEN
					rw			rw	rw	rw	rw		rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SYSCFG LPEN	SPI4 LPEN	SPI1 LPEN	SDMMC1 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 31:27 Reserved, must be kept at reset value.

Bit 26 **LTDCLPEN**: LTDC clock enable during Sleep mode

This bit is set and cleared by software.

0: LTDC clock disabled during Sleep mode

1: LTDC clock enabled during Sleep mode

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

Bit 23 **SAI2LPEN**: SAI2 clock enable during Sleep mode

This bit is set and cleared by software.

0: SAI2 clock disabled during Sleep mode

1: SAI2 clock enabled during Sleep mode

Bit 22 **SAI1LPEN**: SAI1 clock enable during Sleep mode

This bit is set and cleared by software.

0: SAI1 clock disabled during Sleep mode

1: SAI1 clock enabled during Sleep mode

Bit 21 **SPI6LPEN**: SPI6 clock enable during Sleep mode

This bit is set and cleared by software.

0: SPI6 clock disabled during Sleep mode

1: SPI6 clock enabled during Sleep mode

Bit 20 **SPI5LPEN**: SPI5 clock enable during Sleep mode

This bit is set and cleared by software.

0: SPI5 clock disabled during Sleep mode

1: SPI5 clock enabled during Sleep mode

Bit 19 Reserved, must be kept at reset value.

Bit 18 **TIM11LPEN**: TIM11 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM11 clock disabled during Sleep mode

1: TIM11 clock enabled during Sleep mode

Bit 17 **TIM10LPEN**: TIM10 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM10 clock disabled during Sleep mode

1: TIM10 clock enabled during Sleep mode

Bit 16 **TIM9LPEN:** TIM9 clock enable during sleep mode

This bit is set and cleared by software.

0: TIM9 clock disabled during Sleep mode

1: TIM9 clock enabled during Sleep mode

Bit 15 Reserved, must be kept at reset value.

Bit 14 **SYSCFG\_LPEN:** System configuration controller clock enable during Sleep mode

This bit is set and cleared by software.

0: System configuration controller clock disabled during Sleep mode

1: System configuration controller clock enabled during Sleep mode

Bit 13 **SPI4LPEN:** SPI4 clock enable during Sleep mode

This bit is set and cleared by software.

0: SPI4 clock disabled during Sleep mode

1: SPI4 clock enabled during Sleep mode

Bit 12 **SPI1LPEN:** SPI1 clock enable during Sleep mode

This bit is set and cleared by software.

0: SPI1 clock disabled during Sleep mode

1: SPI1 clock enabled during Sleep mode

Bit 11 **SDMMC1LPEN:** SDMMC1 clock enable during Sleep mode

This bit is set and cleared by software.

0: SDMMC1 module clock disabled during Sleep mode

1: SDMMC1 module clock enabled during Sleep mode

Bit 10 **ADC3LPEN:** ADC 3 clock enable during Sleep mode

This bit is set and cleared by software.

0: ADC 3 clock disabled during Sleep mode

1: ADC 3 clock enabled during Sleep mode

Bit 9 **ADC2LPEN:** ADC2 clock enable during Sleep mode

This bit is set and cleared by software.

0: ADC2 clock disabled during Sleep mode

1: ADC2 clock enabled during Sleep mode

Bit 8 **ADC1LPEN:** ADC1 clock enable during Sleep mode

This bit is set and cleared by software.

0: ADC1 clock disabled during Sleep mode

1: ADC1 clock enabled during Sleep mode

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

### 5.3.20 RCC backup domain control register (RCC\_BDCR)

Address offset: 0x70

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

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

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

The LSEON, LSEBYP, LSEDRV[1:0], RTCSEL and RTCEN bits in the [RCC backup domain control register \(RCC\\_BDCR\)](#) are in the Backup domain. As a result, after Reset, these bits are write-protected and the DBP bit in the [PWR power control register \(PWR\\_CR1\)](#) has to be set before these can be modified. Refer to [Section 5.1.1: System reset on page 133](#) for further information. These bits are only reset after a Backup domain Reset (see [Section 5.1.3: Backup domain reset](#)). Any internal or external Reset will not have any effect on these bits.

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

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

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

This bit is set and cleared by software.

0: Reset not activated

1: Resets the entire Backup domain

*Note: The BKPSRAM is not affected by this reset, the only way of resetting the BKPSRAM is through the Flash interface when a protection level change from level 1 to level 0 is requested.*

Bit 15 **RTCEN:** RTC clock enable

This bit is set and cleared by software.

0: RTC clock disabled

1: RTC clock enabled

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

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

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

00: No clock

01: LSE oscillator clock used as the RTC clock

10: LSI oscillator clock used as the RTC clock

11: HSE oscillator clock divided by a programmable prescaler (selection through the RTCPRE[4:0] bits in the RCC clock configuration register (RCC\_CFG)) used as the RTC clock

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

Bits 4:3 **LSEDRV[1:0]:** LSE oscillator drive capability

Set by software to modulate the LSE oscillator's drive capability.

00: Low driving capability

01: Medium high driving capability

10: Medium low driving capability

11: High driving capability

Bit 2 **LSEBYP:** External low-speed oscillator bypass

This bit is set and cleared by software to bypass the oscillator. This bit can be written only when the LSE clock is disabled.

0: LSE oscillator not bypassed

1: LSE oscillator bypassed

Bit 1 **LSERDY:** External low-speed oscillator ready

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

0: LSE clock not ready

1: LSE clock ready

Bit 0 **LSEON:** External low-speed oscillator enable

This bit is set and cleared by software.

0: LSE clock OFF

1: LSE clock ON

**5.3.21 RCC clock control & status register (RCC\_CSR)**

Address offset: 0x74

Reset value: 0x0E00 0000, reset by system reset, except reset flags by power reset only.

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

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

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

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

This bit is set by hardware when a Low-power management reset occurs.

Cleared by writing to the RMVF bit.

0: No Low-power management reset occurred

1: Low-power management reset occurred

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

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

This bit is set by hardware when a window watchdog reset occurs.

Cleared by writing to the RMVF bit.

0: No window watchdog reset occurred

1: Window watchdog reset occurred

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

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

Cleared by writing to the RMVF bit.

0: No watchdog reset occurred

1: Watchdog reset occurred

**Bit 28 SFRSTF:** Software reset flag

This bit is set by hardware when a software reset occurs.

Cleared by writing to the RMVF bit.

0: No software reset occurred

1: Software reset occurred

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

This bit is set by hardware when a POR/PDR reset occurs.

Cleared by writing to the RMVF bit.

0: No POR/PDR reset occurred

1: POR/PDR reset occurred

**Bit 26 PINRSTF:** PIN reset flag

This bit is set by hardware when a reset from the NRST pin occurs.

Cleared by writing to the RMVF bit.

0: No reset from NRST pin occurred

1: Reset from NRST pin occurred

**Bit 25 BORRSTF:** BOR reset flag

Cleared by software by writing the RMVF bit.

This bit is set by hardware when a POR/PDR or BOR reset occurs.

0: No POR/PDR or BOR reset occurred

1: POR/PDR or BOR reset occurred

**Bit 24 RMVF:** Remove reset flag

This bit is set by software to clear the reset flags.

0: No effect

1: Clear the reset flags

Bits 23:2 Reserved, must be kept at reset value.

**Bit 1 LSIRDY:** Internal low-speed oscillator ready

This bit is set and cleared by hardware to indicate when the internal RC 40 kHz oscillator is stable. After the LSION bit is cleared, LSIRDY goes low after 3 LSI clock cycles.

0: LSI RC oscillator not ready

1: LSI RC oscillator ready

**Bit 0 LSION:** Internal low-speed oscillator enable

This bit is set and cleared by software.

0: LSI RC oscillator OFF

1: LSI RC oscillator ON

### 5.3.22 RCC spread spectrum clock generation register (RCC\_SSCGR)

Address offset: 0x80

Reset value: 0x0000 0000

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

The spread spectrum clock generation is available only for the main PLL.

The RCC\_SSCGR register must be written either before the main PLL is enabled or after the main PLL disabled.

**Note:** *For full details about PLL spread spectrum clock generation (SSCG) characteristics, refer to the “Electrical characteristics” section in your device datasheet.*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SSCG EN	SPR EAD SEL	Res.	Res.												
rw	rw			rw	rw	rw		rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INCSTEP				MODPER											
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

**Bit 31 SSCGEN:** Spread spectrum modulation enable

This bit is set and cleared by software.

0: Spread spectrum modulation DISABLE. (To write after clearing CR[24]=PLLON bit)

1: Spread spectrum modulation ENABLE. (To write before setting CR[24]=PLLON bit)

**Bit 30 SPREADSEL:** Spread Select

This bit is set and cleared by software.

To write before to set CR[24]=PLLON bit.

0: Center spread

1: Down spread

Bits 29:28 Reserved, must be kept at reset value.

Bits 27:13 **INCSTEP**: Incrementation step

These bits are set and cleared by software. To write before setting CR[24]=PLLON bit.  
Configuration input for modulation profile amplitude.

Bits 12:0 **MODPER**: Modulation period

These bits are set and cleared by software. To write before setting CR[24]=PLLON bit.  
Configuration input for modulation profile period.

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

Address offset: 0x84

Reset value: 0x2400 3000

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

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

$$f_{(VCO\ clock)} = f_{(PLL2S\ clock\ input)} \times (PLL2SN / PLLM)$$

$$f_{(PLL2S\_P)} = f_{(VCO\ clock)} / PLL2SP$$

$$f_{(PLL2S\_Q)} = f_{(VCO\ clock)} / PLL2SQ$$

$$f_{(PLL2S\_R)} = f_{(VCO\ clock)} / PLL2SR$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	PLL2SR[2:0]				PLL2SQ[0:3]				Res.	Res.	Res.	Res.	Res.	Res.	PLL2SP[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.	PLL2SN[8:0]								Res.							
	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 **PLL12SR[2:0]**: PLLI2S division factor for I2S clocks

These bits are set and cleared by software to control the I2S clock frequency. These bits should be written only if the PLLI2S is disabled. The factor must be chosen in accordance with the prescaler values inside the I2S peripherals, to reach 0.3% error when using standard crystals and 0% error with audio crystals. For more information about I2S clock frequency and precision, refer to [Section 32.7.4: Start-up description](#) in the I2S chapter.

**Caution:** The I2Ss requires a frequency lower than or equal to 192 MHz to work correctly.

I2S clock frequency = VCO frequency / PLLR with  $2 \leq \text{PLLR} \leq 7$

000: PLLR = 0, wrong configuration

001: PLLR = 1, wrong configuration

010: PLLR = 2

...

111: PLLR = 7

Bits 27:24 **PLL12SQ[3:0]**: PLLI2S division factor for SAIs clock

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

They should be written when the PLLI2S is disabled.

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

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 **PLL12SP[1:0]**: PLLI2S division factor for SPDIFRX clock

These bits are set and cleared by software to control the SPDIFRX clock. These bits can be written only if the PLLI2S is disabled.

The factor must be chosen in accordance with the prescaler values inside the SPDIF to reach an audio clock close to 44 kHz or 48 kHz according to the SPDIF mode

SPDIF clock frequency = VCO frequency / PLL12SP with PLLI2S P = 2, 4, 6, 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  $\times$  PLLI2SN with  $50 \leq \text{PLLI2SN} \leq 432$

00000000: PLLI2SN = 0, wrong configuration

00000001: PLLI2SN = 1, wrong configuration

...

001100010: PLLI2SN = 50

...

001100011: PLLI2SN = 99

001100100: PLLI2SN = 100

001100101: PLLI2SN = 101

001100110: PLLI2SN = 102

...

110110000: PLLI2SN = 432

110110000: PLLI2SN = 433, wrong configuration

...

111111111: PLLI2SN = 511, wrong configuration

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

Bits 5:0 Reserved, must be kept at reset value.

### 5.3.24 RCC PLLSAI configuration register (RCC\_PLLSAICFGR)

Address offset: 0x88

Reset value: 0x2400 3000

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

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

- $f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLLSAIN / PLLM)$
- $f_{(PLLSAI\_P)} = f_{(VCO\ clock)} / PLLSAIP$
- $f_{(PLLSAI\_Q)} = f_{(VCO\ clock)} / PLLSAIQ$
- $f_{(PLLSAI\_R)} = f_{(VCO\ clock)} / PLLSAIR$

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

Bit 31 Reserved, must be kept at reset value.

Bits 30:28 **PLLSAIR[2:0]: PLLSAI division factor for LCD clock**

Set and reset by software to control the LCD clock frequency.

These bits should be written when the PLLSAI is disabled.

LCD clock frequency = VCO frequency / PLLSAIR with  $2 \leq \text{PLLSAIR} \leq 7$

000: PLLSAIR = 0, wrong configuration

001: PLLSAIR = 1, wrong configuration

010: PLLSAIR = 2

...

111: PLLSAIR = 7

Bits 27:24 **PLLSAIQ[3:0]: PLLSAI division factor for SAI clock**

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

These bits should be written when the PLLSAI is disabled.

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

0000: PLLSAIQ = 0, wrong configuration

0001: PLLSAIQ = 1, wrong configuration

...

0010: PLLSAIQ = 2

0011: PLLSAIQ = 3

0100: PLLSAIQ = 4

0101: PLLSAIQ = 5

...

1111: PLLSAIQ = 15

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

Bits 17:16 **PLLSAIP[1:0]**: PLLSAI division factor for 48MHz clock

Set and reset by software to control the frequency of the PLLSAI output clock (PLLSAI48CLK). This output can be selected for USB, RNG, SDMMC (48 MHz clock). These bits should be written only if the PLLSAI is disabled.

Only half-word and word accesses are allowed to write these bits.

PLLSAI48 output clock frequency = VCO frequency / PLLSAIP with PLLSAI P = 2, 4, 6, or 8

00: PLLSAIP = 2

01: PLLSAIP = 4

10: PLLSAIP = 6

11: PLLSAIP = 8

Bit 15 Reserved, must be kept at reset value.

Bits 14:6 **PLLSAIN[8:0]**: PLLSAI division factor for VCO

Set and reset by software to control the multiplication factor of the VCO.

These bits should be written when the PLLSAI is disabled.

Only half-word and word accesses are allowed to write these bits.

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

VCO output frequency = VCO input frequency x PLLSAIN with  $50 \leq \text{PLLSAIN} \leq 432$

00000000: PLLSAIN = 0, wrong configuration

00000001: PLLSAIN = 1, wrong configuration

.....

001100010: PLLSAIN = 50

...

001100011: PLLSAIN = 99

001100100: PLLSAIN = 100

001100101: PLLSAIN = 101

001100110: PLLSAIN = 102

.....

110110000: PLLSAIN = 432

110110000: PLLSAIN = 433, wrong configuration

.....

111111111: PLLSAIN = 511, wrong configuration

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

Bits 5:0 Reserved, must be kept at reset value

### 5.3.25 RCC dedicated clocks configuration register (RCC\_DCKCFGR1)

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 and LTDC peripherals according to the following formula:

$$f_{(\text{PLLSAIDIVQ clock output})} = f_{(\text{PLLSAI}_Q)} / \text{PLLSAIDIVQ}$$

$$f_{(\text{PLLSAIDIVR clock output})} = f_{(\text{PLLSAI}_R)} / \text{PLLSAIDIVR}$$

$$f_{(\text{PLLI2SDIVQ clock output})} = f_{(\text{PLLI2S}_Q)} / \text{PLLI2SDIVQ}$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TIMPRE	SAI2SEL[1:0]	SAI1SEL[1:0]	Res.	Res.	Res.	PLLSAIDIVR[1:0]		
							rw	rw	rw	rw	rw			rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	PLLSAIDIVQ[4:0]					Res.	Res.	Res.	PLLI2SDIVQ[4:0]				
			rw	rw	rw	rw	rw				rw	rw	rw	rw	rw

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

Bit 24 **TIMPRE**: Timers clocks prescalers selection

This bit is set and reset by software to control the clock frequency of all the timers connected to APB1 and APB2 domain.

0: If the APB prescaler (PPRE1, PPRE2 in the RCC\_CFG 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\_CFG 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 **SAI2SEL[1:0]**: SAI2 clock source selection:

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

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

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

01: SAI2 clock frequency =  $f(\text{PLLI2S}_Q) / \text{PLLI2SDIVQ}$

10: SAI2 clock frequency = Alternate function input frequency

11: wrong configuration

Bits 21:20 **SAI1SEL[1:0]**: SAI1 clock source selection

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

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

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

01: SAI1 clock frequency =  $f(\text{PLLI2S}_Q) / \text{PLLI2SDIVQ}$

10: SAI1 clock frequency = Alternate function input frequency

11: wrong configuration

Bits 19: 18 Reserved, must be kept at reset value.

Bits 17:16 **PLLSAIDIVR[1:0]**: division factor for LCD\_CLK

These bits are set and cleared by software to control the frequency of LCD\_CLK.

They should be written only if PLLSAI is disabled.

LCD\_CLK frequency =  $f(\text{PLLSAI}_R) / \text{PLLSAIDIVR}$  with  $2 \leq \text{PLLSAIDIVR} \leq 16$

00: PLLSAIDIVR = /2

01: PLLSAIDIVR = /4

10: PLLSAIDIVR = /8

11: PLLSAIDIVR = /16

Bits 15: 13 Reserved, must be kept at reset value.

Bits 12:8 **PLLSAIDIVQ[4:0]**: PLLSAI division factor for SAI1 clock

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

They should be written only if PLLSAI is disabled.

SAI1 clock frequency =  $f(\text{PLLSAI\_Q}) / \text{PLLSAIDIVQ}$  with  $1 \leq \text{PLLSAIDIVQ} \leq 31$

00000: PLLSAIDIVQ = /1

00001: PLLSAIDIVQ = /2

00010: PLLSAIDIVQ = /3

00011: PLLSAIDIVQ = /4

00100: PLLSAIDIVQ = /5

...

11111: PLLSAIDIVQ = /32

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

Bits 4:0 **PLL2SDIV[4:0]**: PLLI2S division factor for SAI1 clock

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

They should be written only if PLLI2S is disabled.

SAI1 clock frequency =  $f(\text{PLLI2S\_Q}) / \text{PLLI2SDIVQ}$  with  $1 \leq \text{PLLI2SDIVQ} \leq 31$

00000: PLLI2SDIVQ = /1

00001: PLLI2SDIVQ = /2

00010: PLLI2SDIVQ = /3

00011: PLLI2SDIVQ = /4

00100: PLLI2SDIVQ = /5

...

11111: PLLI2SDIVQ = /32

### 5.3.26 RCC dedicated clocks configuration register (DCKCFGR2)

Address: 0x90h

Reset value: 0x0000 0000h

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

This register allows to select the source clock for the 48MHz, SDMMC, HDMI-CEC, LPTIM1, UARTs, USARTs and I2Cs clocks.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	SDMM C1SEL	CK48M SEL	CECSE L	LPTIM1SEL		I2C4SEL		I2C3SEL		I2C2SEL		I2C1SEL		
			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	
UART8SEL			UART7SEL		USART6SEL		UART5SEL		UART4SEL		UART3SEL		UART2SEL		UART1SEL	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

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

Bit 28 **SDMMCSEL**: SDMMC clock source selection

Set and reset by software.

0: 48 MHz clock is selected as SDMMC clock

1: System clock is selected as SDMMC clock

Bit 27 **CK48MSEL**: 48MHz clock source selection

Set and reset by software.

0: 48MHz clock from PLL is selected

1: 48MHz clock from PLLSAI is selected.

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

Set and reset by software.

0: LSE clock is selected as HDMI-CEC clock

1: HSI divided by 488 clock is selected as HDMI-CEC clock

Bits 25:24 **LPTIM1SEL**: Low-power timer 1 clock source selection

Set and reset by software.

00: APB1 clock (PCLK1) selected as LPTILM1 clock

01: LSI clock is selected as LPTILM1 clock

10: HSI clock is selected as LPTILM1 clock

11: LSE clock is selected as LPTILM1 clock

Bits 23:22 **I2C4SEL**: I2C4 clock source selection

Set and reset by software.

00: APB1 clock ( PCLK1) is selected as I2C4 clock

01: System clock is selected as I2C4 clock

10: HSI clock is selected as I2C4 clock

11: reserved

Bits 21:20 **I2C3SEL**: I2C3 clock source selection

Set and reset by software.

00: APB clock is selected as I2C3 clock

01: System clock is selected as I2C3 clock

10: HSI clock is selected as I2C3 clock

11: reserved

Bits 19:18 **I2C2SEL**: I2C2 clock source selection

Set and reset by software.

00: APB1 clock (PCLK1) is selected as I2C2 clock

01: System clock is selected as I2C2 clock

10: HSI clock is selected as I2C2 clock

11: reserved

Bits 17:16 **I2C1SEL**: I2C1 clock source selection

Set and reset by software.

00: APB clock (PCLK1) is selected as I2C1 clock

01: System clock is selected as I2C1 clock

10: HSI clock is selected as I2C1 clock

11: reserved

Bits 15:14 **UART8SEL[1:0]**: UART 8 clock source selection

Set and reset by software.

- 00: APB1 clock (PCLK1) is selected as UART 8 clock
- 01: System clock is selected as UART 8 clock
- 10: HSI clock is selected as UART 8 clock
- 11: LSE clock is selected as UART 8 clock

Bits 13:12 **UART7SEL[1:0]**: UART 7 clock source selection

Set and reset by software.

- 00: APB1 clock (PCLK1) is selected as UART 7 clock
- 01: System clock is selected as UART 7 clock
- 10: HSI clock is selected as UART 7 clock
- 11: LSE clock is selected as UART 7 clock

Bits 11:10 **USART6SEL[1:0]**: USART 6 clock source selection

Set and reset by software.

- 00: APB2 clock(PCLK2) is selected as USART 6 clock
- 01: System clock is selected as USART 6 clock
- 10: HSI clock is selected as USART 6 clock
- 11: LSE clock is selected as USART 6 clock

Bits 9:8 **UART5SEL[1:0]**: UART 5 clock source selection

Set and reset by software.

- 00: APB1 clock(PCLK1) is selected as UART 5 clock
- 01: System clock is selected as UART 5 clock
- 10: HSI clock is selected as UART 5 clock
- 11: LSE clock is selected as UART 5 clock

Bits7:6 **UART4SEL[1:0]**: UART 4 clock source selection

Set and reset by software.

- 00: APB1 clock (PLCLK1) is selected as UART 4 clock
- 01: System clock is selected as UART 4 clock
- 10: HSI clock is selected as UART 4 clock
- 11: LSE clock is selected as UART 4 clock

Bits5:4 **USART3SEL[1:0]**: USART 3 clock source selection

Set and reset by software.

- 00: APB1 clock (PCLK1) is selected as USART 3 clock
- 01: System clock is selected as USART 3 clock
- 10: HSI clock is selected as USART 3 clock
- 11: LSE clock is selected as USART 3 clock

Bits 3:2 **USART2SEL[1:0]**: USART 2 clock source selection

Set and reset by software.

- 00: APB1 clock (PCLK1) is selected as USART 2 clock
- 01: System clock is selected as USART 2 clock
- 10: HSI clock is selected as USART 2 clock
- 11: LSE clock is selected as USART 2 clock

Bits 1:0 **USART1SEL[1:0]**: USART 1 clock source selection

Set and reset by software.

- 00: APB2 clock (PCLK2) is selected as USART 1 clock
- 01: System clock is selected as USART 1 clock
- 10: HSI clock is selected as USART 1 clock
- 11: LSE clock is selected as USART 1 clock

### 5.3.27 RCC register map

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

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

	Addr. offset	Register name	
0x34	0x00	RCC_CR	
	0x04	RCC_PLLCFG_R	Res.
	0x08	RCC_CFGR	Res.
	0x0C	RCC_CIR	Res.
	0x10	RCC_AHB1RS_TR	Res.
	0x14	RCC_AHB2RS_TR	Res.
	0x18	RCC_AHB3RS_TR	Res.
	0x1C	Reserved	Res.
	0x20	RCC_APB1RS_TR	Res.
	0x24	RCC_APB2RS_TR	Res.
	0x28	Reserved	Res.
	0x2C	Reserved	Res.
0x30	0x00	RCC_AHB1ENR	Res.
0x34	0x00	RCC_AHB2ENR	Res.
	0x04	OTGHSULPEN	Res.
	0x08	OTGHSEN	Res.
	0x12	ETHMACPTPEN	Res.
	0x16	ETHMACRXEN	Res.
	0x20	ETHMACTXEN	Res.
	0x24	ETHMACEN	Res.
	0x28	DMA2DEN	Res.
	0x2C	DMA1EN	Res.
	0x30	DTCMRAMEN	Res.
	0x34	BKPSRAMEN	Res.
	0x38	TIM11RST	Res.
	0x42	TIM10RST	Res.
	0x46	TIM9RST	Res.
	0x50	SPDIFRXRST	Res.
	0x54	UART4RST	Res.
	0x58	UART3RST	Res.
	0x62	UART2RST	Res.
	0x66	SPI3RST	Res.
	0x70	SPI2RST	Res.
	0x74	SPI45RST	Res.
	0x78	CRCEN	Res.
	0x82	SDMMC1RST	Res.
	0x86	GPIOEN	Res.
	0x90	GPIOJEN	Res.
	0x94	OTGFSEN	Res.
	0x98	RNGEN	Res.
	0xA2	HASHEN	Res.
	0xA6	CRYPTEN	Res.
	0xB0	GPIOHEN	Res.
	0xB4	GPIOGEN	Res.
	0xB8	OTGFSRST	Res.
	0xC2	GPIOF1RST	Res.
	0xC6	GPIOF2RST	Res.
	0xD0	GPIOF3RST	Res.
	0xD4	GPIOF4RST	Res.
	0xD8	GPIOF5RST	Res.
	0xE2	GPIOF6RST	Res.
	0xE6	GPIOF7RST	Res.
	0xF0	GPIOF8RST	Res.
	0xF4	GPIOF9RST	Res.
	0xF8	GPIOF10RST	Res.
	0x04	USART1RST	Res.
	0x08	USART2RST	Res.
	0x0C	USART3RST	Res.
	0x10	USART4RST	Res.
	0x14	USART5RST	Res.
	0x18	USART6RST	Res.
	0x22	USART7RST	Res.
	0x26	USART8RST	Res.
	0x30	QSPIFRST	Res.
	0x34	DCMIRST	Res.
	0x38	DCMIRST	Res.
	0x42	DCMIRST	Res.
	0x46	DCMIRST	Res.
	0x50	DCMIRST	Res.
	0x54	DCMIRST	Res.
	0x58	DCMIRST	Res.
	0x62	DCMIRST	Res.
	0x66	DCMIRST	Res.
	0x70	DCMIRST	Res.
	0x74	DCMIRST	Res.
	0x78	DCMIRST	Res.
	0x82	DCMIRST	Res.
	0x86	DCMIRST	Res.
	0x90	DCMIRST	Res.
	0x94	DCMIRST	Res.
	0x98	DCMIRST	Res.
	0xA2	DCMIRST	Res.
	0xA6	DCMIRST	Res.
	0xB0	DCMIRST	Res.
	0xB4	DCMIRST	Res.
	0xB8	DCMIRST	Res.
	0xC2	DCMIRST	Res.
	0xC6	DCMIRST	Res.
	0xD0	DCMIRST	Res.
	0xD4	DCMIRST	Res.
	0xD8	DCMIRST	Res.
	0xE2	DCMIRST	Res.
	0xE6	DCMIRST	Res.
	0xF0	DCMIRST	Res.
	0xF4	DCMIRST	Res.
	0xF8	DCMIRST	Res.
	0x04	OTGHSRST	Res.
	0x08	PWRRST	Res.
	0x12	CECRST	Res.
	0x16	LTDRCRST	Res.
	0x20	CAN2RST	Res.
	0x24	CAN1RST	Res.
	0x28	I2C2RST	Res.
	0x32	I2C1RST	Res.
	0x36	SAI2RST	Res.
	0x40	I2C3RST	Res.
	0x44	SATRST	Res.
	0x48	DMA2RST	Res.
	0x52	DMA1RST	Res.
	0x56	DMA6RST	Res.
	0x60	DMA15RST	Res.
	0x64	PLL12SRDY	Res.
	0x68	PLL12SON	Res.
	0x72	PLLQ2PRE2	Res.
	0x76	MCO1IPRE1	Res.
	0x80	MCO1IPRE0	Res.
	0x84	MCO2IPRE1	Res.
	0x88	MCO2IPRE0	Res.
	0x92	MCO3IPRE1	Res.
	0x96	MCO3IPRE0	Res.
	0xA0	MCO4IPRE1	Res.
	0xA4	MCO4IPRE0	Res.
	0xB0	MCO5IPRE1	Res.
	0xB4	MCO5IPRE0	Res.
	0xC0	MCO6IPRE1	Res.
	0xC4	MCO6IPRE0	Res.
	0xD0	MCO7IPRE1	Res.
	0xD4	MCO7IPRE0	Res.
	0xE0	MCO8IPRE1	Res.
	0xE4	MCO8IPRE0	Res.
	0xF0	MCO9IPRE1	Res.
	0xF4	MCO9IPRE0	Res.
	0x04	ETHMACRST	Res.
	0x08	DMA2DRST	Res.
	0x12	DMA2DRST	Res.
	0x16	DMA2RST	Res.
	0x20	DMA1RST	Res.
	0x24	PLL12SRDY	Res.
	0x28	PLL12SON	Res.
	0x32	PLLQ2PRE4	Res.
	0x36	RTCPRE4	Res.
	0x40	RTCPRE3	Res.
	0x44	RTCPRE2	Res.
	0x48	RTCPRE1	Res.
	0x52	RTCPRE0	Res.
	0x56	HSERDYC	Res.
	0x60	HSERDYC	Res.
	0x64	HSERDYC	Res.
	0x68	HSERDYC	Res.
	0x72	HSERDYC	Res.
	0x76	HSERDYC	Res.
	0x80	HSERDYC	Res.
	0x84	HSERDYC	Res.
	0x88	HSERDYC	Res.
	0x92	HSERDYC	Res.
	0x96	HSERDYC	Res.
	0xA0	HSERDYC	Res.
	0xA4	HSERDYC	Res.
	0xB0	HSERDYC	Res.
	0xB4	HSERDYC	Res.
	0xC0	HSERDYC	Res.
	0xC4	HSERDYC	Res.
	0xD0	HSERDYC	Res.
	0xD4	HSERDYC	Res.
	0xE0	HSERDYC	Res.
	0xE4	HSERDYC	Res.
	0xF0	HSERDYC	Res.
	0xF4	HSERDYC	Res.
	0x04	PLL12SRDY	Res.
	0x08	PLL12SON	Res.
	0x12	PLLQ2PRE2	Res.
	0x16	MCO1IPRE1	Res.
	0x20	MCO1IPRE0	Res.
	0x24	MCO2IPRE1	Res.
	0x28	MCO2IPRE0	Res.
	0x32	MCO3IPRE1	Res.
	0x36	MCO3IPRE0	Res.
	0x40	MCO4IPRE1	Res.
	0x44	MCO4IPRE0	Res.
	0x48	MCO5IPRE1	Res.
	0x52	MCO5IPRE0	Res.
	0x56	MCO6IPRE1	Res.
	0x60	MCO6IPRE0	Res.
	0x64	MCO7IPRE1	Res.
	0x68	MCO7IPRE0	Res.
	0x72	MCO8IPRE1	Res.
	0x76	MCO8IPRE0	Res.
	0x80	MCO9IPRE1	Res.
	0x84	MCO9IPRE0	Res.
	0x88	MCO10IPRE1	Res.
	0x92	MCO10IPRE0	Res.
	0x96	MCO11IPRE1	Res.
	0xA0	MCO11IPRE0	Res.
	0xA4	MCO12IPRE1	Res.
	0xB0	MCO12IPRE0	Res.
	0xB4	MCO13IPRE1	Res.
	0xC0	MCO13IPRE0	Res.
	0xC4	MCO14IPRE1	Res.
	0xD0	MCO14IPRE0	Res.
	0xD4	MCO15IPRE1	Res.
	0xE0	MCO15IPRE0	Res.
	0xE4	MCO16IPRE1	Res.
	0xF0	MCO16IPRE0	Res.
	0xF4	MCO17IPRE1	Res.
	0x04	HSI0RDY	Res.
	0x08	HSI1RDY	Res.
	0x12	HSI2RDY	Res.
	0x16	HSI3RDY	Res.
	0x20	HSI4RDY	Res.
	0x24	HSI5RDY	Res.
	0x28	HSI6RDY	Res.
	0x32	HSI7RDY	Res.
	0x36	HSI8RDY	Res.
	0x40	HSI9RDY	Res.
	0x44	HSI10RDY	Res.
	0x48	HSI11RDY	Res.
	0x52	HSI12RDY	Res.
	0x56	HSI13RDY	Res.
	0x60	HSI14RDY	Res.
	0x64	HSI15RDY	Res.
	0x68	HSI16RDY	Res.
	0x72	HSI17RDY	Res.
	0x76	HSI18RDY	Res.
	0x80	HSI19RDY	Res.
	0x84	HSI20RDY	Res.
	0x88	HSI21RDY	Res.
	0x92	HSI22RDY	Res.
	0x96	HSI23RDY	Res.
	0xA0	HSI24RDY	Res.
	0xA4	HSI25RDY	Res.
	0xB0	HSI26RDY	Res.
	0xB4	HSI27RDY	Res.
	0xC0	HSI28RDY	Res.
	0xC4	HSI29RDY	Res.
	0xD0	HSI30RDY	Res.
	0xD4	HSI31RDY	Res.
	0xE0	HSI32RDY	Res.
	0xE4	HSI33RDY	Res.
	0xF0	HSI34RDY	Res.
	0xF4	HSI35RDY	Res.
	0x04	HSI0ARST	Res.
	0x08	HSI1ARST	Res.
	0x12	HSI2ARST	Res.
	0x16	HSI3ARST	Res.
	0x20	HSI4ARST	Res.
	0x24	HSI5ARST	Res.
	0x28	HSI6ARST	Res.
	0x32	HSI7ARST	Res.
	0x36	HSI8ARST	Res.
	0x40	HSI9ARST	Res.
	0x44	HSI10ARST	Res.
	0x48	HSI11ARST	Res.
	0x52	HSI12ARST	Res.
	0x56	HSI13ARST	Res.
	0x60	HSI14ARST	Res.
	0x64	HSI15ARST	Res.
	0x68	HSI16ARST	Res.
	0x72	HSI17ARST	Res.
	0x76	HSI18ARST	Res.
	0x80	HSI19ARST	Res.
	0x84	HSI20ARST	Res.
	0x88	HSI21ARST	Res.
	0x92	HSI22ARST	Res.
	0x96	HSI23ARST	Res.
	0xA0	HSI24ARST	Res.
	0xA4	HSI25ARST	Res.
	0xB0	HSI26ARST	Res.
	0xB4	HSI27ARST	Res.
	0xC0	HSI28ARST	Res.
	0xC4	HSI29ARST	Res.
	0xD0	HSI30ARST	Res.
	0xD4	HSI31ARST	Res.
	0xE0	HSI32ARST	Res.
	0xE4	HSI33ARST	Res.
	0xF0	HSI34ARST	Res.
	0xF4	HSI35ARST	Res.
	0x04	CRYPRST	Res.
	0x08	TIM6RST	Res.
	0x12	TIM5RST	Res.
	0x16	TIM4RST	Res.
	0x20	TIM3RST	Res.
	0x24	TIM2RST	Res.
	0x28	TIM1RST	Res.
	0x32	USART1RST	Res.
	0x36	USART0RST	Res.
	0x40	GPIOE0RST	Res.
	0x44	GPIOE1RST	Res.
	0x48	GPIOE2RST	Res.
	0x52	GPIOE3RST	Res.
	0x56	GPIOE4RST	Res.
	0x60	GPIOE5RST	Res.
	0x64	GPIOE6RST	Res.
	0x68	GPIOE7RST	Res.
	0x72	GPIOE8RST	Res.
	0x76	GPIOE9RST	Res.
	0x80	GPIOE10RST	Res.
	0x84	GPIOE11RST	Res.
	0x88	GPIOE12RST	Res.
	0x92	GPIOE13RST	Res.
	0x96	GPIOE14RST	Res.
	0xA0	GPIOE15RST	Res.
	0xA4	GPIOE16RST	Res.
	0xB0	GPIOE17RST	Res.
	0xB4	GPIOE18RST	Res.
	0xC0	GPIOE19RST	Res.
	0xC4	GPIOE20RST	Res.
	0xD0	GPIOE21RST	Res.
	0xD4	GPIOE22RST	Res.
	0xE0	GPIOE23RST	Res.
	0xE4	GPIOE24RST	Res.
	0xF0	GPIOE25RST	Res.
	0xF4	GPIOE26RST	Res.
	0x04	GPIOF0RST	Res.
	0x08	GPIOF1RST	Res.
	0x12	GPIOF2RST	Res.
	0x16	GPIOF3RST	Res.
	0x20	GPIOF4RST	Res.
	0x24	GPIOF5RST	Res.
	0x28	GPIOF6RST	Res.
	0x32	GPIOF7RST	Res.
	0x36	GPIOF8RST	Res.
	0x40	GPIOF9RST	Res.
	0x44	GPIOF10RST	Res.
	0x48	GPIOF11RST	Res.
	0x52	GPIOF12RST	Res.
	0x56	GPIOF13RST	Res.
	0x60	GPIOF14RST	Res.
	0x64	GPIOF15RST	Res.
	0x68</td		

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

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

Addr. offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x80	RCC_SSCGR	SSCGEN	Res.	SPREADSEL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
0x84	RCC_PLLI2SC_FGR	PLLII2SR[2:0]	PLLII2SQ[3:0]	PLLII2SQ[4:0]	PLLSAIQ[4:0]	PLLSAIR[2:0]	SDMMC1SEL	CK48MSEL	CECSEL	LPTIM1SEL	TIMPRE	SAI2SEL[1:0]	SAI1SEL[1:0]	I2C3SEL	I2C4SEL	I2C2SEL	I2C1SEL	PLLSAIDIVR1[1:0]  PLLSAIP[1:0]  PLLII2SP[1:0]	PLLSAIDIVQ[4:0]	PLLSAIN[8:0]	PLLI2SN[8:0]	PLLI2SDIVQ[4:0]	UART8SEL	UART7SEL	UART6SEL	UART5SEL	UART4SEL	UART3SEL	UART2SEL	UART1SEL				
0x88	RCC_PLLSAI_CFGR	PLLII2SR[2:0]	PLLII2SQ[3:0]	PLLII2SQ[4:0]	PLLSAIQ[4:0]	PLLSAIR[2:0]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
0x8C	RCC_DCKCF_GR1	SDMMC1SEL	CK48MSEL	CECSEL	LPTIM1SEL	TIMPRE	SAI2SEL[1:0]	SAI1SEL[1:0]	I2C3SEL	I2C4SEL	I2C2SEL	I2C1SEL	PLLSAIDIVR1[1:0]  PLLSAIP[1:0]  PLLII2SP[1:0]	PLLSAIDIVQ[4:0]	PLLSAIN[8:0]	PLLI2SN[8:0]	PLLI2SDIVQ[4:0]	UART8SEL	UART7SEL	UART6SEL	UART5SEL	UART4SEL	UART3SEL	UART2SEL	UART1SEL									
0x90	RCC_DCKCF_GR2	CK48MSEL	CECSEL	LPTIM1SEL	SAI2SEL[1:0]	SAI1SEL[1:0]	I2C3SEL	I2C4SEL	I2C2SEL	I2C1SEL	PLLSAIDIVR1[1:0]  PLLSAIP[1:0]  PLLII2SP[1:0]	PLLSAIDIVQ[4:0]	PLLSAIN[8:0]	PLLI2SN[8:0]	PLLI2SDIVQ[4:0]	UART8SEL	UART7SEL	UART6SEL	UART5SEL	UART4SEL	UART3SEL	UART2SEL	UART1SEL											

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

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

### 6.1 Introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIO<sub>x</sub>\_MODER, GPIO<sub>x</sub>\_OTYPER, GPIO<sub>x</sub>\_OSPEEDR and GPIO<sub>x</sub>\_PUPDR), two 32-bit data registers (GPIO<sub>x</sub>\_IDR and GPIO<sub>x</sub>\_ODR) and a 32-bit set/reset register (GPIO<sub>x</sub>\_BSRR). In addition all GPIOs have a 32-bit locking register (GPIO<sub>x</sub>\_LCKR) and two 32-bit alternate function selection registers (GPIO<sub>x</sub>\_AFRH and GPIO<sub>x</sub>\_AFRL).

### 6.2 GPIO main features

- Output states: push-pull or open drain + pull-up/down
- Output data from output data register (GPIO<sub>x</sub>\_ODR) or peripheral (alternate function output)
- Speed selection for each I/O
- Input states: floating, pull-up/down, analog
- Input data to input data register (GPIO<sub>x</sub>\_IDR) or peripheral (alternate function input)
- Bit set and reset register (GPIO<sub>x</sub>\_BSRR) for bitwise write access to GPIO<sub>x</sub>\_ODR
- Locking mechanism (GPIO<sub>x</sub>\_LCKR) provided to freeze the I/O port configurations
- Analog function
- Alternate function selection registers
- Fast toggle capable of changing every two clock cycles
- Highly flexible pin multiplexing allows the use of I/O pins as GPIOs or as one of several peripheral functions

### 6.3 GPIO functional description

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

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

*Figure 17* and *Figure 18* shows the basic structures of a standard and a 5-Volt tolerant I/O port bit, respectively. *Table 22* gives the possible port bit configurations.

Figure 17. Basic structure of an I/O port bit

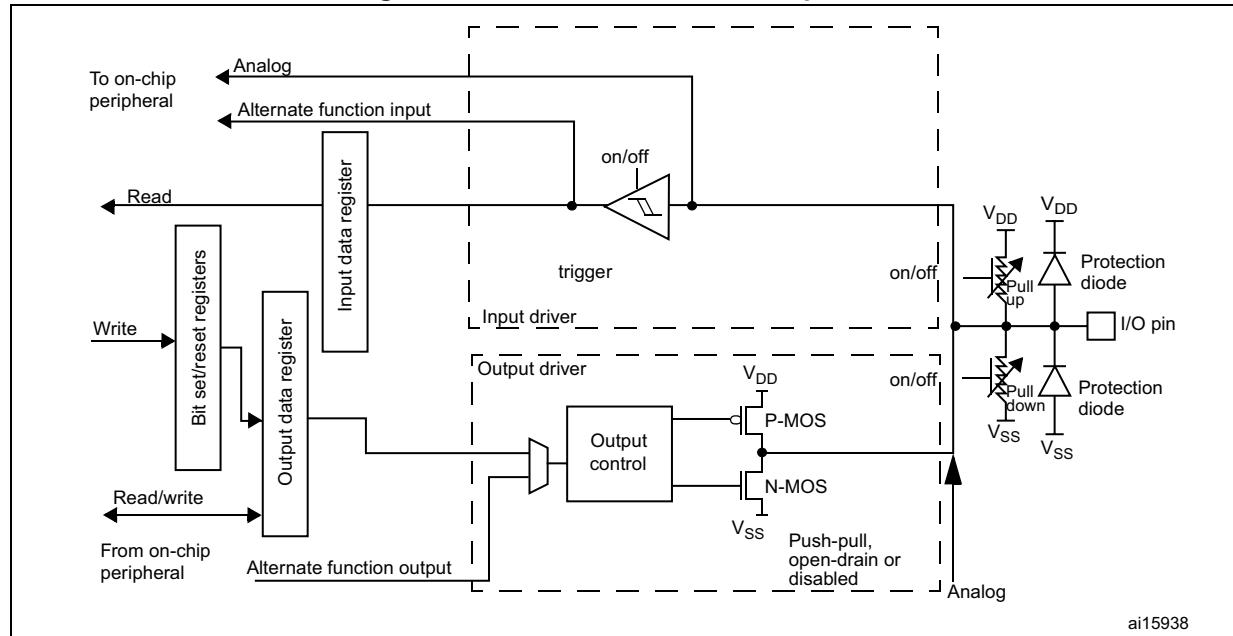
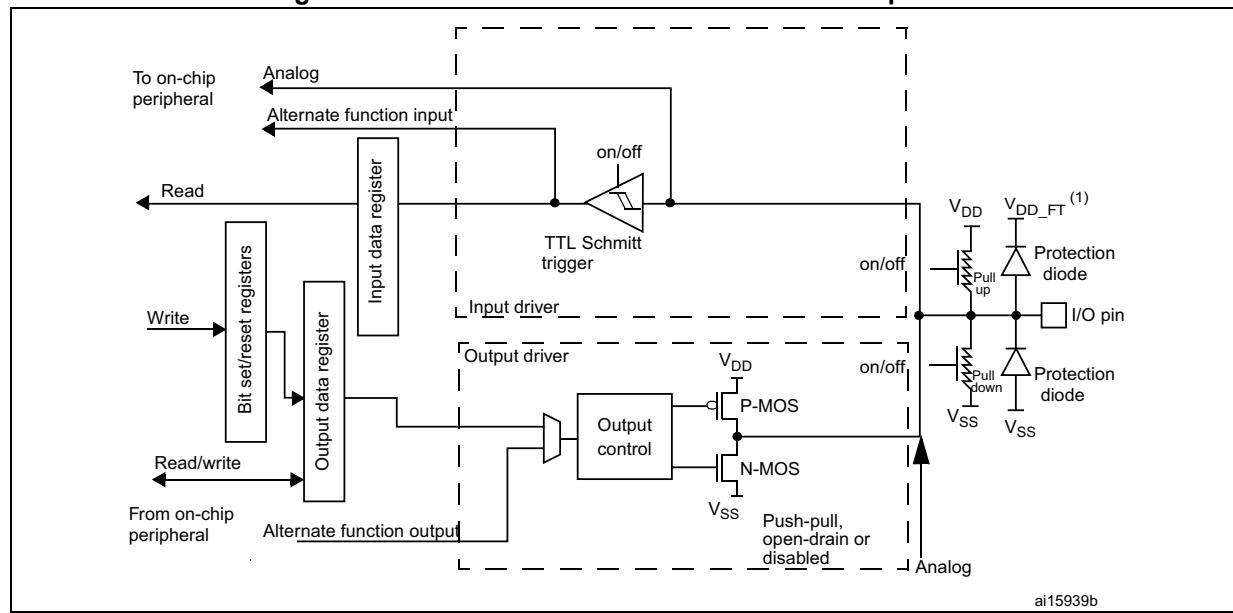


Figure 18. Basic structure of a 5-Volt tolerant I/O port bit



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

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

MODE(i) [1:0]	OTYPER(i)	OSPEED(i) [1:0]	PUPD(i) [1:0]		I/O configuration	
01	0	SPEED [1:0]	0	0	GP output	PP
	0		0	1	GP output	PP + PU
	0		1	0	GP output	PP + PD
	0		1	1	Reserved	
	1		0	0	GP output	OD
	1		0	1	GP output	OD + PU
	1		1	0	GP output	OD + PD
	1		1	1	Reserved (GP output OD)	
10	0	SPEED [1:0]	0	0	AF	PP
	0		0	1	AF	PP + PU
	0		1	0	AF	PP + PD
	0		1	1	Reserved	
	1		0	0	AF	OD
	1		0	1	AF	OD + PU
	1		1	0	AF	OD + PD
	1		1	1	Reserved	
00	x	x	x	0	0	Input
	x	x	x	0	1	Input
	x	x	x	1	0	Input
	x	x	x	1	1	Reserved (input floating)
11	x	x	x	0	0	Input/output
	x	x	x	0	1	Reserved
	x	x	x	1	0	
	x	x	x	1	1	

1. GP = general-purpose, PP = push-pull, PU = pull-up, PD = pull-down, OD = open-drain, AF = alternate function.

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

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

The debug pins are in AF pull-up/pull-down after reset:

- PA15: JTDI in pull-up
- PA14: JTCK/SWCLK in pull-down
- PA13: JTMS/SWDAT in pull-up
- PB4: NJTRST in pull-up
- PB3: JTDO in floating state

When the pin is configured as output, the value written to the output data register (GPIOx\_ODR) is output on the I/O pin. It is possible to use the output driver in push-pull mode or open-drain mode (only the low level is driven, high level is HI-Z).

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

All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not depending on the value in the GPIOx\_PUPDR register.

### 6.3.2 I/O pin alternate function multiplexer and mapping

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

Each I/O pin has a multiplexer with up to sixteen alternate function inputs (AF0 to AF15) that can be configured through the GPIOx\_AFRL (for pin 0 to 7) and GPIOx\_AFRH (for pin 8 to 15) registers:

- After reset the multiplexer selection is alternate function 0 (AF0). The I/Os are configured in alternate function mode through GPIOx\_MODER register.
- The specific alternate function assignments for each pin are detailed in the device datasheet.
- Cortex-M7 with FPU EVENTOUT is mapped on AF15

In addition to this flexible I/O multiplexing architecture, each peripheral has alternate functions mapped onto different I/O pins to optimize the number of peripherals available in smaller packages.

To use an I/O in a given configuration, the user has to proceed as follows:

- **Debug function:** after each device reset these pins are assigned as alternate function pins immediately usable by the debugger host
- **System function:** MCOx pins have to be configured in alternate function mode.
- **GPIO:** configure the desired I/O as output, input or analog in the GPIOx\_MODER register.
- **Peripheral alternate function:**
  - Connect the I/O to the desired AFx in one of the GPIOx\_AFRL or GPIOx\_AFRH register.
  - Select the type, pull-up/pull-down and output speed via the GPIOx\_OTYPER, GPIOx\_PUPDR and GPIOx\_OSPEEDER registers, respectively.

- Configure the desired I/O as an alternate function in the GPIOx\_MODER register.
- **Additional functions:**
  - For the ADC and DAC, configure the desired I/O in analog mode in the GPIOx\_MODER register and configure the required function in the ADC and DAC registers.
  - For the additional functions like RTC\_OUT, RTC\_TS, RTC\_TAMPx, WKUPx and oscillators, configure the required function in the related RTC, PWR and RCC registers. These functions have priority over the configuration in the standard GPIO registers. For details about I/O control by the RTC, refer to [Section 29.3: RTC functional description on page 904](#).
- **EVENTOUT**
  - Configure the I/O pin used to output the core EVENTOUT signal by connecting it to AF15.

Refer to the “Alternate function mapping” table in the device datasheet for the detailed mapping of the alternate function I/O pins.

### 6.3.3 I/O port control registers

Each of the GPIO ports has four 32-bit memory-mapped control registers (GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR, GPIOx\_PUPDR) to configure up to 16 I/Os. The GPIOx\_MODER register is used to select the I/O mode (input, output, AF, analog). The GPIOx\_OTYPER and GPIOx\_OSPEEDR registers are used to select the output type (push-pull or open-drain) and speed. The GPIOx\_PUPDR register is used to select the pull-up/pull-down whatever the I/O direction.

### 6.3.4 I/O port data registers

Each GPIO has two 16-bit memory-mapped data registers: input and output data registers (GPIOx\_IDR and GPIOx\_ODR). GPIOx\_ODR stores the data to be output, it is read/write accessible. The data input through the I/O are stored into the input data register (GPIOx\_IDR), a read-only register.

See [Section 6.4.5: GPIO port input data register \(GPIOx\\_IDR\) \(x = A to K\)](#) and [Section 6.4.6: GPIO port output data register \(GPIOx\\_ODR\) \(x = A to K\)](#) for the register descriptions.

### 6.3.5 I/O data bitwise handling

The bit set reset register (GPIOx\_BSRR) is a 32-bit register which allows the application to set and reset each individual bit in the output data register (GPIOx\_ODR). The bit set reset register has twice the size of GPIOx\_ODR.

To each bit in GPIOx\_ODR, correspond two control bits in GPIOx\_BSRR: BS(i) and BR(i). When written to 1, bit BS(i) **sets** the corresponding ODR(i) bit. When written to 1, bit BR(i) **resets** the ODR(i) corresponding bit.

Writing any bit to 0 in GPIOx\_BSRR does not have any effect on the corresponding bit in GPIOx\_ODR. If there is an attempt to both set and reset a bit in GPIOx\_BSRR, the set action takes priority.

Using the GPIOx\_BSRR register to change the values of individual bits in GPIOx\_ODR is a “one-shot” effect that does not lock the GPIOx\_ODR bits. The GPIOx\_ODR bits can always

be accessed directly. The GPIOx\_BSRR register provides a way of performing atomic bitwise handling.

There is no need for the software to disable interrupts when programming the GPIOx\_ODR at bit level: it is possible to modify one or more bits in a single atomic AHB write access.

### 6.3.6 GPIO locking mechanism

It is possible to freeze the GPIO control registers by applying a specific write sequence to the GPIOx\_LCKR register. The frozen registers are GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR, GPIOx\_PUPDR, GPIOx\_AFRL and GPIOx\_AFRH.

To write the GPIOx\_LCKR register, a specific write / read sequence has to be applied. When the right LOCK sequence is applied to bit 16 in this register, the value of LCKR[15:0] is used to lock the configuration of the I/Os (during the write sequence the LCKR[15:0] value must be the same). When the LOCK sequence has been applied to a port bit, the value of the port bit can no longer be modified until the next MCU reset or peripheral reset. Each GPIOx\_LCKR bit freezes the corresponding bit in the control registers (GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR, GPIOx\_PUPDR, GPIOx\_AFRL and GPIOx\_AFRH).

The LOCK sequence (refer to [Section 6.4.8: GPIO port configuration lock register \(GPIOx\\_LCKR\) \(x = A to K\)](#)) can only be performed using a word (32-bit long) access to the GPIOx\_LCKR register due to the fact that GPIOx\_LCKR bit 16 has to be set at the same time as the [15:0] bits.

For more details refer to LCKR register description in [Section 6.4.8: GPIO port configuration lock register \(GPIOx\\_LCKR\) \(x = A to K\)](#).

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

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

This means that a number of possible peripheral functions are multiplexed on each GPIO using the GPIOx\_AFRL and GPIOx\_AFRH alternate function registers. The application can thus select any one of the possible functions for each I/O. The AF selection signal being common to the alternate function input and alternate function output, a single channel is selected for the alternate function input/output of a given I/O.

To know which functions are multiplexed on each GPIO pin refer to the device datasheet.

### 6.3.8 External interrupt/wake-up lines

All ports have external interrupt capability. To use external interrupt lines, the port must be configured in input mode.

Refer to [Section 11: Extended interrupts and events controller \(EXTI\)](#) and to [Section 11.3: Wakeup event management](#).

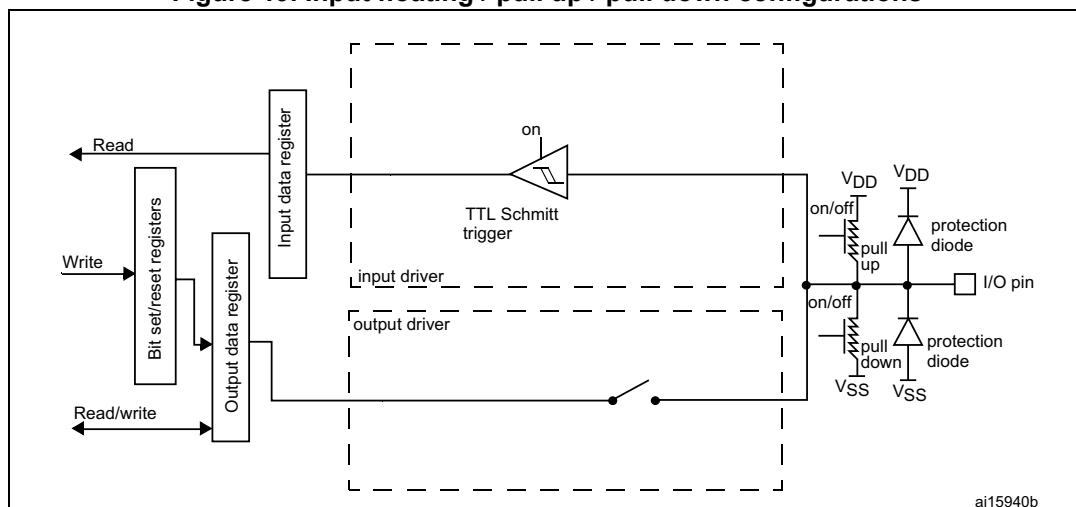
### 6.3.9 Input configuration

When the I/O port is programmed as input:

- The output buffer is disabled
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx\_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register provides the I/O state

*Figure 19* shows the input configuration of the I/O port bit.

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



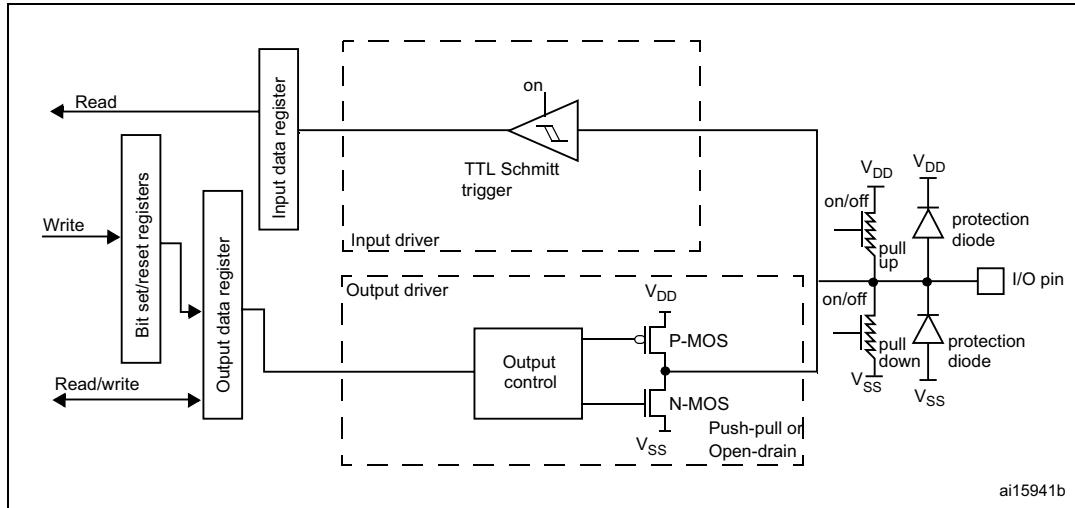
### 6.3.10 Output configuration

When the I/O port is programmed as output:

- The output buffer is enabled:
  - Open drain mode: a “0” in the output register activates the N-MOS whereas a “1” in the output register leaves the port in Hi-Z (the P-MOS is never activated)
  - Push-pull mode: a “0” in the output register activates the N-MOS whereas a “1” in the output register activates the P-MOS
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx\_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state
- A read access to the output data register gets the last written value

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

**Figure 20. Output configuration**



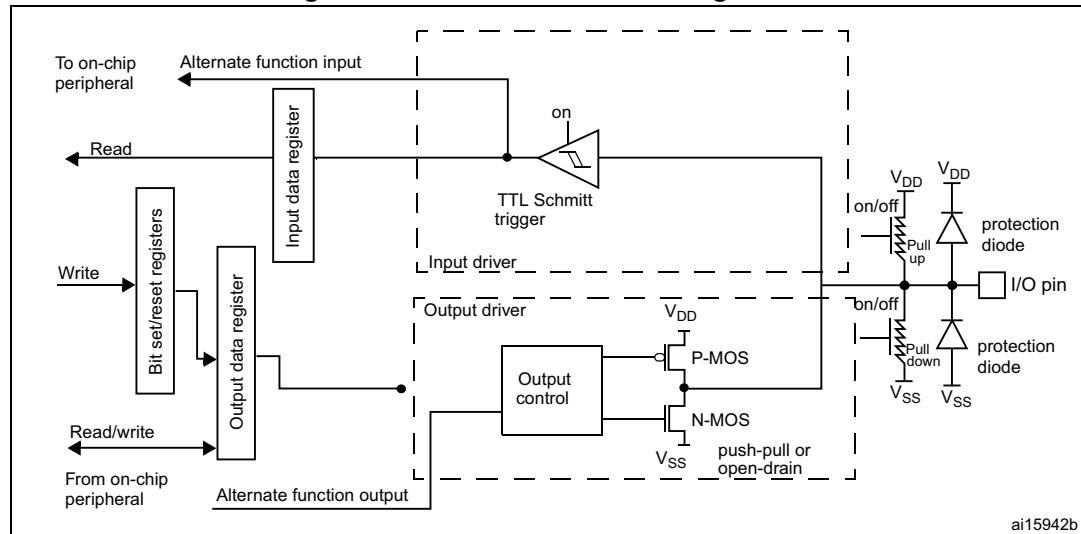
### 6.3.11 Alternate function configuration

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

- The output buffer can be configured in open-drain or push-pull mode
- The output buffer is driven by the signals coming from the peripheral (transmitter enable and data)
- The Schmitt trigger input is activated
- The weak pull-up and pull-down resistors are activated or not depending on the value in the **GPIOx\_PUPDR** register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state

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

**Figure 21. Alternate function configuration**



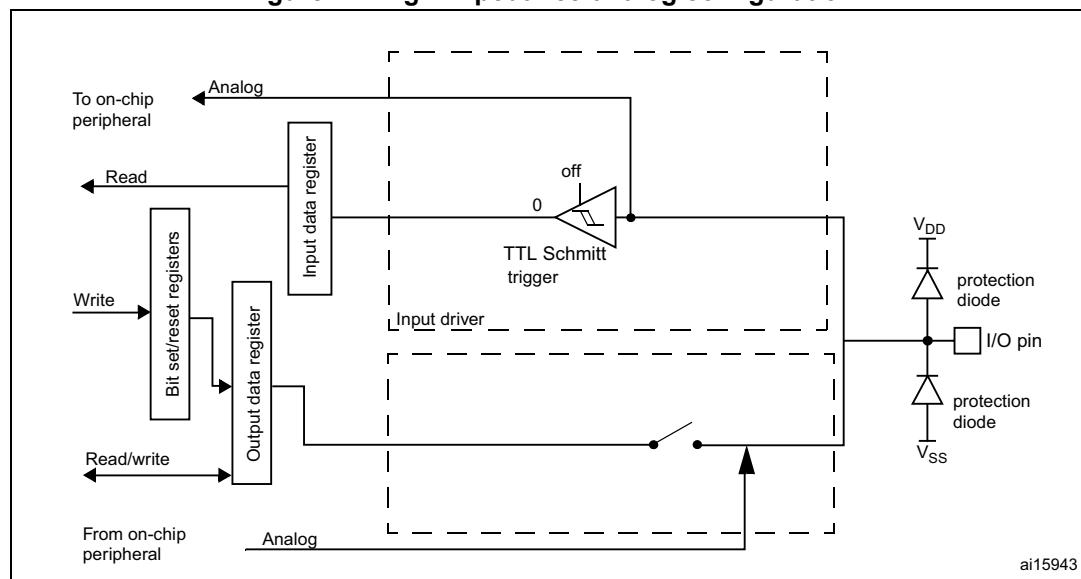
### 6.3.12 Analog configuration

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

- The output buffer is disabled
- The Schmitt trigger input is deactivated, providing zero consumption for every analog value of the I/O pin. The output of the Schmitt trigger is forced to a constant value (0).
- The weak pull-up and pull-down resistors are disabled by hardware
- Read access to the input data register gets the value "0"

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

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



### 6.3.13 Using the HSE or LSE oscillator pins as GPIOs

When the HSE or LSE oscillator is switched OFF (default state after reset), the related oscillator pins can be used as normal GPIOs.

When the HSE or LSE oscillator is switched ON (by setting the HSEON or LSEON bit in the RCC\_CSR register) the oscillator takes control of its associated pins and the GPIO configuration of these pins has no effect.

When the oscillator is configured in a user external clock mode, only the OSC\_IN or OSC32\_IN pin is reserved for clock input and the OSC\_OUT or OSC32\_OUT pin can still be used as normal GPIO.

### 6.3.14 Using the GPIO pins in the backup supply domain

The PC13/PC14/PC15/PI8 GPIO functionality is lost when the core supply domain is powered off (when the device enters Standby mode). In this case, if their GPIO configuration is not bypassed by the RTC configuration, these pins are set in an analog input mode.

## 6.4 GPIO registers

For a summary of register bits, register address offsets and reset values, refer to [Table 23](#).

The peripheral registers can be written in word, half word or byte mode.

### 6.4.1 GPIO port mode register (GPIOx\_MODER) (x = A to K)

Address offset: 0x00

Reset value: 0xA800 0000 for port A

Reset value: 0x0000 0280 for port B

Reset value: 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw										

Bits 31:0 **MODER[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

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

00: Input mode (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

### 6.4.2 GPIO port output type register (GPIOx\_OTYPER) (x = A to K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw															

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

Bits 15:0 **OT[15:0]**: Port x configuration I/O pin y (y = 15 to 0)

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

0: Output push-pull (reset state)

1: Output open-drain

### 6.4.3 GPIO port output speed register (GPIOx\_OSPEEDR) (x = A to K)

Address offset: 0x08

Reset value: 0x0C00 0000 (for port A)

Reset value: 0x0000 00C0 (for port B)

Reset value: 0x0000 0000 (for other ports)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw										

Bits 31:0 **OSPEEDR[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

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

00: Low speed

01: Medium speed

10: High speed

11: Very high speed

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

### 6.4.4 GPIO port pull-up/pull-down register (GPIOx\_PUPDR) (x = A to K)

Address offset: 0x0C

Reset value: 0x6400 0000 (for port A)

Reset value: 0x0000 0100 (for port B)

Reset value: 0x0000 0000 (for other ports)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw										

Bits 31:0 **PUPDR[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

### 6.4.5 GPIO port input data register (GPIO<sub>x</sub>\_IDR) (x = A to K)

Address offset: 0x10

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **IDR[15:0]**: Port x input data I/O pin y (y = 15 to 0)

These bits are read-only. They contain the input value of the corresponding I/O port.

### 6.4.6 GPIO port output data register (GPIO<sub>x</sub>\_ODR) (x = A to K)

Address offset: 0x14

Reset value: 0x0000 0000

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

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

Bits 15:0 **ODR[15:0]**: Port output data I/O pin y (y = 15 to 0)

These bits can be read and written by software.

### 6.4.7 GPIO port bit set/reset register (GPIO<sub>x</sub>\_BSRR) (x = A to K)

Address offset: 0x18

Reset value: 0x0000 0000

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

Bits 31:16 **BR[15:0]**: Port x reset I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

*Note: If both BS<sub>x</sub> and BR<sub>x</sub> are set, BS<sub>x</sub> has priority.*

Bits 15:0 **BS[15:0]**: Port x set I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

#### 6.4.8 GPIO port configuration lock register (GPIO<sub>x</sub>\_LCKR) (x = A to K)

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

*Note: A specific write sequence is used to write to the GPIO<sub>x</sub>\_LCKR register. Only word access (32-bit long) is allowed during this locking sequence.*

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

Address offset: 0x1C

Reset value: 0x0000 0000

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

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

#### Bit 16 **LCKK:** Lock key

This bit can be read any time. It can only be modified using the lock key write sequence.

0: Port configuration lock key not active

1: Port configuration lock key active. The GPIOx\_LCKR register is locked until the next MCU reset or peripheral reset.

LOCK key write sequence:

WR LCKR[16] = 1 + LCKR[15:0]

WR LCKR[16] = 0 + LCKR[15:0]

WR LCKR[16] = 1 + LCKR[15:0]

RD LCKR

RD LCKR[16] = 1 (this read operation is optional but it confirms that the lock is active)

*Note: During the LOCK key write sequence, the value of LCK[15:0] must not change.*

*Any error in the lock sequence aborts the lock.*

*After the first lock sequence on any bit of the port, any read access on the LCKK bit returns 1 until the next MCU reset or peripheral reset.*

#### Bits 15:0 **LCK[15:0]:** Port x lock I/O pin y (y = 15 to 0)

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

0: Port configuration not locked

1: Port configuration locked

### 6.4.9 GPIO alternate function low register (GPIOx\_AFRL) (x = A to K)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR7[3:0]				AFR6[3:0]				AFR5[3:0]				AFR4[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR3[3:0]				AFR2[3:0]				AFR1[3:0]				AFR0[3:0]			
rw	rw	rw	rw												

Bits 31:0 **AFR[7:0][3:0]**: Alternate function selection for port x I/O pin y (y = 7 to 0)

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

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

#### 6.4.10 GPIO alternate function high register (GPIOx\_AFRH) (x = A to J)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR15[3:0]				AFR14[3:0]				AFR13[3:0]				AFR12[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR11[3:0]				AFR10[3:0]				AFR9[3:0]				AFR8[3:0]			
rw	rw	rw	rw												

Bits 31:0 **AFR[15:8][3:0]**: Alternate function selection for port x I/O pin y (y = 15 to 8)

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

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



### 6.4.11 GPIO register map

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

**Table 23. GPIO register map and reset values**

Offset	Register name	Reset value	31
0x00	GPIOA_MODER	0 MODER15[1:0] 0 MODER15[1:0]	30
0x00	GPIOB_MODER	0 MODER14[1:0] 0 MODER14[1:0]	29
0x00	GPIOx_MODER (where x = C..K)	0 MODER13[1:0] 0 MODER13[1:0]	28
0x04	GPIOx_OTYPER (where x = A to K)	0 MODER12[1:0] 0 MODER12[1:0]	27
0x08	GPIOA_OSPEEDR	0 MODER11[1:0] 0 MODER11[1:0]	26
0x08	GPIOB_OSPEEDR	0 MODER10[1:0] 0 MODER10[1:0]	25
0x08	GPIOx_OSPEEDR (where x = C..K)	0 MODER9[1:0] 0 MODER9[1:0]	24
0x0C	GPIOA_PUPDR	0 MODER8[1:0] 0 MODER8[1:0]	23
0x0C	GPIOx_PUPDR	0 MODER7[1:0] 0 MODER7[1:0]	22
0x0C	GPIOx_PUPDR	0 MODER6[1:0] 0 MODER6[1:0]	21
0x0C	GPIOx_PUPDR	0 MODER5[1:0] 0 MODER5[1:0]	20
0x0C	GPIOx_PUPDR	0 MODER4[1:0] 0 MODER4[1:0]	19
0x0C	GPIOx_PUPDR	0 MODER3[1:0] 0 MODER3[1:0]	18
0x0C	GPIOx_PUPDR	0 MODER2[1:0] 0 MODER2[1:0]	17
0x0C	GPIOx_PUPDR	0 MODER1[1:0] 0 MODER1[1:0]	16
0x0C	GPIOx_PUPDR	0 MODER0[1:0] 0 MODER0[1:0]	15

**Table 23. GPIO register map and reset values (continued)**

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 7 System configuration controller (SYSCFG)

The system configuration controller is mainly used to:

- Remap the memory areas
- Select the Ethernet PHY interface
- Manage the external interrupt line connection to the GPIOs.

### 7.1 I/O compensation cell

By default the I/O compensation cell is not used. However when the I/O output buffer speed is configured in 50 MHz or 100 MHz mode, it is recommended to use the compensation cell for slew rate control on I/O  $t_{f(FOUT)}/t_{r(FOUT)}$  commutation to reduce the I/O noise on power supply.

When the compensation cell is enabled, a READY flag is set to indicate that the compensation cell is ready and can be used. The I/O compensation cell can be used only when the supply voltage ranges from 2.4 to 3.6 V.

### 7.2 SYSCFG registers

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

This register is used for specific configurations on memory mapping:

- 1bit is used to indicate which option bytes BOOT\_ADD0 or BOOT\_ADD1 defines the boot memory base address.
- Other bits are used to swap FMC SDRAM Banks with FMC NOR/PSRAM bank

Address offset: 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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	SWP_FMC[1:0]	Res.	MEM_BOOT									
				rw	rw										r

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

Bits 11:10 **SWP\_FMC[1:0]**: FMC memory mapping swap

Set and cleared by software. These bits are used to swap the FMC SDRAM banks and FMC NOR/PSRAM in order to enable the code execution from SDRAM Banks without modifying the default MPU attribute

00: No FMC memory mapping swapping

SDRAM bank1 and Bank2 are mapped at 0xC000 0000 and 0xD000 0000 respectively (default mapping)

NOR/RAM is accessible @ 0x60000000 (default mapping)

01: NOR/RAM and SDRAM memory mapping swapped,

SDRAM bank1 and bank2 are mapped at 0x6000 0000 and 0x7000 0000, respectively

NOR/PSRAM bank is mapped at 0xC000 0000

10: Reserved

11: Reserved

Bits 9:1 Reserved, must be kept at reset value.

Bit 0 **MEM\_BOOT**: Memory boot mapping

This bit indicates which option bytes BOOT\_ADD0 or BOOT\_ADD1 defines the boot memory base address.

0: Boot memory base address is defined by BOOT\_ADD0 option byte  
(Factory Reset value: TCM-FLASH mapped at 0x00200000).

1: Boot memory base address is defined by BOOT\_ADD1 option byte  
(Factory Reset value: System memory mapped at 0x001 0000).

*Note: Refer to section 2.3: Memory map for details about the boot memory base address selection.*

## 7.2.2 SYSCFG peripheral mode configuration register (SYSCFG\_PMC)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	MII_RMII_SEL	Res.	Res.	Res.	Res.	Res.	ADCxDC2								
								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.								

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

Bit 23 **MII\_RMII\_SEL**: Ethernet PHY interface selection

Set and Cleared by software. These bits control the PHY interface for the Ethernet MAC.

0: MII interface is selected

1: RMII PHY interface is selected

*Note: This configuration must be done while the MAC is under reset and before enabling the MAC clocks.*

Bits 22:19 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.

### 7.2.3 SYSCFG external interrupt configuration register 1 (SYSCFG\_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw												

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

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

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1000: PI[x] pin
- 1001: PJ[x] pin
- 1010: PK[x] pin

### 7.2.4 SYSCFG external interrupt configuration register 2 (SYSCFG\_EXTICR2)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
rw	rw	rw	rw												

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

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

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1000: PI[x] pin
- 1001: PJ[x] pin
- 1010: PK[x] pin

## 7.2.5 SYSCFG external interrupt configuration register 3 (SYSCFG\_EXTICR3)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

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

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1000: PI[x] pin
- 1001: PJ[x] pin
- 1010: PK[x] pin

*Note:* PK[11:8] are not used

### 7.2.6 SYSCFG external interrupt configuration register 4 (SYSCFG\_EXTICR4)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw												

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

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

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1001: PJ[x] pin
- 1010: PK[x] pin

Note: PK[15:12] are not used

### 7.2.7 Compensation cell control register (SYSCFG\_CMPCR)

Address offset: 0x20

Reset value: 0x0000 0000

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

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

Bit 8 **READY**: Compensation cell ready flag

- 0: I/O compensation cell not ready
- 1: I/O compensation cell ready

Bits 7:2 Reserved, must be kept at reset value.

Bit 0 **CMP\_PD**: Compensation cell power-down

- 0: I/O compensation cell power-down mode
- 1: I/O compensation cell enabled

## 7.2.8 SYSCFG register maps

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

**Table 24. SYSCFG register map and reset values**

Offset	Register	Reset value
0x00	SYSCFG_MEMRMP	31
0x04	SYSCFG_PMC	Res. 30
0x08	SYSCFG_EXTICR1	Res. 29
0x0C	SYSCFG_EXTICR2	Res. 28
0x10	SYSCFG_EXTICR3	Res. 27
0x14	SYSCFG_EXTICR4	Res. 26
0x20	SYSCFG_CMPCR	Res. 25
		o MEM_BOOT 0
		o MII_RMII_SEL 24
		o ADC1DC2 17
		o ADC2DC2 16
		o SWP_FMC[1:0] 15
		o EXTI10[3:0] 14
		o EXTI11[3:0] 13
		o EXTI12[3:0] 12
		o EXTI13[3:0] 11
		o EXTI14[3:0] 10
		o EXTI15[3:0] 9
		o EXTI16[3:0] 8
		o EXTI17[3:0] 7
		o EXTI18[3:0] 6
		o EXTI19[3:0] 5
		o EXTI10[3:0] 4
		o EXTI12[3:0] 3
		o EXTI14[3:0] 2
		o EXTI16[3:0] 1
		o CMP_PD 0
		o READY 0

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 8 Direct memory access controller (DMA)

### 8.1 DMA introduction

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

The DMA controller combines a powerful dual AHB master bus architecture with independent FIFO to optimize the bandwidth of the system, based on a complex bus matrix architecture.

The two DMA controllers (DMA1, DMA2) have 8 streams each, dedicated to managing memory access requests from one or more peripherals.

Each stream can have up to 8 channels (requests) in total.

Each DMA controller has an arbiter for handling the priority between DMA requests.

### 8.2 DMA main features

The main DMA features are:

- Dual AHB master bus architecture, one dedicated to memory accesses and one dedicated to peripheral accesses
- AHB slave programming interface supporting only 32-bit accesses
- 8 streams for each DMA controller, up to 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 to be:
  - a regular channel that supports peripheral-to-memory, memory-to-peripheral and memory-to-memory transfers
  - a double buffer channel that also supports double buffering on the memory side
- Priorities between DMA stream requests are software-programmable (four levels consisting of very high, high, medium, low) or hardware in case of equality (for example, request 0 has priority over request 1)
- Each stream also supports software trigger for memory-to-memory transfers (only available for the DMA2 controller)
- Each stream request can be selected among up to 8 possible channel requests. This selection is software-configurable and allows several peripherals to initiate DMA requests
- The number of data items to be transferred can be managed either by the DMA controller or by the peripheral:

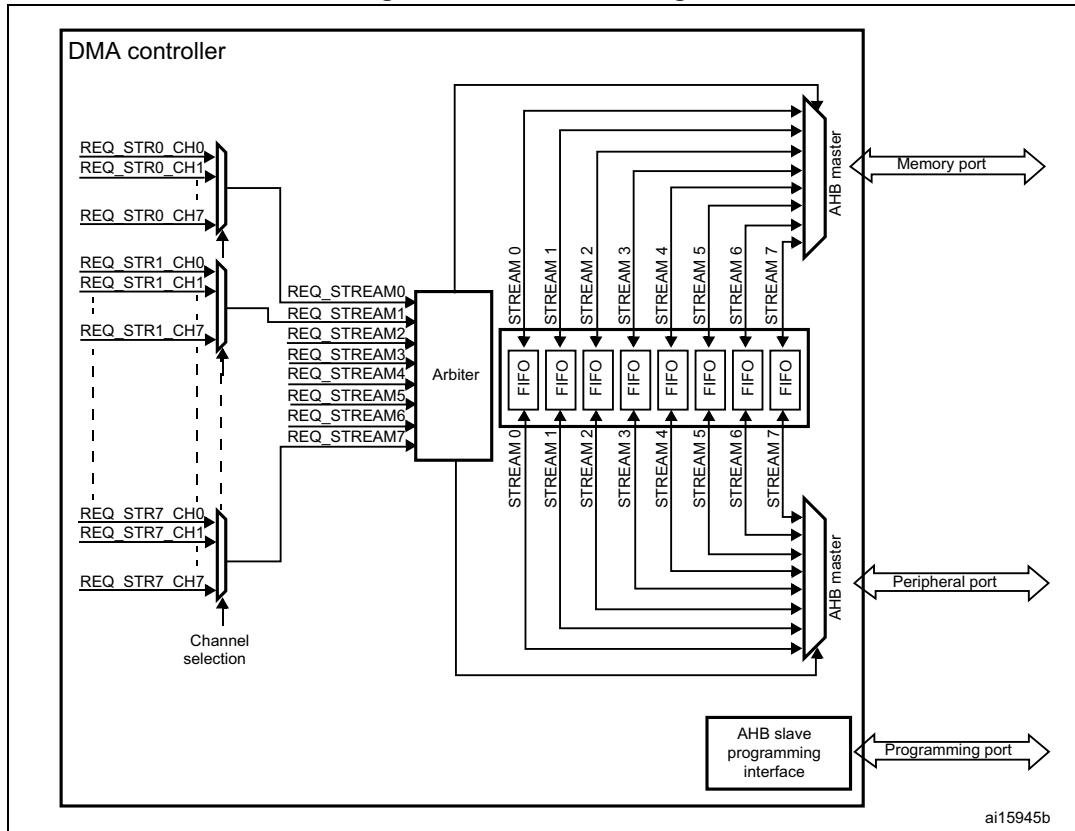
- DMA flow controller: the number of data items to be transferred is software-programmable from 1 to 65535
- Peripheral flow controller: the number of data items to be transferred is unknown and controlled by the source or the destination peripheral that signals the end of the transfer by hardware
- Independent source and destination transfer width (byte, half-word, word): when the data widths of the source and destination are not equal, the DMA automatically packs/unpacks the necessary transfers to optimize the bandwidth. This feature is only available in FIFO mode
- Incrementing or non-incrementing addressing for source and destination
- Supports incremental burst transfers of 4, 8 or 16 beats. The size of the burst is software-configurable, usually equal to half the FIFO size of the peripheral
- Each stream supports circular buffer management
- 5 event flags (DMA half transfer, DMA transfer complete, DMA transfer error, DMA FIFO error, direct mode error) logically ORed together in a single interrupt request for each stream

## 8.3 DMA functional description

### 8.3.1 DMA block diagram

The figure below shows the block diagram of a DMA.

**Figure 23. DMA block diagram**



### 8.3.2 DMA overview

The DMA controller performs direct memory transfer: as an AHB master, the DMA controller can take the control of the AHB bus matrix to initiate AHB transactions.

The DMA controller carries out the following transactions:

- peripheral-to-memory
- memory-to-peripheral
- memory-to-memory

The DMA controller provides two AHB master ports: the AHB memory port, intended to be connected to memories and the AHB peripheral port, intended to be connected to peripherals. However, to allow memory-to-memory transfers, the AHB peripheral port must also have access to the memories.

The AHB slave port is used to program the DMA controller (it supports only 32-bit accesses).

**Note:** The DMA1 controller AHB peripheral port is not connected to the bus matrix like in the case of the DMA2 controller, thus only DMA2 streams are able to perform memory-to-memory transfers.

See [Figure 1](#) for the implementation of the system of two DMA controllers.

### 8.3.3 DMA transactions

A DMA transaction consists of a sequence of a given number of data transfers. The number of data items to be transferred and their width (8-bit, 16-bit or 32-bit) are software-programmable.

Each DMA transfer consists of three operations:

- a loading from the peripheral data register or a location in memory, addressed through the DMA\_SxPAR or DMA\_SxM0AR register
- a storage of the data loaded to the peripheral data register or a location in memory addressed through the DMA\_SxPAR or DMA\_SxM0AR register
- a post-decrement of the DMA\_SxNDTR register, containing the number of transactions that still have to be performed

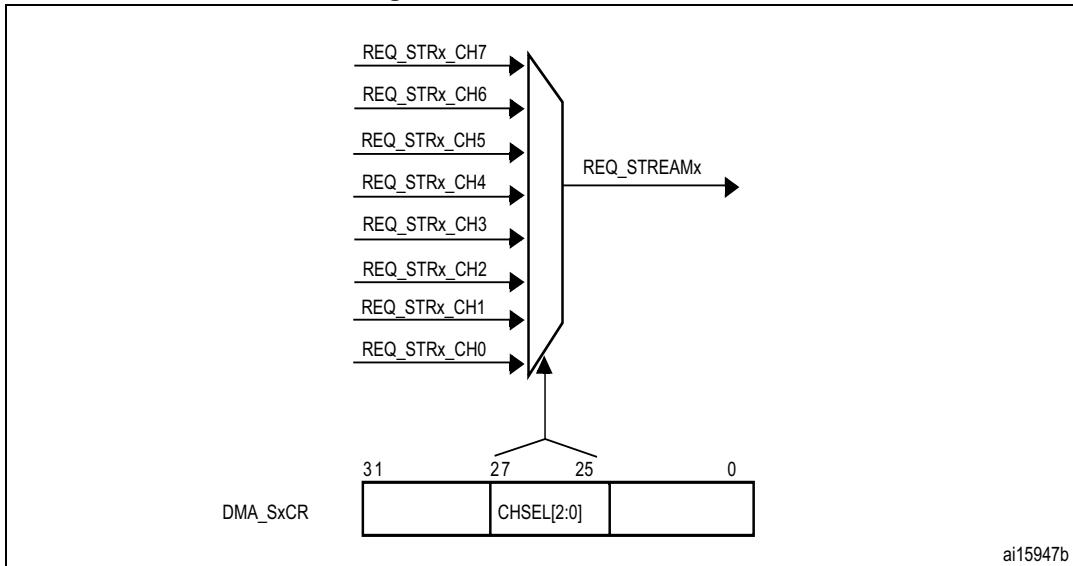
After an event, the peripheral sends a request signal to the DMA controller. The DMA controller serves the request depending on the channel priorities. As soon as the DMA controller accesses the peripheral, an Acknowledge signal is sent to the peripheral by the DMA controller. The peripheral releases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

### 8.3.4 Channel selection

Each stream is associated with a DMA request that can be selected out of 816 possible channel requests. The selection is controlled by the CHSEL[23:0] bits in the DMA\_SxCR register.

**Caution:** A same peripheral request can be assigned to two different channels only if the application ensures that these channels are not requested to be served at the same time. In other words, if two different channels receive a same asserted peripheral request at the same time, an unpredictable DMA hardware behavior occurs.

Figure 24. Channel selection



ai15947b

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

[Table 25](#) and [Table 26](#) give examples of DMA request mappings.

Table 25. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	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	-	I2C4_RX	TIM4_CH2	-	I2C4_TX	TIM4_UP	TIM4_CH3
Channel 3	-	TIM2_UP TIM2_CH3	I2C3_RX	-	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX	UART7_TX	TIM3_CH4 TIM3_UP	UART7_RX	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	-	TIM5_UP	-
Channel 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Table 26. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A	ADC1	SAI1_B	TIM1_CH1 TIM1_CH2 TIM1_CH3	SAI2_B
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B	SPI6_TX	SPI6_RX	DCMI
Channel 2	ADC3	ADC3	-	SPI5_RX	SPI5_TX	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	SAI2_A	SPI1_TX	SAI2_B	QUADSPI
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDMMC1	-	USART1_RX	SDMMC1	USART1_TX

**Table 26. DMA2 request mapping (continued)**

<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 5	-	USART6_RX	USART6_RX	SPI4_RX	SPI4_TX	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX	SPI5_TX	TIM8_CH4 TIM8_TRIG TIM8_COM

### 8.3.5 Arbiter

An arbiter manages the 8 DMA stream requests based on their priority for each of the two AHB master ports (memory and peripheral ports) and launches the peripheral/memory access sequences.

Priorities are managed in two stages:

- Software: each stream priority can be configured in the DMA\_SxCR register. There are four levels:
  - Very high priority
  - High priority
  - Medium priority
  - Low priority
- Hardware: If two requests have the same software priority level, the stream with the lower number takes priority over the stream with the higher number. For example, stream 2 takes priority over stream 4.

### 8.3.6 DMA streams

Each of the eight DMA controller streams provides a unidirectional transfer link between a source and a destination.

Each stream can be configured to perform:

- Regular type transactions: memory-to-peripherals, peripherals-to-memory or memory-to-memory transfers
- Double-buffer type transactions: double buffer transfers using two memory pointers for the memory (while the DMA is reading/writing from/to a buffer, the application can write/read to/from the other buffer).

The amount of data to be transferred (up to 65535) is programmable and related to the source width of the peripheral that requests the DMA transfer connected to the peripheral AHB port. The register that contains the amount of data items to be transferred is decremented after each transaction.

### 8.3.7 Source, destination and transfer modes

Both source and destination transfers can address peripherals and memories in the entire 4-Gbyte 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.

The table below describes the corresponding source and destination addresses.

**Table 27. 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.

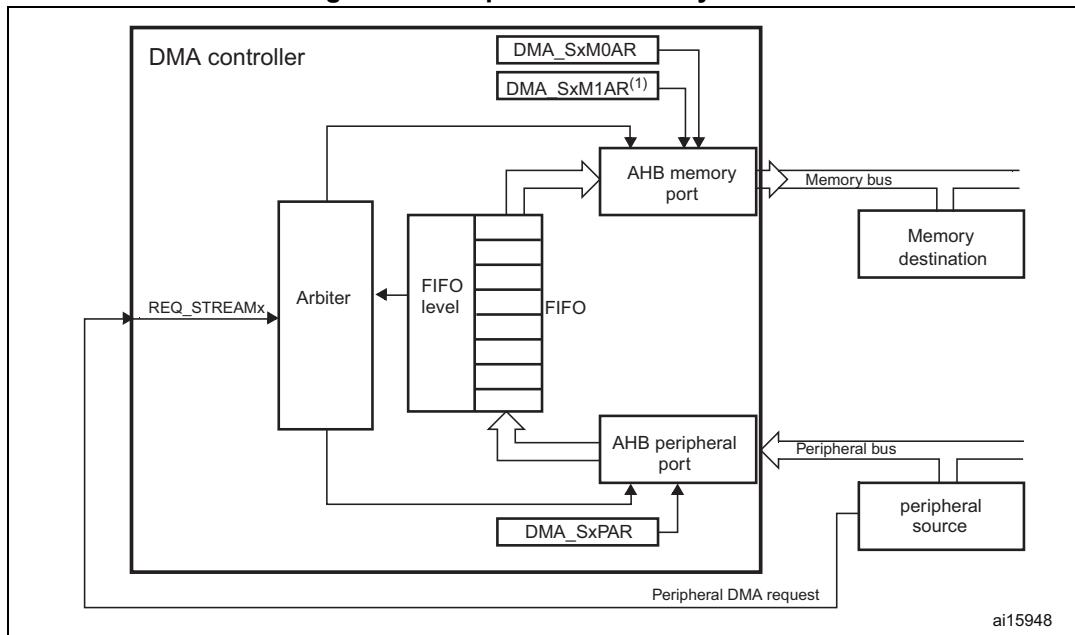
When this mode is enabled (by setting the bit EN in the DMA\_SxCR register), each time a peripheral request occurs, the stream initiates a transfer from the source to fill the FIFO.

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.

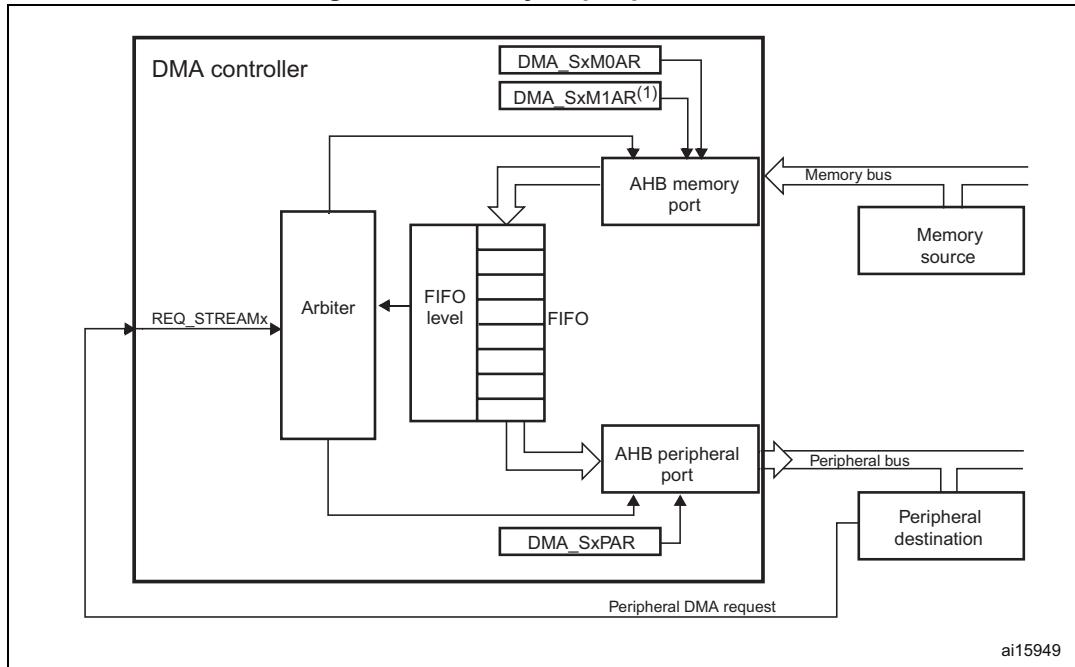
Each time a peripheral request occurs, the contents of the FIFO are drained and stored into the destination. When the level of the FIFO is lower than or equal to the predefined threshold level, the FIFO is fully reloaded with data from the memory.

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.

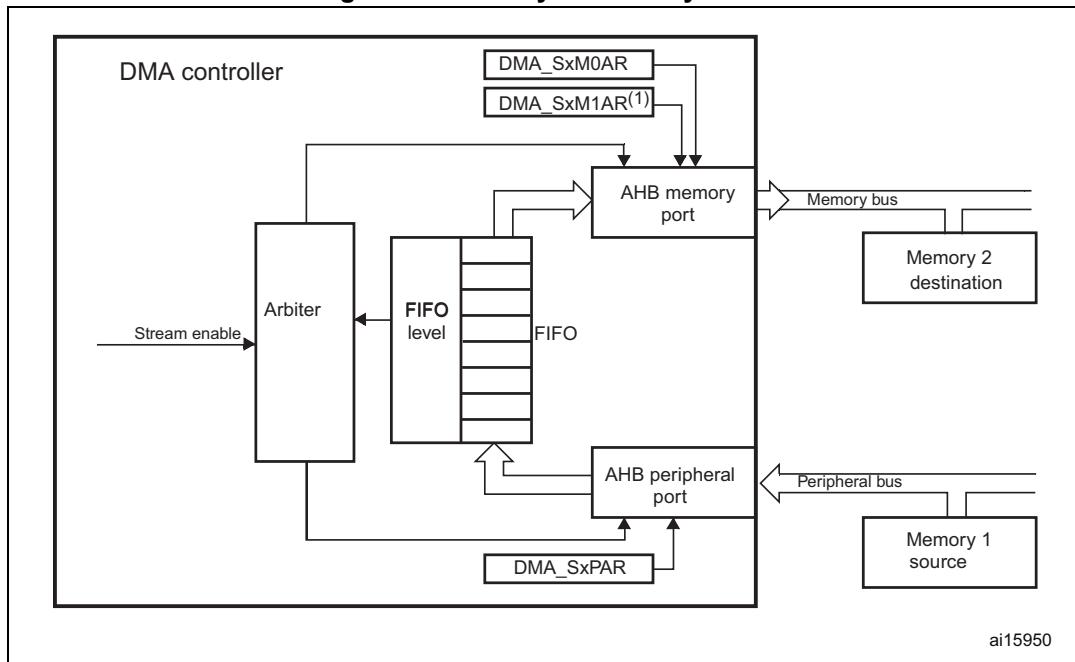
The transfer stops once the DMA\_SxNDTR register reaches zero or when the EN bit in the DMA\_SxCR register is cleared by software.

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.

### 8.3.8 Pointer incrementation

Peripheral and memory pointers can optionally be automatically post-incremented or kept constant after each transfer depending on the PINC and MINC bits in the DMA\_SxCR register.

Disabling the increment mode is useful when the peripheral source or destination data is accessed through a single register.

If the increment mode is enabled, the address of the next transfer is the address of the previous one incremented by 1 (for bytes), 2 (for half-words) or 4 (for words) depending on the data width programmed in the PSIZE or MSIZE bits in the DMA\_SxCR register.

In order to optimize the packing operation, it is possible to fix the increment offset size for the peripheral address whatever the size of the data transferred on the AHB peripheral port. The PINCOS bit in the DMA\_SxCR register is used to align the increment offset size with the data size on the peripheral AHB port, or on a 32-bit address (the address is then incremented by 4). The PINCOS bit has an impact on the AHB peripheral port only.

If the PINCOS bit is set, the address of the following transfer is the address of the previous one incremented by 4 (automatically aligned on a 32-bit address), whatever the PSIZE value. The AHB memory port, however, is not impacted by this operation.

### 8.3.9 Circular mode

The circular mode is available to handle circular buffers and continuous data flows (e.g. ADC scan mode). This feature can be enabled using the CIRC bit in the DMA\_SxCR register.

When the circular mode is activated, the number of data items to be transferred is automatically reloaded with the initial value programmed during the stream configuration phase, and the DMA requests continue to be served.

**Note:** *In the circular mode, it is mandatory to respect the following rule in case of a burst mode configured for memory:*

$$\text{DMA\_SxNDTR} = \text{Multiple of } ((\text{Mburst beat}) \times (\text{Msize})/(\text{Psize})), \text{ where:}$$

- $(\text{Mburst beat}) = 4, 8 \text{ or } 16$  (depending on the MBURST bits in the DMA\_SxCR register)
- $((\text{Msize})/(\text{Psize})) = 1, 2, 4, 1/2 \text{ or } 1/4$  (Msize and Psize represent the MSIZE and PSIZE bits in the DMA\_SxCR register. They are byte dependent)
- $\text{DMA\_SxNDTR} = \text{Number of data items to transfer on the AHB peripheral port}$

For example: Mburst beat = 8 (INCR8), MSIZE = 00 (byte) and PSIZE = 01 (half-word), in this case: DMA\_SxNDTR must be a multiple of  $(8 \times 1/2 = 4)$ .

If this formula is not respected, the DMA behavior and data integrity are not guaranteed.

NDTR must also be a multiple of the Peripheral burst size multiplied by the peripheral data size, otherwise this could result in a bad DMA behavior.

### 8.3.10 Double-buffer mode

This mode is available for all the DMA1 and DMA2 streams.

The double-buffer mode is enabled by setting the DBM bit in the DMA\_SxCR register.

A double-buffer stream works as a regular (single buffer) stream with the difference that it has two memory pointers. When the double-buffer mode is enabled, the circular mode is automatically enabled (CIRC bit in DMA\_SxCR is not relevant) and at each end of transaction, the memory pointers are swapped.

In this mode, the DMA controller swaps from one memory target to another at each end of transaction. This allows the software to process one memory area while the second memory area is being filled/used by the DMA transfer. The double-buffer stream can work in both directions (the memory can be either the source or the destination) as described in [Table 28](#).

**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 28. 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.

### 8.3.11 Programmable data width, packing/unpacking, endianness

The number of data items to be transferred has to be programmed into DMA\_SxNDTR (number of data items to transfer bit, NDT) before enabling the stream (except when the flow controller is the peripheral, PFCTRL bit in DMA\_SxCR is set).

When using the internal FIFO, the data widths of the source and destination data are programmable through the PSIZE and MSIZE bits in the DMA\_SxCR register (can be 8-, 16- or 32-bit).

When PSIZE and MSIZE are not equal:

- The data width of the number of data items to transfer, configured in the DMA\_SxNDTR register is equal to the width of the peripheral bus (configured by the PSIZE bits in the DMA\_SxCR register). For instance, in case of peripheral-to-memory, memory-to-peripheral or memory-to-memory transfers and if the PSIZE[1:0] bits are configured for half-word, the number of bytes to be transferred is equal to  $2 \times \text{NDT}$ .
- The DMA controller only copes with little-endian addressing for both source and destination. This is described in [Table 29](#).

This packing/unpacking procedure may present a risk of data corruption when the operation is interrupted before the data are completely packed/unpacked. So, to ensure data coherence, the stream may be configured to generate burst transfers: in this case, each group of transfers belonging to a burst are indivisible (refer to [Section 8.3.12](#)).

In direct mode (DMDIS = 0 in the DMA\_SxFIFO register), the packing/unpacking of data is not possible. In this case, it is not allowed to have different source and destination transfer data widths: both are equal and defined by the PSIZE bits in the DMA\_SxCR register. MSIZE bits are not relevant.

**Table 29. Packing/unpacking and endian behavior (bit PINC = MINC = 1)**

AHB memory port width	AHB peripheral port width	Number of data items to transfer (NDT)	Memory transfer number	Memory port address / byte lane	Peripheral transfer number	Peripheral port address / byte lane	
						PINCOS = 1	PINCOS = 0
8	8	4	1	0x0 / B0[7:0]	1	0x0 / B0[7:0]	0x0 / B0[7:0]
			2	0x1 / B1[7:0]	2	0x4 / B1[7:0]	0x1 / B1[7:0]
			3	0x2 / B2[7:0]	3	0x8 / B2[7:0]	0x2 / B2[7:0]
			4	0x3 / B3[7:0]	4	0xC / B3[7:0]	0x3 / B3[7:0]
8	16	2	1	0x0 / B0[7:0]	1	0x0 / B1 B0[15:0]	0x0 / B1 B0[15:0]
			2	0x1 / B1[7:0]	2	0x4 / B3 B2[15:0]	0x2 / B3 B2[15:0]
			3	0x2 / B2[7:0]			
			4	0x3 / B3[7:0]			
8	32	1	1	0x0 / B0[7:0]	1	0x0 / B3 B2 B1 B0[31:0]	0x0 / B3 B2 B1 B0[31:0]
			2	0x1 / B1[7:0]			
			3	0x2 / B2[7:0]			
			4	0x3 / B3[7:0]			
16	8	4	1	0x0 / B1 B0[15:0]	1	0x0 / B0[7:0]	0x0 / B0[7:0]
			2	0x2 / B3 B2[15:0]	2	0x4 / B1[7:0]	0x1 / B1[7:0]
			3		3	0x8 / B2[7:0]	0x2 / B2[7:0]
			4		4	0xC / B3[7:0]	0x3 / B3[7:0]
16	16	2	1	0x0 / B1 B0[15:0]	1	0x0 / B1 B0[15:0]	0x0 / B1 B0[15:0]
			2	0x2 / B1 B0[15:0]	2	0x4 / B3 B2[15:0]	0x2 / B3 B2[15:0]
16	32	1	1	0x0 / B1 B0[15:0]	1	0x0 / B3 B2 B1 B0[31:0]	0x0 / B3 B2 B1 B0[31:0]
			2	0x2 / B3 B2[15:0]			
32	8	4	1	0x0 / B3 B2 B1 B0[31:0]	1	0x0 / B0[7:0]	0x0 / B0[7:0]
			2		2	0x4 / B1[7:0]	0x1 / B1[7:0]
			3		3	0x8 / B2[7:0]	0x2 / B2[7:0]
			4		4	0xC / B3[7:0]	0x3 / B3[7:0]
32	16	2	1	0x0 / B3 B2 B1 B0[31:0]	1	0x0 / B1 B0[15:0]	0x0 / B1 B0[15:0]
			2		2	0x4 / B3 B2[15:0]	0x2 / B3 B2[15:0]
32	32	1	1	0x0 / B3 B2 B1 B0[31:0]	1	0x0 / B3 B2 B1 B0[31:0]	0x0 / B3 B2 B1 B0[31:0]

**Note:** Peripheral port may be the source or the destination (it can also be the memory source in the case of memory-to-memory transfer).

PSIZE, MSIZE and NDT[15:0] must be configured so as to ensure that the last transfer is not 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 the table below.

**Table 30. Restriction on NDT versus PSIZE and MSIZE**

PSIZE[1:0] of DMA_SxCR	MSIZE[1:0] of DMA_SxCR	NDT[15:0] of DMA_SxNDTR
00 (8-bit)	01 (16-bit)	Must be a multiple of 2
00 (8-bit)	10 (32-bit)	Must be a multiple of 4
01 (16-bit)	10 (32-bit)	Must be a multiple of 2

### 8.3.12 Single and burst transfers

The DMA controller can generate single transfers or incremental burst transfers of 4, 8, or 16 beats.

The size of the burst is configured by software independently for the two AHB ports by using the MBURST[1:0] and PBURST[1:0] bits in the DMA\_SxCR register.

The burst size indicates the number of beats in the burst, not the number of bytes transferred.

To ensure data coherence, each group of transfers that form a burst are indivisible: AHB transfers are locked and the arbiter of the AHB bus matrix does not degrant the DMA master during the sequence of the burst transfer.

Depending on the single or burst configuration, each DMA request initiates a different number of transfers on the AHB peripheral port:

- When the AHB peripheral port is configured for single transfers, each DMA request generates a data transfer of a byte, half-word or word depending on the PSIZE[1:0] bits in the DMA\_SxCR register
- When the AHB peripheral port is configured for burst transfers, each DMA request generates 4,8 or 16 beats of byte, half word or word transfers depending on the PBURST[1:0] and PSIZE[1:0] bits in the DMA\_SxCR register.

The same as above has to be considered for the AHB memory port considering the MBURST and MSIZE bits.

In direct mode, the stream can only generate single transfers and the MBURST[1:0] and PBURST[1:0] bits are forced by hardware.

The address pointers (DMA\_SxPAR or DMA\_SxM0AR registers) must be chosen so as to ensure that all transfers within a burst block are aligned on the address boundary equal to the size of the transfer.

The burst configuration has to be selected in order to respect the AHB protocol, where bursts **must not** cross the 1 Kbyte address boundary because the minimum address space that can be allocated to a single slave is 1 Kbyte. This means that the 1-Kbyte address boundary **must not** be crossed by a burst block transfer, otherwise an AHB error is generated, that is not reported by the DMA registers.

### 8.3.13 FIFO

#### FIFO structure

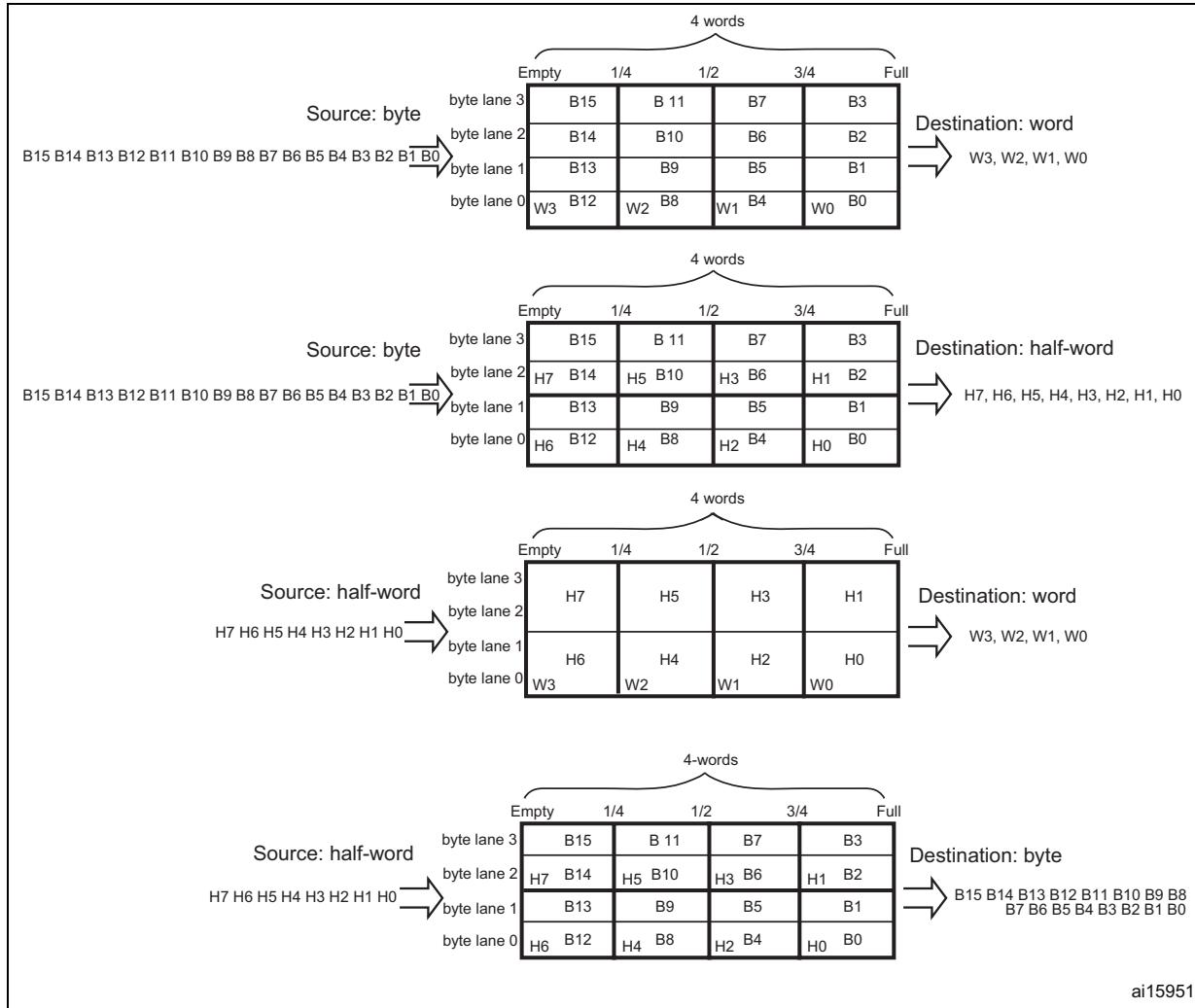
The FIFO is used to temporarily store data coming from the source before transmitting them to the destination.

Each stream has an independent 4-word FIFO and the threshold level is software-configurable between 1/4, 1/2, 3/4 or full.

To enable the use of the FIFO threshold level, the direct mode must be disabled by setting the DMDIS bit in the DMA\_SxFCR register.

The structure of the FIFO differs depending on the source and destination data widths, and is described in the figure below.

**Figure 28. FIFO structure**



### 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 FEIFx of the DMA\_HISR or DMA\_LISR register) is generated when the stream is enabled, then the stream is automatically disabled. The allowed and forbidden configurations are described in the table below. The forbidden configurations are highlighted in gray in the table.

**Table 31. 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
Half-word	1/4	Forbidden	Forbidden	Forbidden
	1/2	1 burst of 4 beats		
	3/4	Forbidden		
	Full	2 bursts of 4 beats	1 burst of 8 beats	
Word	1/4	Forbidden	Forbidden	Forbidden
	1/2			
	3/4			
	Full	1 burst of 4 beats		

In all cases, the burst size multiplied by the data size must not exceed the FIFO size (data size can be: 1 (byte), 2 (half-word) or 4 (word)).

Incomplete burst transfer at the end of a DMA transfer may happen if one of the following conditions occurs:

- For the AHB peripheral port configuration: the total number of data items (set in the DMA\_SxNDTR register) is not a multiple of the burst size multiplied by the data size.
- For the AHB memory port configuration: the number of remaining data items in the FIFO to be transferred to the memory is not a multiple of the burst size multiplied by the data size.

In such cases, the remaining data to be transferred is managed in single mode by the DMA, even if a burst transaction is requested during the DMA stream configuration.

**Note:**

*When burst transfers are requested on the peripheral AHB port and the FIFO is used (DMDIS = 1 in the DMA\_SxCR register), it is mandatory to respect the following rule to avoid permanent underrun or overrun conditions, depending on the DMA stream direction:*

*If  $(PBURST \times PSIZE) = FIFO\_SIZE$  (4 words), FIFO\_Threshold = 3/4 is forbidden with PSIZE = 1, 2 or 4 and PBURST = 4, 8 or 16.*

*This rule ensures that enough FIFO space at a time is free to serve the request from the peripheral.*

### FIFO flush

The FIFO can be flushed when the stream is disabled by resetting the EN bit in the DMA\_SxCR register and when the stream is configured to manage peripheral-to-memory or memory-to-memory transfers. If some data are still present in the FIFO when the stream is disabled, the DMA controller continues transferring the remaining data to the destination (even though stream is effectively disabled). When this flush is completed, the transfer complete status bit (TCIFx) in the DMA\_LISR or DMA\_HISR register is set.

The remaining data counter DMA\_SxNDTR keeps the value in this case to indicate how many data items are currently available in the destination memory.

Note that during the FIFO flush operation, if the number of remaining data items in the FIFO to be transferred to memory (in bytes) is less than the memory data width (for example 2 bytes in FIFO while MSIZE is configured to word), data is sent with the data width set in the MSIZE bit in the DMA\_SxCR register. This means that memory is written with an undesired value. The software may read the DMA\_SxNDTR register to determine the memory area that contains the good data (start address and last address).

If the number of remaining data items in the FIFO is lower than a burst size (if the MBURST bits in DMA\_SxCR register are set to configure the stream to manage burst on the AHB memory port), single transactions are generated to complete the FIFO flush.

### Direct mode

By default, the FIFO operates in direct mode (DMDIS bit in the DMA\_SxFcR is reset) and the FIFO threshold level is not used. This mode is useful when the system requires an immediate and single transfer to or from the memory after each DMA request.

When the DMA is configured in direct mode (FIFO disabled), to transfer data in memory-to-peripheral mode, the DMA preloads one data from the memory to the internal FIFO to ensure an immediate data transfer as soon as a DMA request is triggered by a peripheral.

To avoid saturating the FIFO, it is recommended to configure the corresponding stream with a high priority.

This mode is restricted to transfers where:

- the source and destination transfer widths are equal and both defined by the PSIZE[1:0] bits in DMA\_SxCR (MSIZE[1:0] bits are not relevant)
- burst transfers are not possible (PBURST[1:0] and MBURST[1:0] bits in DMA\_SxCR are don't care)

Direct mode must not be used when implementing memory-to-memory transfers.

### 8.3.14 DMA transfer completion

Different events can generate an end of transfer by setting the TCIFx bit in the DMA\_LISR or DMA\_HISR status register:

- In DMA flow controller mode:
  - The DMA\_SxNDTR counter has reached zero in the memory-to-peripheral mode.
  - The stream is disabled before the end of transfer (by clearing the EN bit in the DMA\_SxCR register) and (when transfers are peripheral-to-memory or memory-

to-memory) all the remaining data have been flushed from the FIFO into the memory.

- In Peripheral flow controller mode:
  - The last external burst or single request has been generated from the peripheral and (when the DMA is operating in peripheral-to-memory mode) the remaining data have been transferred from the FIFO into the memory
  - The stream is disabled by software, and (when the DMA is operating in peripheral-to-memory mode) the remaining data have been transferred from the FIFO into the memory

**Note:** *The transfer completion is dependent on the remaining data in FIFO to be transferred into memory only in the case of peripheral-to-memory mode. This condition is not applicable in memory-to-peripheral mode.*

If the stream is configured in non-circular mode, after the end of the transfer (that is when the number of data to be transferred reaches zero), the DMA is stopped (EN bit in DMA\_SxCR register is cleared by Hardware) and no DMA request is served unless the software reprograms the stream and re-enables it (by setting the EN bit in the DMA\_SxCR register).

### 8.3.15 DMA transfer suspension

At any time, a DMA transfer can be suspended to be restarted later on or to be definitively disabled before the end of the DMA transfer.

There are two cases:

- The stream disables the transfer with no later-on restart from the point where it was stopped. There is no particular action to do, except to clear the EN bit in the DMA\_SxCR register to disable the stream. The stream may take time to be disabled (ongoing transfer is completed first). The transfer complete interrupt flag (TCIF in the DMA\_LISR or DMA\_HISR register) is set in order to indicate the end of transfer. The value of the EN bit in DMA\_SxCR is now 0 to confirm the stream interruption. The DMA\_SxNDTR register contains the number of remaining data items at the moment when the stream was stopped so that the software can determine how many data items have been transferred before the stream was interrupted.
- The stream suspends the transfer before the number of remaining data items to be transferred in the DMA\_SxNDTR register reaches 0. The aim is to restart the transfer later by re-enabling the stream. In order to restart from the point where the transfer was stopped, the software has to read the DMA\_SxNDTR register after disabling the stream by writing the EN bit in DMA\_SxCR register (and then checking that it is at 0) to know the number of data items already collected. Then:
  - The peripheral and/or memory addresses have to be updated in order to adjust the address pointers
  - The SxNDTR register has to be updated with the remaining number of data items to be transferred (the value read when the stream was disabled)
  - The stream may then be re-enabled to restart the transfer from the point it was stopped

**Note:** *A transfer complete interrupt flag (TCIF in DMA\_LISR or DMA\_HISR) is set to indicate the end of transfer due to the stream interruption.*

### 8.3.16 Flow controller

The entity that controls the number of data to be transferred is known as the flow controller. This flow controller is configured independently for each stream using the PFCTRL bit in the DMA\_SxCR register.

The flow controller can be:

- The DMA controller: in this case, the number of data items to be transferred is programmed by software into the DMA\_SxNDTR register before the DMA stream is enabled.
- The peripheral source or destination: this is the case when the number of data items to be transferred is unknown. The peripheral indicates by hardware to the DMA controller when the last data are being transferred. This feature is only supported for peripherals that are able to signal the end of the transfer, that is: SDMMC1.

When the peripheral flow controller is used for a given stream, the value written into the DMA\_SxNDTR has no effect on the DMA transfer. Actually, whatever the value written, it is forced by hardware to 0xFFFF as soon as the stream is enabled, to respect the following schemes:

- Anticipated stream interruption: EN bit in DMA\_SxCR register is reset to 0 by the software to stop the stream before the last data hardware signal (single or burst) is sent by the peripheral. In such a case, the stream is switched off and the FIFO flush is triggered in the case of a peripheral-to-memory DMA transfer. The TCIFx flag of the corresponding stream is set in the status register to indicate the DMA completion. To know the number of data items transferred during the DMA transfer, read the DMA\_SxNDTR register and apply the following formula:
  - Number\_of\_data\_transferred = 0xFFFF – DMA\_SxNDTR
- Normal stream interruption due to the reception of a last data hardware signal: the stream is automatically interrupted when the peripheral requests the last transfer (single or burst) and when this transfer is complete. the TCIFx flag of the corresponding stream is set in the status register to indicate the DMA transfer completion. To know the number of data items transferred, read the DMA\_SxNDTR register and apply the same formula as above.
- The DMA\_SxNDTR register reaches 0: the TCIFx flag of the corresponding stream is set in the status register to indicate the forced DMA transfer completion. The stream is automatically switched off even though the last data hardware signal (single or burst) has not been yet asserted. The already transferred data is not lost. This means that a maximum of 65535 data items can be managed by the DMA in a single transaction, even in peripheral flow control mode.

Note:

*When configured in memory-to-memory mode, the DMA is always the flow controller and the PFCTRL bit is forced to 0 by hardware.*

*The circular mode is forbidden in the peripheral flow controller mode.*

### 8.3.17 Summary of the possible DMA configurations

The table below summarizes the different possible DMA configurations. The forbidden configurations are highlighted in gray in the table.

**Table 32. 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	Forbidden	Peripheral	Forbidden	single	Possible	Forbidden
					burst	Forbidden	
Memory-to-peripheral	AHB memory port	AHB peripheral port	DMA	Possible	single	Possible	Possible
					burst	Forbidden	
	Peripheral	Forbidden	Peripheral	Forbidden	single	Possible	Forbidden
					burst	Forbidden	
Memory-to-memory	AHB peripheral port	AHB memory port	DMA only	Forbidden	single	Forbidden	Forbidden
					burst		

### 8.3.18 Stream configuration procedure

The following sequence must be followed to configure a DMA stream x (where x is the stream number):

1. If the stream is enabled, disable it by resetting the EN bit in the DMA\_SxCR register, then read this bit in order to confirm that there is no ongoing stream operation. Writing this bit to 0 is not immediately effective since it is actually written to 0 once all the current transfers are finished. When the EN bit is read as 0, this means that the stream is ready to be configured. It is therefore necessary to wait for the EN bit to be cleared before starting any stream configuration. All the stream dedicated bits set in the status register (DMA\_LISR and DMA\_HISR) from the previous data block DMA transfer must be cleared before the stream can be re-enabled.
2. Set the peripheral port register address in the DMA\_SxPAR register. The data is moved from/ to this address to/ from the peripheral port after the peripheral event.
3. Set the memory address in the DMA\_SxMA0R register (and in the DMA\_SxMA1R register in the case of a double-buffer mode). The data is written to or read from this memory after the peripheral event.
4. Configure the total number of data items to be transferred in the DMA\_SxNDTR register. After each peripheral event or each beat of the burst, this value is decremented.
5. Select the DMA channel (request) using CHSEL[2:0] in the DMA\_SxCR register.
6. If the peripheral is intended to be the flow controller and if it supports this feature, set the PFCTRL bit in the DMA\_SxCR register.
7. Configure the stream priority using the PL[1:0] bits in the DMA\_SxCR register.
8. Configure the FIFO usage (enable or disable, threshold in transmission and reception)

9. Configure the data transfer direction, peripheral and memory incremented/fixed mode, single or burst transactions, peripheral and memory data widths, circular mode, double-buffer mode and interrupts after half and/or full transfer, and/or errors in the DMA\_SxCR register.
10. Activate the stream by setting the EN bit in the DMA\_SxCR register.

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

Once half the data have been transferred on the AHB destination port, the half-transfer flag (HTIF) is set and an interrupt is generated if the half-transfer interrupt enable bit (HTIE) is set. At the end of the transfer, the transfer complete flag (TCIF) is set and an interrupt is generated if the transfer complete interrupt enable bit (TCIE) is set.

---

**Warning:** To switch off a peripheral connected to a DMA stream request, it is mandatory to, first, switch off the DMA stream to which the peripheral is connected, then to wait for EN bit = 0. Only then can the peripheral be safely disabled.

---

### 8.3.19 Error management

The DMA controller can detect the following errors:

- **Transfer error:** the transfer error interrupt flag (TEIFx) is set when:
  - a bus error occurs during a DMA read or a write access
  - a write access is requested by software on a memory address register in double-buffer mode whereas the stream is enabled and the current target memory is the one impacted by the write into the memory address register (refer to [Section 8.3.10: Double-buffer mode](#))
- **FIFO error:** the FIFO error interrupt flag (FEIFx) is set if:
  - a FIFO underrun condition is detected
  - a FIFO overrun condition is detected (no detection in memory-to-memory mode because requests and transfers are internally managed by the DMA)
  - the stream is enabled while the FIFO threshold level is not compatible with the size of the memory burst (refer to [Table 31: 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 two data items were 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, DMEIE) in the DMA\_SxCR or DMA\_SxFCR register is set.

**Note:** *When a FIFO overrun or underrun condition occurs, the data is not lost because the peripheral request is not acknowledged by the stream until the overrun or underrun condition is cleared. If this acknowledge takes too much time, the peripheral itself may detect an overrun or underrun condition of its internal buffer and data might be lost.*

## 8.4 DMA interrupts

For each DMA stream, an interrupt can be produced on the following events:

- Half-transfer reached
- Transfer complete
- Transfer error
- FIFO error (overrun, underrun or FIFO level error)
- Direct mode error

Separate interrupt enable control bits are available for flexibility as shown in the table below.

**Table 33. DMA interrupt requests**

Interrupt event	Event flag	Enable control bit
Half-transfer	HTIF	HTIE
Transfer complete	TCIF	TCIE
Transfer error	TEIF	TEIE
FIFO overrun/underrun	FEIF	FEIE
Direct mode error	DMEIF	DMEIE

**Note:** *Before setting an enable control bit EN = 1, the corresponding event flag must be cleared, otherwise an interrupt is immediately generated.*

## 8.5 DMA registers

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

### 8.5.1 DMA low interrupt status register (DMA\_LISR)

Address offset: 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.	TCIF3	HTIF3	TEIF3	DMEIF3	Res.	FEIF3	TCIF2	HTIF2	TEIF2	DMEIF2	Res.	FEIF2
				r	r	r	r		r	r	r	r	r		r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	TCIF1	HTIF1	TEIF1	DMEIF1	Res.	FEIF1	TCIF0	HTIF0	TEIF0	DMEIF0	Res.	FEIF0
				r	r	r	r		r	r	r	r	r		r

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

Bits 27, 21, 11, 5 **TCIF[3:0]**: Stream x transfer complete interrupt flag (x = 3 to 0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_LIFCR register.

0: No transfer complete event on stream x

1: A transfer complete event occurred on stream x.

Bits 26, 20, 10, 4 **HTIF[3:0]**: Stream x half transfer interrupt flag (x = 3 to 0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_LIFCR register.

0: No half transfer event on stream x

1: An half transfer event occurred on stream x.

Bits 25, 19, 9, 3 **TEIF[3:0]**: Stream x transfer error interrupt flag (x = 3 to 0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_LIFCR register.

0: No transfer error on stream x

1: A transfer error occurred on stream x.

Bits 24, 18, 8, 2 **DMEIF[3:0]**: Stream x direct mode error interrupt flag (x = 3 to 0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_LIFCR register.

0: No direct mode error on stream x

1: A direct mode error occurred on stream x.

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

Bits 22, 16, 6, 0 **FEIF[3:0]**: Stream x FIFO error interrupt flag (x = 3 to 0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_LIFCR register.

0: No FIFO error event on stream x

1: A FIFO error event occurred on stream x.

### 8.5.2 DMA high interrupt status register (DMA\_HISR)

Address offset: 0x004

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	TCIF7	HTIF7	TEIF7	DMEIF7	Res.	FEIF7	TCIF6	HTIF6	TEIF6	DMEIF6	Res.	FEIF6
				r	r	r	r		r	r	r	r	r		r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	TCIF5	HTIF5	TEIF5	DMEIF5	Res.	FEIF5	TCIF4	HTIF4	TEIF4	DMEIF4	Res.	FEIF4
				r	r	r	r		r	r	r	r	r		r

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

Bits 27, 21, 11, 5 **TCIF[7:4]**: Stream x transfer complete interrupt flag (x = 7 to 4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_HIFCR register.

0: No transfer complete event on stream x

1: A transfer complete event occurred on stream x.

Bits 26, 20, 10, 4 **HTIF[7:4]**: Stream x half transfer interrupt flag (x = 7 to 4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_HIFCR register.

0: No half transfer event on stream x

1: An half transfer event occurred on stream x.

Bits 25, 19, 9, 3 **TEIF[7:4]**: Stream x transfer error interrupt flag (x = 7 to 4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_HIFCR register.

0: No transfer error on stream x

1: A transfer error occurred on stream x.

Bits 24, 18, 8, 2 **DMEIF[7:4]**: Stream x direct mode error interrupt flag (x = 7 to 4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_HIFCR register.

0: No direct mode error on stream x

1: A direct mode error occurred on stream x.

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

Bits 22, 16, 6, 0 **FEIF[7:4]**: Stream x FIFO error interrupt flag (x = 7 to 4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in DMA\_HIFCR register.

0: No FIFO error event on stream x

1: A FIFO error event occurred on stream x.

### 8.5.3 DMA low interrupt flag clear register (DMA\_LIFCR)

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CTCIF3	CHTIF3	CTEIF3	CDMEI F3	Res.	CFEIF3	CTCIF2	CHTIF2	CTEIF2	CDMEI F2	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	CDMEI F1	Res.	CFEIF1	CTCIF0	CHTIF0	CTEIF0	CDMEI F0	Res.	CFEIF0
				w	w	w	w		w	w	w	w	w		w

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

Bits 27, 21, 11, 5 **CTCIF[3:0]**: Stream x clear transfer complete interrupt flag (x = 3 to 0)

Writing 1 to this bit clears the corresponding TCIFx flag in the DMA\_LISR register.

Bits 26, 20, 10, 4 **CHTIF[3:0]**: Stream x clear half transfer interrupt flag (x = 3 to 0)

Writing 1 to this bit clears the corresponding HTIFx flag in the DMA\_LISR register

Bits 25, 19, 9, 3 **CTEIF[3:0]**: Stream x clear transfer error interrupt flag (x = 3 to 0)

Writing 1 to this bit clears the corresponding TEIFx flag in the DMA\_LISR register.

Bits 24, 18, 8, 2 **CDMEIF[3:0]**: Stream x clear direct mode error interrupt flag (x = 3 to 0)

Writing 1 to this bit clears the corresponding DMEIFx flag in the DMA\_LISR register.

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

Bits 22, 16, 6, 0 **CFEIF[3:0]**: Stream x clear FIFO error interrupt flag (x = 3 to 0)

Writing 1 to this bit clears the corresponding CFEIFx flag in the DMA\_LISR register.

### 8.5.4 DMA high interrupt flag clear register (DMA\_HIFCR)

Address offset: 0x00C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CTCIF7	CHTIF7	CTEIF7	CDMEI F7	Res.	CFEIF7	CTCIF6	CHTIF6	CTEIF6	CDMEI F6	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	CDMEI F5	Res.	CFEIF5	CTCIF4	CHTIF4	CTEIF4	CDMEI F4	Res.	CFEIF4
				w	w	w	w		w	w	w	w	w		w

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

Bits 27, 21, 11, 5 **CTCIF[7:4]**: Stream x clear transfer complete interrupt flag (x = 7 to 4)

Writing 1 to this bit clears the corresponding TCIFx flag in the DMA\_HISR register.

Bits 26, 20, 10, 4 **CHTIF[7:4]**: Stream x clear half transfer interrupt flag (x = 7 to 4)

Writing 1 to this bit clears the corresponding HTIFx flag in the DMA\_HISR register.

- Bits 25, 19, 9, 3 **CTEIF[7:4]**: Stream x clear transfer error interrupt flag (x = 7 to 4)  
Writing 1 to this bit clears the corresponding TEIF<sub>x</sub> flag in the DMA\_HISR register.
- Bits 24, 18, 8, 2 **CDMEIF[7:4]**: Stream x clear direct mode error interrupt flag (x = 7 to 4)  
Writing 1 to this bit clears the corresponding DMEIF<sub>x</sub> flag in the DMA\_HISR register.
- Bits 23, 17, 7, 1 Reserved, must be kept at reset value.
- Bits 22, 16, 6, 0 **CFEIF[7:4]**: Stream x clear FIFO error interrupt flag (x = 7 to 4)  
Writing 1 to this bit clears the corresponding CFEIF<sub>x</sub> flag in the DMA\_HISR register.

### 8.5.5 DMA stream x configuration register (DMA\_SxCR)

This register is used to configure the concerned stream.

Address offset: 0x010 + 0x18 \* x, (x = 0 to 7)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	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
PINC OS	MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR[1:0]		PF CTRL	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[1:0]**: Memory burst transfer configuration

These bits are set and cleared by software.  
00: Single transfer  
01: INCR4 (incremental burst of 4 beats)  
10: INCR8 (incremental burst of 8 beats)  
11: INCR16 (incremental burst of 16 beats)

These bits are protected and can be written only if EN = 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 = 0.

In direct mode, these bits are forced to 0x0 by hardware.

Bit 20 Reserved, must be kept at reset value.

Bit 19 **CT**: Current target (only in double-buffer mode)

This bit is set and cleared by hardware. It can also be written by software.

0: Current target memory is memory 0 (addressed by the DMA\_SxM0AR pointer).

1: Current target memory is memory 1 (addressed by the DMA\_SxM1AR pointer).

This bit can be written only if EN = 0 to indicate the target memory area of the first transfer. Once the stream is enabled, this bit operates as a status flag indicating which memory area is the current target.

Bit 18 **DBM**: Double-buffer mode

This bit is set and cleared by software.

0: No buffer switching at the end of transfer

1: Memory target switched at the end of the DMA transfer

This bit is protected and can be written only if EN = 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 = 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 INC = 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 (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 = 0.

In direct mode, MSIZE is forced by hardware to the same value as PSIZE as soon as 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 = 0.

Bit 10 **MINC**: Memory increment mode

This bit is set and cleared by software.

0: Memory address pointer fixed

1: Memory address pointer incremented after each data transfer (increment is done according to MSIZE)

This bit is protected and can be written only if EN = 0.

Bit 9 **PINC**: Peripheral increment mode

This bit is set and cleared by software.

0: Peripheral address pointer fixed

1: Peripheral address pointer incremented after each data transfer (increment done according to PSIZE)

This bit is protected and can be written only if EN = 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 (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 (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 = 0.

Bit 5 **PFCTRL**: Peripheral flow controller

This bit is set and cleared by software.

0: DMA is the flow controller.

1: The peripheral is the flow controller.

This bit is protected and can be written only if EN = 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 is 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 burst size

When this bit is read as 0, the software is allowed to program the configuration and FIFO bits registers. It is forbidden to write these registers when the EN bit is read as 1.

*Note: Before setting EN bit to 1 to start a new transfer, the event flags corresponding to the stream in DMA\_LISR or DMA\_HISR register must be cleared.*

### 8.5.6 DMA stream x number of data register (DMA\_SxNDTR)

Address offset: 0x014 + 0x18 \* x, (x = 0 to 7)

Reset value: 0x0000 0000

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

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

Bits 15:0 **NDT[15:0]**: Number of data items to transfer (0 up to 65535)

This bitfield can be written only when the stream is disabled. When the stream is enabled, this bitfield is read-only, indicating the remaining data items to be transmitted. This bitfield decrements after each DMA transfer.

Once the transfer is completed, this bitfield 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 bitfield is zero, no transaction can be served even if the stream is enabled.

### 8.5.7 DMA stream x peripheral address register (DMA\_SxPAR)

Address offset:  $0x018 + 0x18 * x$ , ( $x = 0$  to  $7$ )

Reset value: 0x0000 0000

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

Bits 31:0 **PAR[31:0]**: Peripheral address

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

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

### 8.5.8 DMA stream x memory 0 address register (DMA\_SxM0AR)

Address offset:  $0x01C + 0x18 * x$ , ( $x = 0$  to  $7$ )

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
M0A[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M0A[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **M0A[31:0]**: Memory 0 address

Base address of memory area 0 from/to which the data is read/written.

These bits are write-protected. They can be written only if:

- the stream is disabled (EN = 0 in DMA\_SxCR) or
- the stream is enabled (EN = 1 in DMA\_SxCR) and CT = 1 in DMA\_SxCR (in double-buffer mode).

### 8.5.9 DMA stream x memory 1 address register (DMA\_SxM1AR)

Address offset:  $0x020 + 0x18 * x$ , ( $x = 0$  to  $7$ )

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
M1A[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M1A[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **M1A[31:0]**: Memory 1 address (used in case of double-buffer mode)

Base address of memory area 1 from/to which the data is read/written.

This bitfield is used only for the double-buffer mode.

These bits are write-protected. They can be written only if:

- the stream is disabled (EN = 0 in DMA\_SxCR) or
- the stream is enabled (EN = 1 in DMA\_SxCR) and bit CT = 0 in DMA\_SxCR .

### 8.5.10 DMA stream x FIFO control register (DMA\_SxFCR)

Address offset: 0x024 + 0x18 \* x, (x = 0 to 7)

Reset value: 0x0000 0021

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
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: Reserved (no meaning)

These bits are not relevant in the direct mode (DMDIS = 0).

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 = 0.

This bit is set by hardware if the memory-to-memory mode is selected (DIR = 10 in DMA\_SxCR), and EN = 1 in DMA\_SxCR 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 = 0.

These bits are protected and can be written only if EN = 0.

### **8.5.11 DMA register map**

**Table 34. DMA register map and reset values**

Table 34. DMA register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0x02C	DMA_S1NDTR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x030	DMA_S1PAR	PA[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		
0x034	DMA_S1M0AR	MOA[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		
0x038	DMA_S1M1AR	M1A[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		
0x03C	DMA_S1FCR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x040	DMA_S2CR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x044	DMA_S2NDTR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x048	DMA_S2PAR	PA[31:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x04C	DMA_S2M0AR	MOA[31:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x050	DMA_S2M1AR	M1A[31:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x054	DMA_S2FCR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x058	DMA_S3CR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x05C	DMA_S3NDTR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x060	DMA_S3PAR	PA[31:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x064	DMA_S3M0AR	MOA[31:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Table 34. DMA register map and reset values (continued)**

**Table 34. DMA register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0x0A4	DMA_S6NDTR	Res																																	
	Reset value																																		
0x0A8	DMA_S6PAR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0AC	DMA_S6M0AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0B0	DMA_S6M1AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0B4	DMA_S6FCR	Res																																	
	Reset value																																		
0x0B8	DMA_S7CR	Res																																	
	Reset value																																		
0x0BC	DMA_S7NDTR	Res																																	
	Reset value																																		
0x0C0	DMA_S7PAR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0C4	DMA_S7M0AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0C8	DMA_S7M1AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0CC	DMA_S7FCR	Res																																	
	Reset value																																		

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

## 9 Chrom-ART Accelerator controller (DMA2D)

### 9.1 DMA2D introduction

The Chrom-ART Accelerator (DMA2D) is a specialized DMA dedicated to image manipulation. It can perform the following operations:

- fill a part or the whole of a destination image with a specific color
- copy a part or the whole of a source image into a part or the whole of a destination image
- copy a part or the whole of a source image into a part or the whole of a destination image with a pixel format conversion
- blend a part and/or two complete source images with different pixel format and copy the result into a part or the whole of a destination image with a different color format

All the classical color coding schemes are supported from 4-bit up to 32-bit per pixel with indexed or direct color mode. The DMA2D has its own dedicated memories for CLUTs (color look-up tables).

### 9.2 DMA2D main features

The main DMA2D features are:

- Single AHB master bus architecture
- AHB slave programming interface supporting 8/16/32-bit accesses (except for CLUT accesses which are 32-bit)
- User programmable working area size
- User programmable offset for sources and destination areas
- User programmable sources and destination addresses on the whole memory space
- Up to 2 sources with blending operation
- Alpha value can be modified (source value, fixed value or modulated value)
- User programmable source and destination color format
- Up to 11 color formats supported from 4-bit up to 32-bit per pixel with indirect or direct color coding
- 2 internal memories for CLUT storage in indirect color mode
- Automatic CLUT loading or CLUT programming via the CPU
- User programmable CLUT size
- Internal timer to control AHB bandwidth
- 4 operating modes: register-to-memory, memory-to-memory, memory-to-memory with pixel format conversion, and memory-to-memory with pixel format conversion and blending
- Area filling with a fixed color
- Copy from an area to another
- Copy with pixel format conversion between source and destination images
- Copy from two sources with independent color format and blending
- Abort and suspend of DMA2D operations
- Watermark interrupt on a user programmable destination line

- Interrupt generation on bus error or access conflict
- Interrupt generation on process completion

## 9.3 DMA2D functional description

### 9.3.1 DMA2D block diagram

The DMA2D controller performs direct memory transfer. As an AHB master, it can take the control of the AHB bus matrix to initiate AHB transactions.

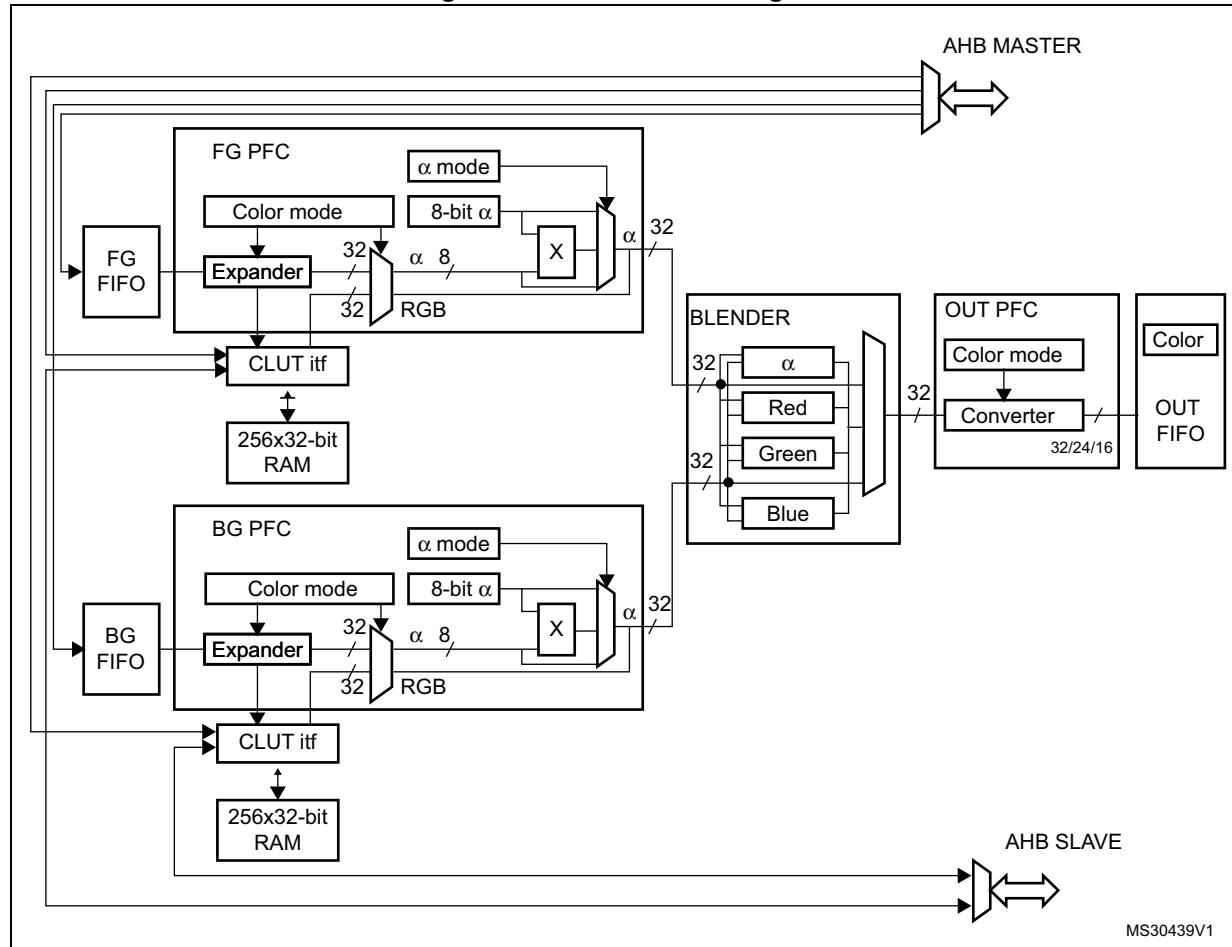
The DMA2D can operate in the following modes:

- Register-to-memory
- Memory-to-memory
- Memory-to-memory with pixel format conversion
- Memory-to-memory with pixel format conversion and blending

The AHB slave port is used to program the DMA2D controller.

The block diagram of the DMA2D is shown in the figure below.

**Figure 29. DMA2D block diagram**



### 9.3.2 DMA2D control

The DMA2D controller is configured through the DMA2D control register (DMA2D\_CR).

The user application can perform the following operations:

- Select the operating mode.
- Enable/disable the DMA2D interrupt.
- Start/suspend/abort ongoing data transfers.

### 9.3.3 DMA2D foreground and background FIFOs

The DMA2D foreground (FG) FG FIFO and background (BG) FIFO fetch the input data to be copied and/or processed.

These FIFOs fetch the pixels according to the color format defined in their respective pixel format converter (PFC). They are programmed through a set of control registers:

- DMA2D foreground memory address register (DMA2D\_FGMAR)
- DMA2D foreground offset register (DMA2D\_FGOR)
- DMA2D background memory address register (DMA2D\_BGMAR)
- DMA2D background offset register (DMA2D\_BGBOR)
- DMA2D number of lines register (number of lines and pixel per lines) (DMA2D\_NLR)

When the DMA2D operates in register-to-memory mode, none of the FIFOs is activated.

When the DMA2D operates in memory-to-memory mode (no pixel format conversion nor blending operation), only the FG FIFO is activated and acts as a buffer.

When the DMA2D operates in memory-to-memory operation with pixel format conversion (no blending operation), the BG FIFO is not activated.

### 9.3.4 DMA2D foreground and background pixel format converter (PFC)

DMA2D foreground pixel format converter (PFC) and background pixel format converter perform the pixel format conversion to generate a 32-bit per pixel value. The PFC can also modify the alpha channel.

The first stage of the converter converts the color format. The original color format of the foreground pixel and background pixels are configured through the CM[3:0] bits of the DMA2D\_FGPCCR and DMA2D\_BGPCCR, respectively.

The supported input formats are given in the table below.

**Table 35. Supported color mode in input**

CM[3:0]	Color mode
0000	ARGB8888
0001	RGB888
0010	RGB565
0011	ARGB1555
0100	ARGB4444
0101	L8

**Table 35. Supported color mode in input (continued)**

CM[3:0]	Color mode
0110	AL44
0111	AL88
1000	L4
1001	A8
1010	A4

The color format is coded as follows:

- Alpha value field: transparency  
0xFF value corresponds to an opaque pixel and 0x00 to a transparent one.
- R field for Red
- G field for Green
- B field for Blue
- L field: luminance

This field is the index to a CLUT to retrieve the three/four RGB/ARGB components.

If the original format is direct color mode (ARGB/RGB), the extension to 8 bits per channel is performed by copying the MSBs into the LSBs. This ensures a perfect linearity of the conversion.

If the original format is indirect color mode (L/AL), a CLUT is required and each pixel format converter is associated with a 256 entry 32-bit CLUT.

If the original format does not include an alpha channel, the alpha value is automatically set to 0xFF (opaque).

For the specific alpha mode A4 and A8, no color information is stored nor indexed. The color to be used for the image generation is fixed and is defined in the DMA2D\_FGCOLR for foreground pixels and in the DMA2D\_BGCOLR register for background pixels.

The order of the fields in the system memory is defined in the table below.

**Table 36. Data order in memory**

Color Mode	@ + 3	@ + 2	@ + 1	@ + 0
ARGB8888	A <sub>0</sub> [7:0]	R <sub>0</sub> [7:0]	G <sub>0</sub> [7:0]	B <sub>0</sub> [7:0]
RGB888	B <sub>1</sub> [7:0]	R <sub>0</sub> [7:0]	G <sub>0</sub> [7:0]	B <sub>0</sub> [7:0]
	G <sub>2</sub> [7:0]	B <sub>2</sub> [7:0]	R <sub>1</sub> [7:0]	G <sub>1</sub> [7:0]
	R <sub>3</sub> [7:0]	G <sub>3</sub> [7:0]	B <sub>3</sub> [7:0]	R <sub>2</sub> [7:0]
	R <sub>1</sub> [4:0]G <sub>1</sub> [5:3]	G <sub>1</sub> [2:0]B <sub>1</sub> [4:0]	R <sub>0</sub> [4:0]G <sub>0</sub> [5:3]	G <sub>0</sub> [2:0]B <sub>0</sub> [4:0]
ARGB1555	A <sub>1</sub> [0]R <sub>1</sub> [4:0]G <sub>1</sub> [4:3]	G <sub>1</sub> [2:0]B <sub>1</sub> [4:0]	A <sub>0</sub> [0]R <sub>0</sub> [4:0]G <sub>0</sub> [4:3]	G <sub>0</sub> [2:0]B <sub>0</sub> [4:0]
ARGB4444	A <sub>1</sub> [3:0]R <sub>1</sub> [3:0]	G <sub>1</sub> [3:0]B <sub>1</sub> [3:0]	A <sub>0</sub> [3:0]R <sub>0</sub> [3:0]	G <sub>0</sub> [3:0]B <sub>0</sub> [3:0]
L8	L <sub>3</sub> [7:0]	L <sub>2</sub> [7:0]	L <sub>1</sub> [7:0]	L <sub>0</sub> [7:0]
AL44	A <sub>3</sub> [3:0]L <sub>3</sub> [3:0]	A <sub>2</sub> [3:0]L <sub>2</sub> [3:0]	A <sub>1</sub> [3:0]L <sub>1</sub> [3:0]	A <sub>0</sub> [3:0]L <sub>0</sub> [3:0]
AL88	A <sub>1</sub> [7:0]	L <sub>1</sub> [7:0]	A <sub>0</sub> [7:0]	L <sub>0</sub> [7:0]

**Table 36. Data order in memory (continued)**

L4	$L_7[3:0]L_6[3:0]$	$L_5[3:0]L_4[3:0]$	$L_3[3:0]L_2[3:0]$	$L_1[3:0]L_0[3:0]$
A8	$A_3[7:0]$	$A_2[7:0]$	$A_1[7:0]$	$A_0[7:0]$
A4	$A_7[3:0]A_6[3:0]$	$A_5[3:0]A_4[3:0]$	$A_3[3:0]A_2[3:0]$	$A_1[3:0]A_0[3:0]$

The 24-bit RGB888 aligned on 32 bits is supported through the ARGB8888 mode.

Once the 32-bit value is generated, the alpha channel can be modified according to AM[1:0] in DMA2D\_FGPFCCR/DMA2D\_BGPFCCR registers as shown in the table below.

The alpha channel can be:

- kept as it is (no modification)
- replaced by the ALPHA[7:0] value of DMA2D\_FGPFCCR/DMA2D\_BGPFCCR
- replaced by the original alpha value multiplied by the ALPHA[7:0] value of DMA2D\_FGPFCCR/DMA2D\_BGPFCCR divided by 255.

**Table 37. Alpha mode configuration**

AM[1:0]	Alpha mode
00	No modification
01	Replaced by value in DMA2D_xxPFCCR
10	Replaced by original value multiplied by the value in DMA2D_xxPFCCR / 255
11	Reserved

### 9.3.5 DMA2D foreground and background CLUT interface

The CLUT interface manages the CLUT memory access and the automatic loading of the CLUT.

Three access types are possible:

- CLUT read by the PFC during pixel format conversion operation
- CLUT accessed through the AHB slave port when the CPU is reading or writing data into the CLUT
- CLUT written through the AHB master port when an automatic loading of the CLUT is performed

The CLUT memory loading can be done in two different ways:

- Automatic loading

The following sequence must be followed to load the CLUT:

- a) Program the CLUT address in DMA2D\_FGCMAR (foreground CLUT) or DMA2D\_BGCMAR (background CLUT).
- b) Program the CLUT size with CS[7:0] in DMA2D\_FGPFCCR (foreground CLUT) or DMA2D\_BGPFCCR (background CLUT).
- c) Set the START bit in DMA2D\_FGPFCCR (foreground CLUT) or DMA2D\_BGPFCCR (background CLUT) to start the transfer. During this automatic loading process, the CLUT is not accessible by the CPU. If a conflict

occurs, a CLUT access error interrupt is raised assuming CAEIE = 1 in DMA2D\_CR.

- Manual loading

The application has to program the CLUT manually through the DMA2D AHB slave port to which the local CLUT memory is mapped. The foreground CLUT (FGCLUT) is located at address offset 0x0400 and the background CLUT (BGCLUT) at address offset 0x0800.

The CLUT format is 24 or 32 bits. It is configured through the CCM bit in DMA2D\_FGPCCR (foreground CLUT) or DMA2D\_BGPCCR (background CLUT) as shown in the table below.

**Table 38. Supported CLUT color mode**

CCM	CLUT color mode
0	32-bit ARGB8888
1	24-bit RGB888

The way the CLUT data are organized in the system memory is specified in the table below.

**Table 39. CLUT data order in system memory**

CLUT color mode	@ + 3	@ + 2	@ + 1	@ + 0
ARGB8888	A <sub>0</sub> [7:0]	R <sub>0</sub> [7:0]	G <sub>0</sub> [7:0]	B <sub>0</sub> [7:0]
RGB888	B <sub>1</sub> [7:0]	R <sub>0</sub> [7:0]	G <sub>0</sub> [7:0]	B <sub>0</sub> [7:0]
	G <sub>2</sub> [7:0]	B <sub>2</sub> [7:0]	R <sub>1</sub> [7:0]	G <sub>1</sub> [7:0]
	R <sub>3</sub> [7:0]	G <sub>3</sub> [7:0]	B <sub>3</sub> [7:0]	R <sub>2</sub> [7:0]

### 9.3.6 DMA2D blender

The DMA2D blender blends the source pixels by pair to compute the resulting pixel.

The blending is performed according to the following equation:

$$\text{with } \alpha_{\text{Mult}} = \frac{\alpha_{\text{FG}} \cdot \alpha_{\text{BG}}}{255}$$

$$\alpha_{\text{OUT}} = \alpha_{\text{FG}} + \alpha_{\text{BG}} - \alpha_{\text{Mult}}$$

$$C_{\text{OUT}} = \frac{C_{\text{FG}} \cdot \alpha_{\text{FG}} + C_{\text{BG}} \cdot \alpha_{\text{BG}} - C_{\text{BG}} \cdot \alpha_{\text{Mult}}}{\alpha_{\text{OUT}}} \quad \text{with } C = R \text{ or } G \text{ or } B$$

*Division is rounded to the nearest lower integer*

No configuration register is required by the blender. The blender use depends on the DMA2D operating mode defined by MODE[1:0] in DMA2D\_CR.

### 9.3.7 DMA2D output PFC

The output PFC performs the pixel format conversion from 32 bits to the output format defined by CM[2:0] in DMA2D\_OPFCCR.

The supported output formats are given in the table below.

**Table 40. Supported color mode in output**

CM[2:0]	Color mode
000	ARGB8888
001	RGB888
010	RGB565
011	ARGB1555
100	ARGB4444

### 9.3.8 DMA2D output FIFO

The output FIFO programs the pixels according to the color format defined in the output PFC.

The destination area is defined through a set of control registers:

- DMA2D output memory address register (DMA2D\_OMAR)
- DMA2D output offset register (DMA2D\_OOR)
- DMA2D number of lines register (number of lines and pixel per lines) (DMA2D\_NLR)

If the DMA2D operates in register-to-memory mode, the configured output rectangle is filled by the color specified in DMA2D\_OCOLR which contains a fixed 32-, 24-, or 16-bit value. The format is selected by CM[2:0] in DMA2D\_OPFCCR.

The data are stored into the memory in the order defined in the table below.

**Table 41. Data order in memory**

Color mode	@ + 3	@ + 2	@ + 1	@ + 0
ARGB8888	A <sub>0</sub> [7:0]	R <sub>0</sub> [7:0]	G <sub>0</sub> [7:0]	B <sub>0</sub> [7:0]
RGB888	B <sub>1</sub> [7:0]	R <sub>0</sub> [7:0]	G <sub>0</sub> [7:0]	B <sub>0</sub> [7:0]
	G <sub>2</sub> [7:0]	B <sub>2</sub> [7:0]	R <sub>1</sub> [7:0]	G <sub>1</sub> [7:0]
	R <sub>3</sub> [7:0]	G <sub>3</sub> [7:0]	B <sub>3</sub> [7:0]	R <sub>2</sub> [7:0]
	R <sub>1</sub> [4:0]G <sub>1</sub> [5:3]	G <sub>1</sub> [2:0]B <sub>1</sub> [4:0]	R <sub>0</sub> [4:0]G <sub>0</sub> [5:3]	G <sub>0</sub> [2:0]B <sub>0</sub> [4:0]
ARGB1555	A <sub>1</sub> [0]R <sub>1</sub> [4:0]G <sub>1</sub> [4:3]	G <sub>1</sub> [2:0]B <sub>1</sub> [4:0]	A <sub>0</sub> [0]R <sub>0</sub> [4:0]G <sub>0</sub> [4:3]	G <sub>0</sub> [2:0]B <sub>0</sub> [4:0]
ARGB4444	A <sub>1</sub> [3:0]R <sub>1</sub> [3:0]	G <sub>1</sub> [3:0]B <sub>1</sub> [3:0]	A <sub>0</sub> [3:0]R <sub>0</sub> [3:0]	G <sub>0</sub> [3:0]B <sub>0</sub> [3:0]

The RGB888 aligned on 32 bits is supported through the ARGB8888 mode.

### 9.3.9 DMA2D AHB master port timer

An 8-bit timer is embedded into the AHB master port to provide an optional limitation of the bandwidth on the crossbar.

This timer is clocked by the AHB clock and counts a dead time between two consecutive accesses. This limits the bandwidth availability.

The timer enabling and the dead time value are configured through the AHB master port timer configuration register (DMA2D\_AMPTCR).

### 9.3.10 DMA2D transactions

DMA2D transactions consist of a sequence of a given number of data transfers. The number of data and the width can be programmed by software.

Each DMA2D data transfer is composed of up to four steps:

1. Data loading from the memory location pointed by DMA2D\_FGMAR and pixel format conversion as defined in DMA2D\_FGCR
2. Data loading from a memory location pointed by DMA2D\_BGMAR and pixel format conversion as defined in DMA2D\_BGCR
3. Blending of all retrieved pixels according to the alpha channels resulting of the PFC operation on alpha values
4. Pixel format conversion of the resulting pixels according to DMA2D\_OCR and programming of the data to the memory location addressed through DMA2D\_OMAR

### 9.3.11 DMA2D configuration

Both source and destination data transfers can target peripherals and memories in the whole 4 Gbyte memory area, at addresses ranging between 0x0000 0000 and 0xFFFF FFFF.

The DMA2D can operate in any of the following modes selected through MODE[1:0] in DMA2D\_CR:

- Register-to-memory
- Memory-to-memory
- Memory-to-memory with PFC
- Memory-to-memory with PFC and blending

#### Register-to-memory

The register-to-memory mode is used to fill a user defined area with a predefined color.

The color format is set in DMA2D\_OPFCCR.

The DMA2D does not perform any data fetching from any source. It just writes the color defined in DMA2D\_OCOLR to the area located at the address pointed by DMA2D\_OMAR, and defined in DMA2D\_NLR and DMA2D\_OOR.

#### Memory-to-memory

In memory-to-memory mode, the DMA2D does not perform any graphical data transformation. The foreground input FIFO acts as a buffer and the data are transferred from the source memory location defined in DMA2D\_FGMAR to the destination memory location pointed by DMA2D\_OMAR.

The color mode programmed by CM[3:0] in DMA2D\_FGPCCR defines the number of bits per pixel for both input and output.

The size of the area to be transferred is defined by DMA2D\_NLR and DMA2D\_FGOR for the source, and by DMA2D\_NLR and DMA2D\_OOR for the destination.

### Memory-to-memory with PFC

In this mode, the DMA2D performs a pixel format conversion of the source data and stores them in the destination memory location.

The size of the areas to be transferred are defined by DMA2D\_NLR and DMA2D\_FGOR for the source, and by DMA2D\_NLR and DMA2D\_OOR for the destination.

Data are fetched from the location defined in DMA2D\_FGMAR and processed by the foreground PFC. The original pixel format is configured through DMA2D\_FGPFCCR.

If the original pixel format is direct color mode, then the color channels are all expanded to 8 bits.

If the pixel format is indirect color mode, the associated CLUT has to be loaded into the CLUT memory.

The CLUT loading can be done automatically by following the sequence below:

1. Set the CLUT address into DMA2D\_FGCMAR.
2. Set the CLUT size with CS[7:0] bits in DMA2D\_FGPFCCR.
3. Set the CLUT format (24 or 32 bits) with CCM in DMA2D\_FGPFCCR.
4. Start the CLUT loading by setting START in DMA2D\_FGPFCCR.

Once the CLUT loading is complete, the CTCIF flag in DMA2D\_IFR is raised, and an interrupt is generated if CTCIE = 1 in DMA2D\_CR. The automatic CLUT loading process can not work in parallel with classical DMA2D transfers.

The CLUT can also be filled by the CPU or by any other master through the APB port. The access to the CLUT is not possible when a DMA2D transfer is ongoing and uses the CLUT (indirect color format).

In parallel to the color conversion process, the alpha value is added or changed depending on the value programmed in DMA2D\_FGPFCCR. If the original image does not have an alpha channel, a default alpha value of 0xFF is automatically added to obtain a fully opaque pixel. The alpha value is modified according to AM[1:0] in DMA2D\_FGPFCCR:

- It can be unchanged.
- It can be replaced by the value defined by ALPHA[7:0] in DMA2D\_FGPFCCR.
- It can be replaced by the original value multiplied by ALPHA[7:0] divided by 255.

The resulting 32-bit data are encoded by the OUT PFC into the format specified by CM[2:0] in DMA2D\_OPFCCR. The output pixel format cannot be the indirect mode since no CLUT generation process is supported.

The processed data are written into the destination memory location pointed by DMA2D\_OMAR.

### Memory-to-memory with PFC and blending

In this mode, two sources are fetched in the foreground and background FIFOs from the memory locations defined by DMA2D\_FGMAR and DMA2D\_BGMAR.

The two pixel format converters have to be configured as described in the memory-to-memory mode. Their configurations can be different as each pixel format converter is independent and has its own CLUT memory.

Once each pixel has been converted into 32 bits by its respective PFC, all pixels are blended according to the equation below:

$$\text{with } \alpha_{\text{Mult}} = \frac{\alpha_{\text{FG}} \cdot \alpha_{\text{BG}}}{255}$$

$$\alpha_{\text{OUT}} = \alpha_{\text{FG}} + \alpha_{\text{BG}} - \alpha_{\text{Mult}}$$

$$C_{\text{OUT}} = \frac{C_{\text{FG}} \cdot \alpha_{\text{FG}} + C_{\text{BG}} \cdot \alpha_{\text{BG}} - C_{\text{BG}} \cdot \alpha_{\text{Mult}}}{\alpha_{\text{OUT}}} \quad \text{with } C = \text{R or G or B}$$

*Division are rounded to the nearest lower integer*

The resulting 32-bit pixel value is encoded by the output PFC according to the specified output format, and the data are written into the destination memory location pointed by DMA2D\_OMAR.

### Configuration error detection

The DMA2D checks that the configuration is correct before any transfer. The configuration error interrupt flag is set by hardware when a wrong configuration is detected when a new transfer/automatic loading starts. An interrupt is then generated if CEIE = 1 in DMA2D\_CR.

The wrong configurations that can be detected are listed below:

- Foreground CLUT automatic loading: MA bits in DMA2D\_FGCMAR are not aligned with CCM in DMA2D\_FGPCCR.
- Background CLUT automatic loading: MA bits in DMA2D\_BGCMAR are not aligned with CCM in DMA2D\_BGPCCR.
- Memory transfer (except in register-to-memory mode): MA bits in DMA2D\_FGMAR are not aligned with CM in DMA2D\_FGPCCR.
- Memory transfer (except in register-to-memory mode): CM bits in DMA2D\_FGPCCR are invalid
- Memory transfer (except in register-to-memory mode and except in memory-to-memory mode with blending and fixed color FG): PL bits in DMA2D\_NLR are odd while CM in DMA2D\_FGPCCR is A4 or L4.
- Memory transfer (except in register-to-memory mode): LO bits in DMA2D\_FGOR are odd while CM in DMA2D\_FGPCCR is A4 or L4.
- Memory transfer (only in blending mode): MA bits in DMA2D\_BGMAR are not aligned with CM in DMA2D\_BGPCCR.
- Memory transfer: (only in blending mode): CM bits in DMA2D\_BGPCCR are invalid
- Memory transfer (only in blending mode): PL bits in DMA2D\_NLR odd while CM in DMA2D\_BGPCCR is A4 or L4.
- Memory transfer (only in blending mode): LO bits in DMA2D\_BGOR are odd while CM in DMA2D\_BGPCCR is A4 or L4.
- Memory transfer (except in memory-to-memory mode): MA bits in DMA2D\_OMAR are not aligned with CM bits in DMA2D\_OPFCCR.
- Memory transfer (except in memory to memory mode): CM bits in DMA2D\_OPFCCR are invalid.

- Memory transfer: NL = 0x0 in DMA2D\_NLR
- Memory transfer: PL = 0x0 in DMA2D\_NLR

### 9.3.12 DMA2D transfer control (start, suspend, abort and completion)

Once the DMA2D is configured, the transfer can be launched by setting START in DMA2D\_CR. Once the transfer is completed, START is automatically reset and TCIF flag in DMA2D\_ISR is raised. An interrupt can be generated if TCIE is set in DMA2D\_CR.

The user application can suspend the DMA2D at any time by setting SUSP in DMA2D\_CR. The transaction can then be aborted by setting ABORT in DMA2D\_CR, or can be restarted by resetting SUSP in DMA2D\_CR.

The user application can abort at any time an ongoing transaction by setting ABORT in DMA2D\_CR. In this case, the TCIF flag is not raised.

Automatic CLUT transfers can also be aborted or suspended by using ABORT or SUSP in DMA2D\_CR.

### 9.3.13 Watermark

A watermark can be programmed to generate an interrupt when the last pixel of a given line has been written to the destination memory area.

The line number is defined by LW[15:0] in DMA2D\_LWR.

When the last pixel of this line has been transferred, the TWIF flag in DMA2D\_ISR is raised, and an interrupt is generated if TWIE is set in DMA2D\_CR.

### 9.3.14 Error management

Two kind of errors can be triggered:

- AHB master port errors signaled by TEIF in DMA2D\_ISR
- conflicts caused by a CLUT access (CPU trying to access the CLUT while a CLUT loading or a DMA2D transfer is ongoing) signaled by CAEIF in DMA2D\_ISR.

Both flags are associated to their own interrupt enable flag in DMA2D\_CR to generate an interrupt if need be (TEIE and CAEIE).

### 9.3.15 AHB dead time

To limit the AHB bandwidth use, a dead time between two consecutive AHB accesses can be programmed.

This feature can be enabled by setting EN in DMA2D\_AMTCR.

The dead time value is stored into DT[7:0] in DMA2D\_AMTCR. This value represents the guaranteed minimum number of cycles between two consecutive transactions on the AHB bus.

The update of the dead time value while the DMA2D runs is taken into account for the next AHB transfer.

## 9.4 DMA2D interrupts

An interrupt can be generated on the following events:

- Configuration error
- CLUT transfer complete
- CLUT access error
- Transfer watermark reached
- Transfer complete
- Transfer error

Separate interrupt enable bits are available for flexibility.

**Table 42. DMA2D interrupt requests**

Interrupt event	Event flag	Enable control bit
Configuration error	CEIF	CEIE
CLUT transfer complete	CTCIF	CTCIE
CLUT access error	CAEIF	CAEIE
Transfer watermark	TWF	TWIE
Transfer complete	TCIF	TCIE
Transfer error	TEIF	TEIE

## 9.5 DMA2D registers

### 9.5.1 DMA2D control register (DMA2D\_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MODE[1:0]
																rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	CEIE	CTCIE	CAEIE	TWIE	TCIE	TEIE	Res.	Res.	Res.	Res.	Res.	ABORT	SUSP	START	
		rw	rw	rw	rw	rw	rw						rs	rw	rs	

Bits 31:18 Reserved, must be kept at reset value.

Bits 17:16 **MODE[1:0]**: DMA2D mode

These bits are set and cleared by software. They cannot be modified while a transfer is ongoing.

00: Memory-to-memory (FG fetch only)

01: Memory-to-memory with PFC (FG fetch only with FG PFC active)

10: Memory-to-memory with blending (FG and BG fetch with PFC and blending)

11: Register-to-memory (no FG nor BG, only output stage active)

Bits 15:14 Reserved, must be kept at reset value.

Bit 13 **CEIE**: Configuration error interrupt enable

This bit is set and cleared by software.

0: CE interrupt disabled

1: CE interrupt enabled

Bit 12 **CTCIE**: CLUT transfer complete interrupt enable

This bit is set and cleared by software.

0: CTC interrupt disabled

1: CTC interrupt enabled

Bit 11 **CAEIE**: CLUT access error interrupt enable

This bit is set and cleared by software.

0: CAE interrupt disabled

1: CAE interrupt enabled

Bit 10 **TWIE**: Transfer watermark interrupt enable

This bit is set and cleared by software.

0: TW interrupt disabled

1: TW interrupt enabled

Bit 9 **TCIE**: Transfer complete interrupt enable

This bit is set and cleared by software.

0: TC interrupt disabled

1: TC interrupt enabled

Bit 8 **TEIE**: Transfer error interrupt enable

This bit is set and cleared by software.

0: TE interrupt disabled

1: TE interrupt enabled

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

Bit 2 **ABORT**: Abort

This bit can be used to abort the current transfer. This bit is set by software and is automatically reset by hardware when the START bit is reset.

0: No transfer abort requested

1: Transfer abort requested

Bit 1 **SUSP**: Suspend

This bit can be used to suspend the current transfer. This bit is set and reset by software. It is automatically reset by hardware when the START bit is reset.

0: Transfer not suspended

1: Transfer suspended

Bit 0 **START**: Start

This bit can be used to launch the DMA2D according to the parameters loaded in the various configuration registers. This bit is automatically reset by the following events:

- at the end of the transfer

- when the data transfer is aborted by the user application (setting ABORT in DMA2D\_CR)

- when a data transfer error occurs

- when the data transfer has not started due to a configuration error or another transfer operation already ongoing (automatic CLUT loading).

### 9.5.2 DMA2D interrupt status register (DMA2D\_ISR)

Address offset: 0x004

Reset value: 0x0000 0000

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

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

Bit 5 **CEIF**: Configuration error interrupt flag

This bit is set when START in DMA2D\_CR, DMA2DFGPCCR or DMA2D\_BGPCCR is set and a wrong configuration has been programmed.

Bit 4 **CTCIF**: CLUT transfer complete interrupt flag

This bit is set when the CLUT copy from a system memory area to the internal DMA2D memory is complete.

Bit 3 **CAEIF**: CLUT access error interrupt flag

This bit is set when the CPU accesses the CLUT while the CLUT is being automatically copied from a system memory to the internal DMA2D.

Bit 2 **TWIF**: Transfer watermark interrupt flag

This bit is set when the last pixel of the watermarked line has been transferred.

Bit 1 **TCIF**: Transfer complete interrupt flag

This bit is set when a DMA2D transfer operation is complete (data transfer only).

Bit 0 **TEIF**: Transfer error interrupt flag

This bit is set when an error occurs during a DMA transfer (data transfer or automatic CLUT loading).

### 9.5.3 DMA2D interrupt flag clear register (DMA2D\_IFCR)

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CCEIF	CCTCF	CAECIF	CTWIF	CTCIF	CTEIF									
										rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

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

Bit 5 **CCEIF**: Clear configuration error interrupt flag

Programming this bit to 1 clears CEIF in DMA2D\_ISR.

- Bit 4 **CCTCIF**: Clear CLUT transfer complete interrupt flag  
Programming this bit to 1 clears CTCIF in DMA2D\_ISR.
- Bit 3 **CAECIF**: Clear CLUT access error interrupt flag  
Programming this bit to 1 clears CAEIF in DMA2D\_ISR.
- Bit 2 **CTWIF**: Clear transfer watermark interrupt flag  
Programming this bit to 1 clears TWIF in DMA2D\_ISR.
- Bit 1 **CTCIF**: Clear transfer complete interrupt flag  
Programming this bit to 1 clears TCIF in DMA2D\_ISR.
- Bit 0 **CTEIF**: Clear Transfer error interrupt flag  
Programming this bit to 1 clears TEIF in DMA2D\_ISR.

### 9.5.4 DMA2D foreground memory address register (DMA2D\_FGMAR)

Address offset: 0x00C

Reset value: 0x0000 0000

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

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

This field contains the address of the data used for the foreground image. This register can only be written when data transfers are disabled. Once the data transfer has started, this register is read-only.

The address alignment must match the image format selected: for example, a 32-bit per pixel format must be 32-bit aligned, a 16-bit per pixel format must be 16-bit aligned and a 4-bit per pixel format must be 8-bit aligned.

### 9.5.5 DMA2D foreground offset register (DMA2D\_FGOR)

Address offset: 0x010

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	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.	LO[13:0]													
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 13:0 **LO[13:0]**: Line offset

Line offset used for the foreground expressed in pixel. This value is used to generate the address. It is added at the end of each line to determine the starting address of the next line. These bits can only be written when data transfers are disabled. Once a data transfer has started, they become read-only.

If the image format is 4-bit per pixel, the line offset must be even.

**9.5.6 DMA2D background memory address register (DMA2D\_BGMAR)**

Address offset: 0x014

Reset value: 0x0000 0000

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

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

This field contains the address of the data used for the background image. This register can only be written when data transfers are disabled. Once a data transfer has started, this register is read-only.

The address alignment must match the image format selected e.g. a 32-bit per pixel format must be 32-bit aligned, a 16-bit per pixel format must be 16-bit aligned and a 4-bit per pixel format must be 8-bit aligned.

**9.5.7 DMA2D background offset register (DMA2D\_BGOR)**

Address offset: 0x018

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.	LO[13:0]													
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 13:0 **LO[13:0]**: Line offset

This field contains the line offset used for the background image (expressed in pixel). This value is used for the address generation. It is added at the end of each line to determine the starting address of the next line.

These bits can only be written when data transfers are disabled. Once data transfer has started, they become read-only.

If the image format is 4-bit per pixel, the line offset must be even.

### 9.5.8 DMA2D foreground PFC control register (DMA2D\_FGPFCCR)

Address offset: 0x01C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
ALPHA[7:0]										Res.	Res.	Res.	Res.	Res.	AM[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	
CS[7:0]										Res.	Res.	START	CCM	CM[3:0]		
rw	rw	rw	rw	rw	rw	rw	rw			rs	rw	rw	rw	rw	rw	

Bits 31:24 **ALPHA[7:0]**: Alpha value

These bits define a fixed alpha channel value which can replace the original alpha value or be multiplied by the original alpha value according to the alpha mode selected through the AM[1:0] bits.

These bits can only be written when data transfers are disabled. Once a transfer has started, they become read-only.

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

Bits 17:16 **AM[1:0]**: Alpha mode

These bits select the alpha channel value to be used for the foreground image. They can only be written data the transfer are disabled. Once the transfer has started, they become read-only.

00: No modification of the foreground image alpha channel value

01: Replace original foreground image alpha channel value by ALPHA[7:0]

10: Replace original foreground image alpha channel value by ALPHA[7:0] multiplied with original alpha channel value

Other: Reserved

Bits 15:8 **CS[7:0]**: CLUT size

These bits define the size of the CLUT used for the foreground image. Once the CLUT transfer has started, this field is read-only.

The number of CLUT entries is equal to CS[7:0] + 1.

Bits 7:6 Reserved, must be kept at reset value.

Bit 5 **START**: Start

This bit can be set to start the automatic loading of the CLUT. It is automatically reset:

- at the end of the transfer
- when the transfer is aborted by the user application by setting ABORT in DMA2D\_CR
- when a transfer error occurs
- when the transfer has not started due to a configuration error or another transfer operation already ongoing (data transfer or automatic background CLUT transfer)

Bit 4 **CCM**: CLUT color mode

This bit defines the color format of the CLUT. It can only be written when the transfer is disabled. Once the CLUT transfer has started, this bit is read-only.

0: ARGB8888

1: RGB888

Other: Reserved

Bits 3:0 **CM[3:0]**: Color mode

These bits defines the color format of the foreground image. They can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

0000: ARGB8888  
 0001: RGB888  
 0010: RGB565  
 0011: ARGB1555  
 0100: ARGB4444  
 0101: L8  
 0110: AL44  
 0111: AL88  
 1000: L4  
 1001: A8  
 1010: A4  
 Other: Reserved

**9.5.9 DMA2D foreground color register (DMA2D\_FGCOLR)**

Address offset: 0x020

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.	RED[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GREEN[7:0]								BLUE[7:0]							
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 23:16 **RED[7:0]**: Red value

These bits define the Red value for the A4 or A8 mode of the foreground image. They can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

Bits 15:8 **GREEN[7:0]**: Green value

These bits defines the Green value for the A4 or A8 mode of the foreground image. They can only be written when data transfers are disabled. Once the transfer has started, They are read-only.

Bits 7:0 **BLUE[7:0]**: Blue value

These bits defines the Blue value for the A4 or A8 mode of the foreground image. They can only be written when data transfers are disabled. Once the transfer has started, They are read-only.

### 9.5.10 DMA2D background PFC control register (DMA2D\_BGPFCCR)

Address offset: 0x024

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
ALPHA[7:0]										Res.	Res.	Res.	Res.	Res.	AM[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	
CS[7:0]										Res.	Res.	START	CCM	CM[3:0]		
rw	rw	rw	rw	rw	rw	rw	rw			rs	rw	rw	rw	rw	rw	

Bits 31:24 **ALPHA[7:0]**: Alpha value

These bits define a fixed alpha channel value which can replace the original alpha value or be multiplied with the original alpha value according to the alpha mode selected with AM[1:0]. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

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

Bits 17:16 **AM[1:0]**: Alpha mode

These bits define which alpha channel value to be used for the background image. They can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

00: No modification of the foreground image alpha channel value

01: Replace original background image alpha channel value by ALPHA[7:0]

10: Replace original background image alpha channel value by ALPHA[7:0] multiplied with original alpha channel value

Other: Reserved

Bits 15:8 **CS[7:0]**: CLUT size

These bits define the size of the CLUT used for the BG. Once the CLUT transfer has started, this field is read-only.

The number of CLUT entries is equal to CS[7:0] + 1.

Bits 7:6 Reserved, must be kept at reset value.

Bit 5 **START**: Start

This bit is set to start the automatic loading of the CLUT. This bit is automatically reset:

- at the end of the transfer
- when the transfer is aborted by the user application by setting ABORT in DMA2D\_CR
- when a transfer error occurs
- when the transfer has not started due to a configuration error or another transfer operation already ongoing (data transfer or automatic foreground CLUT transfer).

Bit 4 **CCM**: CLUT color mode

These bits define the color format of the CLUT. This register can only be written when the transfer is disabled. Once the CLUT transfer has started, this bit is read-only.

0: ARGB8888

1: RGB888

Other: Reserved

Bits 3:0 **CM[3:0]**: Color mode

These bits define the color format of the foreground image. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

0000: ARGB8888  
 0001: RGB888  
 0010: RGB565  
 0011: ARGB1555  
 0100: ARGB4444  
 0101: L8  
 0110: AL44  
 0111: AL88  
 1000: L4  
 1001: A8  
 1010: A4  
 Other: Reserved

**9.5.11 DMA2D background color register (DMA2D\_BGCOLR)**

Address offset: 0x028

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RED[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GREEN[7:0]								BLUE[7:0]							
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 23:16 **RED[7:0]**: Red value

These bits define the Red value for the A4 or A8 mode of the background. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

Bits 15:8 **GREEN[7:0]**: Green value

These bits define the green value for the A4 or A8 mode of the background. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

Bits 7:0 **BLUE[7:0]**: Blue value

These bits define the blue value for the A4 or A8 mode of the background. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

### 9.5.12 DMA2D foreground CLUT memory address register (DMA2D\_FGCMAR)

Address offset: 0x002C

Reset value: 0x0000 0000

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

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

This field contains the address of the data used for the CLUT address dedicated to the foreground image. This register can only be written when no transfer is ongoing. Once the CLUT transfer has started, this register is read-only.

If the foreground CLUT format is 32-bit, the address must be 32-bit aligned.

### 9.5.13 DMA2D background CLUT memory address register (DMA2D\_BGCMAR)

Address offset: 0x0030

Reset value: 0x0000 0000

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

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

This field contains the address of the data used for the CLUT address dedicated to the background image. This register can only be written when no transfer is ongoing. Once the CLUT transfer has started, this register is read-only.

If the background CLUT format is 32-bit, the address must be 32-bit aligned.

### 9.5.14 DMA2D output PFC control register (DMA2D\_OPFCCR)

Address offset: 0x034

Reset value: 0x0000 0000

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

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

Bits 19:3 Reserved, must be kept at reset value.

Bits 2:0 **CM[2:0]**: Color mode

These bits define the color format of the output image. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

000: ARGB8888

001: RGB888

010: RGB565

011: ARGB1555

100: ARGB4444

Other: Reserved

### 9.5.15 DMA2D output color register (DMA2D\_OCOLR)

Address offset: 0x038

Reset value: 0x0000 0000

The same register is used to show the color values, with different formats depending on the color mode.

This register can only be written when transfers are disabled. Once the CLUT transfer started, this register is read-only.

ARGB8888 or RGB888 color mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ALPHA[7:0]								RED[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
GREEN[7:0]								BLUE[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **ALPHA[7:0]**: Alpha channel value of the output color in ARGB8888 mode (otherwise reserved)

Bits 23:16 **RED[7:0]**: Red value of the output image in ARGB8888 or RGB888 mode

Bits 15:8 **GREEN[7:0]**: Green value of the output image in ARGB8888 or RGB888

Bits 7:0 **BLUE[7:0]**: Blue value of the output image in ARGB8888 or RGB888

### 9.5.16 DMA2D output color register [alternate] (DMA2D\_OCOLR)

Address offset: 0x038

Reset value: 0x0000 0000

The same register is used to show the color values, with different formats depending on the color mode.

This register can only be written when transfers are disabled. Once the CLUT transfer started, this register is read-only.

RGB565 color 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
RED[4:0]						GREEN[5:0]						BLUE[4:0]			
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 15:11 **RED[4:0]**: Red value of the output image in RGB565 mode

Bits 10:5 **GREEN[5:0]**: Green value of the output image in RGB565 mode

Bits 4:0 **BLUE[4:0]**: Blue value of the output image in RGB565 mode

### 9.5.17 DMA2D output color register [alternate] (DMA2D\_OCOLR)

Address offset: 0x038

Reset value: 0x0000 0000

The same register is used to show the color values, with different formats depending on the color mode.

This register can only be written when transfers are disabled. Once the CLUT transfer started, this register is read-only.

ARGB1555 color 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	
A	RED[4:0]						GREEN[4:0]						BLUE[4:0]			
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.

Bit 15 **A**: Alpha channel value of the output color in ARGB1555 mode

Bits 14:10 **RED[4:0]**: Red value of the output image in ARGB1555 mode

Bits 9:5 **GREEN[4:0]**: Green value of the output image in ARGB1555 mode

Bits 4:0 **BLUE[4:0]**: Blue value of the output image in ARGB1555 mode

### 9.5.18 DMA2D output color register [alternate] (DMA2D\_OCOLR)

Address offset: 0x038

Reset value: 0x0000 0000

The same register is used to show the color values, with different formats depending on the color mode.

This register can only be written when transfers are disabled. Once the CLUT transfer started, this register is read-only.

ARGB4444 color 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
ALPHA[3:0]				RED[3:0]				GREEN[3:0]				BLUE[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 15:12 **ALPHA[3:0]**: Alpha channel of the output color value in ARGB4444

Bits 11:8 **RED[3:0]**: Red value of the output image in ARGB4444 mode

Bits 7:4 **GREEN[3:0]**: Green value of the output image in ARGB4444 mode

Bits 3:0 **BLUE[3:0]**: Blue value of the output image in ARGB4444 mode

### 9.5.19 DMA2D output memory address register (DMA2D\_OMAR)

Address offset: 0x003C

Reset value: 0x0000 0000

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

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

This field contains the address of the data used for the output FIFO. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only. The address alignment must match the image format selected e.g. a 32-bit per pixel format must be 32-bit aligned and a 16-bit per pixel format must be 16-bit aligned.

### 9.5.20 DMA2D output offset register (DMA2D\_OOR)

Address offset: 0x040

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
LO[13:0]															
Res.	Res.	rw													

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

Bits 13:0 **LO[13:0]**: Line offset

This field contains the line offset used for the output (expressed in pixels). This value is used for the address generation. It is added at the end of each line to determine the starting address of the next line. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

### 9.5.21 DMA2D number of line register (DMA2D\_NLR)

Address offset: 0x044

Reset value: 0x0000 0000

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

Number of pixels per lines of the area to be transferred. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

If any of the input image format is 4-bit per pixel, pixel per lines must be even.

Bits 15:0 **NL[15:0]**: Number of lines

Number of lines of the area to be transferred. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

### 9.5.22 DMA2D line watermark register (DMA2D\_LWR)

Address offset: 0x048

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
LW[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 **LW[15:0]**: Line watermark

These bits allow the configuration of the line watermark for interrupt generation.

An interrupt is raised when the last pixel of the watermarked line has been transferred. These bits can only be written when data transfers are disabled. Once the transfer has started, they are read-only.

### 9.5.23 DMA2D AHB master timer configuration register (DMA2D\_AMTCR)

Address offset: 0x04C

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
DT[7:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw							rw

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

Bits 15:8 **DT[7:0]**: Dead time

Dead time value in the AHB clock cycle inserted between two consecutive accesses on the AHB master port. These bits represent the minimum guaranteed number of cycles between two consecutive AHB accesses.

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

Bit 0 **EN**: Enable

Enables the dead time functionality.

### 9.5.24 DMA2D foreground CLUT (DMA2D\_FGCLUTx)

Address offset: 0x400 + 0x4 \* x, (x = 0 to 255)

Reset value: 0xXXXX XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ALPHA[7:0]								RED[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
GREEN[7:0]								BLUE[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **ALPHA[7:0]**: Alpha

Alpha value for index {x} for the foreground

Bits 23:16 **RED[7:0]**: Red

Red value for index {x} for the foreground

Bits 15:8 **GREEN[7:0]**: Green

Green value for index {x} for the foreground

Bits 7:0 **BLUE[7:0]**: Blue

Blue value for index {x} for the foreground

### 9.5.25 DMA2D background CLUT (DMA2D\_BGCLUTx)

Address offset: 0x800 + 0x4 \* x, (x = 0 to 255)

Reset value: 0xXXXX XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ALPHA[7:0]								RED[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
GREEN[7:0]								BLUE[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **ALPHA[7:0]**: Alpha

Alpha value for index {x} for the background

Bits 23:16 **RED[7:0]**: Red

Red value for index {x} for the background

Bits 15:8 **GREEN[7:0]**: Green

Green value for index {x} for the background

Bits 7:0 **BLUE[7:0]**: Blue

Blue value for index {x} for the background

### 9.5.26 DMA2D register map

**Table 43. DMA2D register map and reset values**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	DMA2D_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value																																
0x004	DMA2D_ISR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value																																
0x008	DMA2D_IFCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value																																
0x00C	DMA2D_FGMAR	MA[31:0]												LO[13:0]												000000							
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x010	DMA2D_FGOR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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																																
0x014	DMA2D_BGMAR	MA[31:0]												LO[13:0]												000000							
	Reset value	0	0	0	0	0	0	0	0	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	DMA2D_BGOR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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																																
0x01C	DMA2D_FGPCCR	ALPHA[7:0]												AM [1:0]												000000							
	Reset value	0	0	0	0	0	0	0	0	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	DMA2D_FGCOLR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RED[7:0]												000000							
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x024	DMA2D_BGPCCR	ALPHA[7:0]												AM [1:0]												000000							
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x028	DMA2D_BGCOLR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RED[7:0]												000000							
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x02C	DMA2D_FGCMAR	MA[31:0]												GREEN[7:0]												000000							
	Reset value	0	0	0	0	0	0	0	0	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	DMA2D_BGCMAR	MA[31:0]												BLUE[7:0]												000000							
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x034	DMA2D_OPFCCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GREEN[7:0]												000000							
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x038	DMA2D_OCOLR ARGB8888 or RGB888 color mode	ALPHA[7:0]												RED[7:0]												000000							
	Reset value	0	0	0	0	0	0	0	0	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	DMA2D_OCOLR RGB565 color mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GREEN[6:0]												000000							
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x038	DMA2D_OCOLR ARGB1555 color mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BLUE[4:0]												000000							
	Reset value													A	RED[4:0]												000000						

**Table 43. DMA2D register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x038	DMA2D_OCOLR ARGB4444 color mode	Res.																																
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x03C	DMA2D_OMAR																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x040	DMA2D_OOR	Res.																																
	Reset value																																	
0x044	DMA2D_NLR	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		
0x048	DMA2D_LWR	Res.																																
	Reset value																																	
0x04C	DMA2D_AMTCR	Res.	EN																															
	Reset value																																0	
0x050-0x3FC	Reserved																																	
0x400 + 0x4 * x, (x=0 to 255)	DMA2D_FGCLUTx																																	
	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			
0x800 + 0x4 * x, (x=0 to 255)	DMA2D_BGCLUTx																																	
	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			

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

## 10 Nested vectored interrupt controller (NVIC)

### 10.1 NVIC features

The nested vector interrupt controller NVIC includes the following features:

- up to 98 maskable interrupt channels for STM32F75xxx and STM32F74xxx (not including the 16 interrupt lines of Cortex®-M7 with FPU)
- 16 programmable priority levels (4 bits of interrupt priority are used)
- low-latency exception and interrupt handling
- power management control
- implementation of system control registers

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

All interrupts including the core exceptions are managed by the NVIC. For more information on exceptions and NVIC programming, refer to programming manual PMxxxx.

#### 10.1.1 SysTick calibration value register

The SysTick calibration value is fixed to 18750, which gives a reference time base of 1 ms with the SysTick clock set to 18.75 MHz (HCLK/8, with HCLK set to 150 MHz).

#### 10.1.2 Interrupt and exception vectors

See [Table 44](#), for the vector table for the STM32F75xxx and STM32F74xxx devices.

**Table 44. STM32F75xxx and STM32F74xxx vector table**

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	fixed	Reset	Reset	0x0000 0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-	-1	fixed	HardFault	All class of fault	0x0000 000C
-	0	settable	MemManage	Memory management	0x0000 0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	-	Reserved	0x0000 001C - 0x0000 002B

**Table 44. STM32F75xxx and STM32F74xxx vector table (continued)**

<b>Position</b>	<b>Priority</b>	<b>Type of priority</b>	<b>Acronym</b>	<b>Description</b>	<b>Address</b>
-	3	settable	SVCall	System service call via SWI instruction	0x0000 002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000 0030
-	-	-	-	Reserved	0x0000 0034
-	5	settable	PendSV	Pendable request for system service	0x0000 0038
-	6	settable	SysTick	System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	PVD	PVD through EXTI line detection interrupt	0x0000 0044
2	9	settable	TAMP_STAMP	Tamper andTimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	0x0000 004C
4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068
11	18	settable	DMA1_Stream0	DMA1 Stream0 global interrupt	0x0000 006C
12	19	settable	DMA1_Stream1	DMA1 Stream1 global interrupt	0x0000 0070
13	20	settable	DMA1_Stream2	DMA1 Stream2 global interrupt	0x0000 0074
14	21	settable	DMA1_Stream3	DMA1 Stream3 global interrupt	0x0000 0078
15	22	settable	DMA1_Stream4	DMA1 Stream4 global interrupt	0x0000 007C
16	23	settable	DMA1_Stream5	DMA1 Stream5 global interrupt	0x0000 0080
17	24	settable	DMA1_Stream6	DMA1 Stream6 global interrupt	0x0000 0084
18	25	settable	ADC	ADC1, ADC2 and ADC3 global interrupts	0x0000 0088
19	26	settable	CAN1_TX	CAN1 TX interrupts	0x0000 008C
20	27	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000 0090
21	28	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000 0094

**Table 44. STM32F75xxx and STM32F74xxx vector table (continued)**

<b>Position</b>	<b>Priority</b>	<b>Type of priority</b>	<b>Acronym</b>	<b>Description</b>	<b>Address</b>
22	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000 0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C
24	31	settable	TIM1_BRK_TIM9	TIM1 Break interrupt and TIM9 global interrupt	0x0000 00A0
25	32	settable	TIM1_UP_TIM10	TIM1 Update interrupt and TIM10 global interrupt	0x0000 00A4
26	33	settable	TIM1_TRG_COM_TIM11	TIM1 Trigger and Commutation interrupts and TIM11 global interrupt	0x0000 00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000 00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000 00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000 00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000 00B8
31	38	settable	I2C1_EV	I <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

**Table 44. STM32F75xxx and STM32F74xxx vector table (continued)**

<b>Position</b>	<b>Priority</b>	<b>Type of priority</b>	<b>Acronym</b>	<b>Description</b>	<b>Address</b>
47	54	settable	DMA1_Stream7	DMA1 Stream7 global interrupt	0x0000 00FC
48	55	settable	FSMC	FSMC global interrupt	0x0000 0100
49	56	settable	SDMMC1	SDMMC1 global interrupt	0x0000 0104
50	57	settable	TIM5	TIM5 global interrupt	0x0000 0108
51	58	settable	SPI3	SPI3 global interrupt	0x0000 010C
52	59	settable	UART4	UART4 global interrupt	0x0000 0110
53	60	settable	UART5	UART5 global interrupt	0x0000 0114
54	61	settable	TIM6_DAC	TIM6 global interrupt, DAC1 and DAC2 underrun error interrupts	0x0000 0118
55	62	settable	TIM7	TIM7 global interrupt	0x0000 011C
56	63	settable	DMA2_Stream0	DMA2 Stream0 global interrupt	0x0000 0120
57	64	settable	DMA2_Stream1	DMA2 Stream1 global interrupt	0x0000 0124
58	65	settable	DMA2_Stream2	DMA2 Stream2 global interrupt	0x0000 0128
59	66	settable	DMA2_Stream3	DMA2 Stream3 global interrupt	0x0000 012C
60	67	settable	DMA2_Stream4	DMA2 Stream4 global interrupt	0x0000 0130
61	68	settable	ETH	Ethernet global interrupt	0x0000 0134
62	69	settable	ETH_WKUP	Ethernet Wakeup through EXTI line interrupt	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

**Table 44. STM32F75xxx and STM32F74xxx vector table (continued)**

<b>Position</b>	<b>Priority</b>	<b>Type of priority</b>	<b>Acronym</b>	<b>Description</b>	<b>Address</b>
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	settable	CRYP	CRYP crypto global interrupt	0x0000 017C
80	87	settable	HASH_RNG	Hash and Rng global interrupt	0x0000 0180
81	88	settable	FPU	FPU global interrupt	0x0000 0184
82	89	settable	UART7	UART7 global interrupt	0x0000 0188
83	90	settable	UART8	UART8 global interrupt	0x0000 018C
84	91	settable	SPI4	SPI4 global interrupt	0x0000 0190
85	92	settable	SPI5	SPI5 global interrupt	0x0000 0194
86	93	settable	SPI6	SPI6 global interrupt	0x0000 0198
87	94	settable	SAI1	SAI1 global interrupt	0x0000 019C
88	95	settable	LCD-TFT	LCD-TFT global interrupt	0x0000 01A0
89	96	settable	LCD-TFT	LCD-TFT global Error interrupt	0x0000 01A4
90	97	settable	DMA2D	DMA2D global interrupt	0x0000 01A8
91	98	settable	SAI2	SAI2 global interrupt	0x0000 01AC
92	99	settable	QuadSPI	QuadSPI global interrupt	0x0000 01B0
93	100	settable	LP Timer1	LP Timer1 global interrupt	0x0000 01B4
94	101	settable	HDMI-CEC	HDMI-CEC global interrupt	0x0000 01B8
95	102	settable	I2C4_EV	I2C4 event interrupt	0x0000 01BC
96	103	settable	I2C4_ER	I2C4 Error interrupt	0x0000 01C0
97	104	settable	SPDIFRX	SPDIFRX global interrupt	0x0000 01C4

## 11 Extended interrupts and events controller (EXTI)

The external interrupt/event controller consists of up to 24 edge detectors for generating event/interrupt requests. Each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently. A pending register maintains the status line of the interrupt requests.

### 11.1 EXTI main features

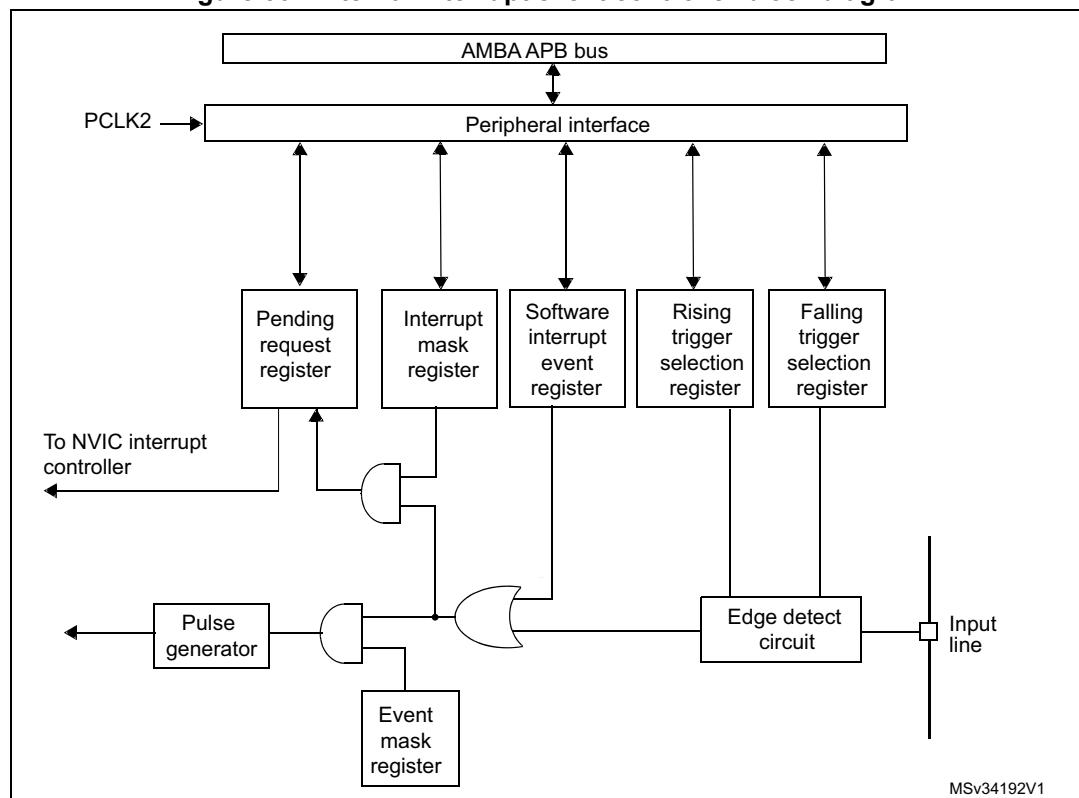
The main features of the EXTI controller are the following:

- independent trigger and mask on each interrupt/event line
- dedicated status bit for each interrupt line
- generation of up to 24 software event/interrupt requests
- detection of external signals with a pulse width lower than the APB2 clock period. Refer to the electrical characteristics section of the STM32F75xxx and STM32F74xxx datasheets for details on this parameter.

### 11.2 EXTI block diagram

*Figure 30* shows the block diagram.

**Figure 30. External interrupt/event controller block diagram**



## 11.3 Wakeup event management

The STM32F75xxx and STM32F74xxx devices are able to handle external or internal events in order to wake up the core (WFE). The wakeup event can be generated either by:

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

To use an external line as a wakeup event, refer to [Section 11.4: Functional description](#).

## 11.4 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.

## 11.5 Hardware interrupt selection

To configure a line as interrupt sources, use the following procedure:

1. Configure the corresponding mask bit (EXTI\_IMR)
2. Configure the Trigger selection bits of the interrupt lines (EXTI\_RTSR and EXTI\_FTSR)
3. Configure the enable and mask bits that control the NVIC IRQ channel mapped to the external interrupt controller (EXTI) so that an interrupt coming from one of the 24 lines can be correctly acknowledged.

## 11.6 Hardware event selection

To configure a line as event sources, use the following procedure:

1. Configure the corresponding mask bit (EXTI\_EMR)
2. Configure the Trigger selection bits of the event line (EXTI\_RTSR and EXTI\_FTSR)

## 11.7 Software interrupt/event selection

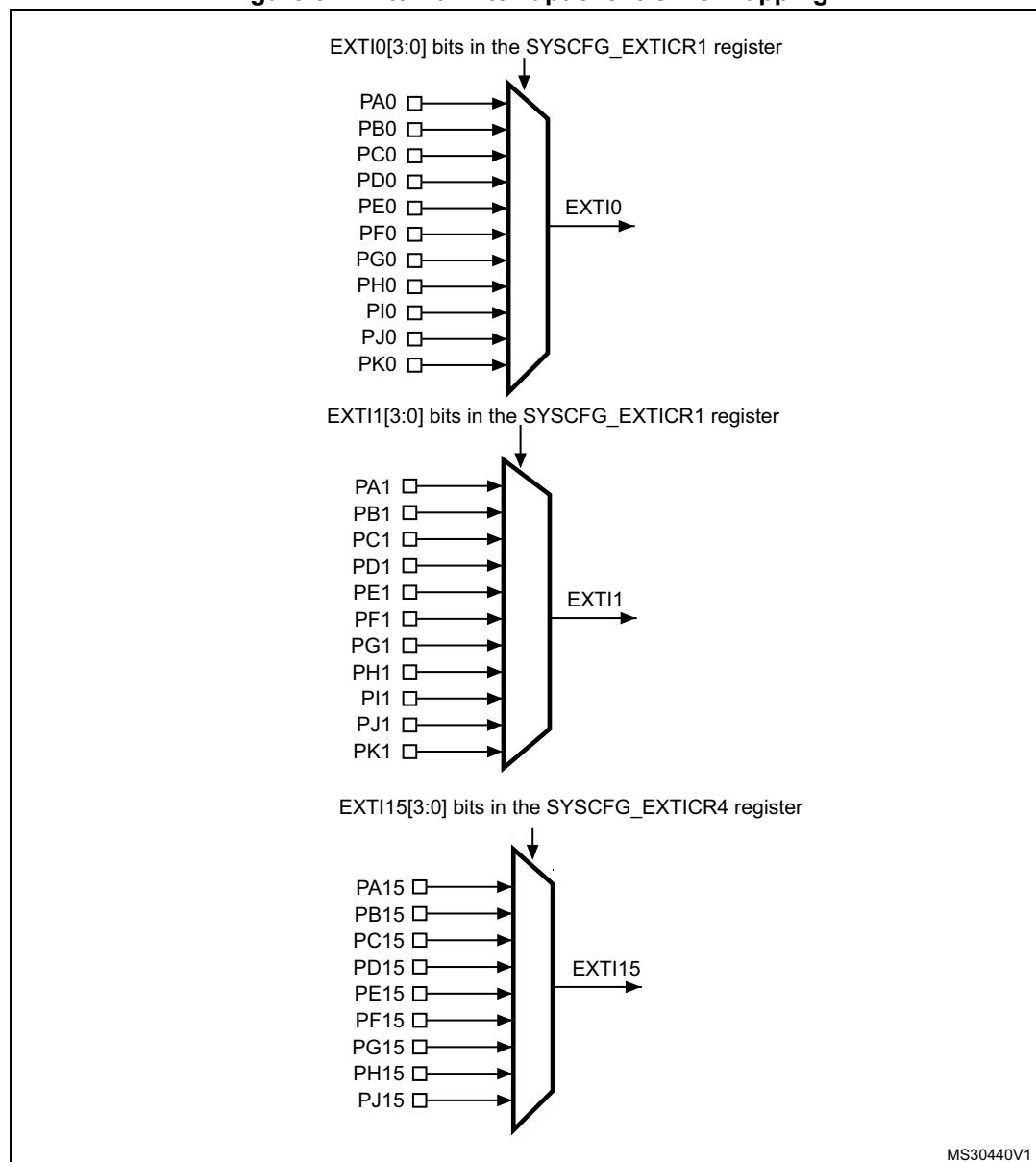
The line can be configured as software interrupt/event line. The following is the procedure to generate a software interrupt.

1. Configure the corresponding mask bit (EXTI\_IMR, EXTI\_EMR)
2. Set the required bit in the software interrupt register (EXTI\_SWIER)

## 11.8 External interrupt/event line mapping

Up to 168 GPIOs are connected to the 16 external interrupt/event lines in the following manner:

**Figure 31. External interrupt/event GPIO mapping**



MS30440V1

The eight other EXTI lines are connected as follows:

- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the USB OTG FS Wakeup event
- EXTI line 19 is connected to the Ethernet 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
- EXTI line 23 is connected to the LPTIM1 asynchronous event

## 11.9 EXTI registers

Refer to [Section 1.2 on page 63](#) for a list of abbreviations used in register descriptions.

### 11.9.1 Interrupt mask register (EXTI\_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16							
								rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
rw															

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

Bits 23:0 **IMx**: Interrupt mask on line x

- 0: Interrupt request from line x is masked
- 1: Interrupt request from line x is not masked

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

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	EM23	EM22	EM21	EM20	EM19	EM18	EM17	EM16							
								rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM15	EM14	EM13	EM12	EM11	EM10	EM9	EM8	EM7	EM6	EM5	EM4	EM3	EM2	EM1	EM0
rw															

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

Bits 23:0 **EMx**: Event mask on line x

- 0: Event request from line x is masked
- 1: Event request from line x is not masked

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

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TR23	TR22	TR21	TR20	TR19	TR18	TR17	TR16							
								rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw															

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

Bits 23:0 **TRx**: Rising trigger event configuration bit of line x

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

**Note:** *The external wakeup lines are edge triggered, no glitch must be generated on these lines. If a rising edge occurs on the external interrupt line while writing to the EXTI\_RTSR register, the pending bit is set. Rising and falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.*

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

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TR23	TR22	TR21	TR20	TR19	TR18	TR17	TR16							
								rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw															

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

Bits 23:0 **TRx**: Falling trigger event configuration bit of line x

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

**Note:** The external wakeup lines are edge triggered, no glitch must be generated on these lines. If a falling edge occurs on the external interrupt line while writing to the EXTI\_FTSR register, the pending bit is not set.

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

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

Address offset: 0x10

Reset value: 0x0000 0000

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

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

Bits 23:0 **SWIERx:** Software Interrupt on line x

If interrupt are enabled on line x in the EXTI\_IMR register, writing '1' to SWIERx 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).

### 11.9.6 Pending register (EXTI\_PR)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	PR23	PR22	PR21	PR20	PR19	PR18	PR17	PR16							
								rc_w1							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1															

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

Bits 23:0 **PRx:** Pending bit

0: No trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line.

This bit is cleared by programming it to '1'.

### 11.9.7 EXTI register map

[Table 45](#) gives the EXTI register map and the reset values.

**Table 45. External interrupt/event controller register map and reset values**

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 12 Cyclic redundancy check calculation unit (CRC)

### 12.1 Introduction

The CRC (cyclic redundancy check) calculation unit is used to get a CRC code from 8-, 16- or 32-bit data word and a generator polynomial.

Among other applications, CRC-based techniques are used to verify data transmission or storage integrity. In the scope of the functional safety standards, they offer a means of verifying the flash memory integrity. The CRC calculation unit helps compute a signature of the software during runtime, to be compared with a reference signature generated at link time and stored at a given memory location.

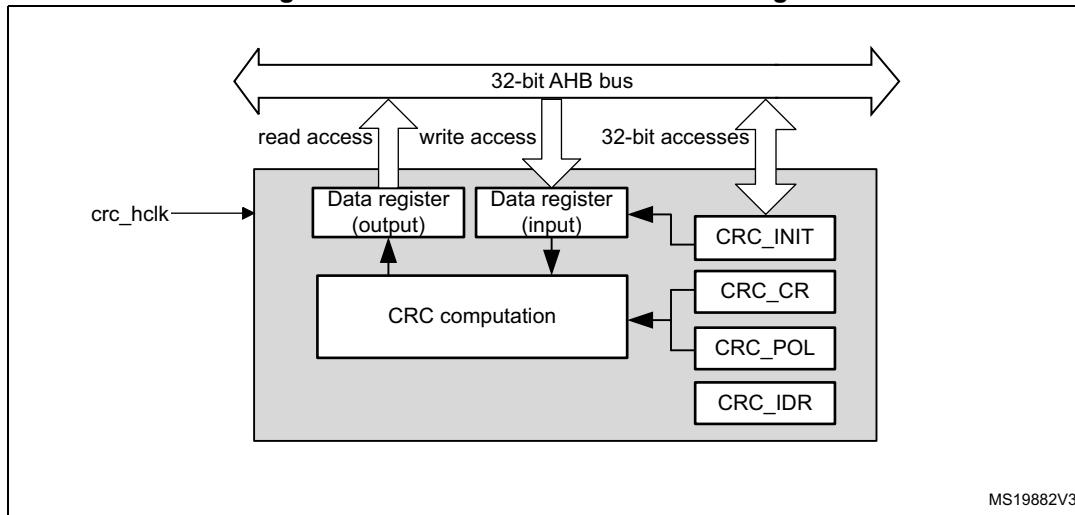
### 12.2 CRC main features

- Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7
$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$
- Alternatively, uses fully programmable polynomial with programmable size (7, 8, 16, 32 bits)
- Handles 8-, 16-, 32-bit data size
- Programmable CRC initial value
- Single input/output 32-bit data register
- Input buffer to avoid bus stall during calculation
- CRC computation done in 4 AHB clock cycles (HCLK) for the 32-bit data size
- General-purpose 8-bit register (can be used for temporary storage)
- Reversibility option on I/O data
- Accessed through AHB slave peripheral by 32-bit words only, with the exception of CRC\_DR register that can be accessed by words, right-aligned half-words and right-aligned bytes

## 12.3 CRC functional description

### 12.3.1 CRC block diagram

Figure 32. CRC calculation unit block diagram



### 12.3.2 CRC internal signals

Table 46. CRC internal input/output signals

Signal name	Signal type	Description
crc_hclk	Digital input	AHB clock

### 12.3.3 CRC operation

The CRC calculation unit has a single 32-bit read/write data register (CRC\_DR). It is used to input new data (write access), and holds the result of the previous CRC calculation (read access).

Each write operation to the data register creates a combination of the previous CRC value (stored in CRC\_DR) and the new one. CRC computation is done on the whole 32-bit data word or byte by byte depending on the format of the data being written.

The CRC\_DR register can be accessed by word, right-aligned half-word and right-aligned byte. For the other registers only 32-bit accesses are allowed.

The duration of the computation depends on data width:

- 4 AHB clock cycles for 32 bits
- 2 AHB clock cycles for 16 bits
- 1 AHB clock cycles for 8 bits

An input buffer allows a second data to be immediately written without waiting for any wait-states due to the previous CRC calculation.

The data size can be dynamically adjusted to minimize the number of write accesses for a given number of bytes. For instance, a CRC for 5 bytes can be computed with a word write followed by a byte write.

The input data can be reversed to manage the various endianness schemes. The reversing operation can be performed on 8 bits, 16 bits and 32 bits depending on the REV\_IN[1:0] bits in the CRC\_CR register.

For example, 0x1A2B3C4D input data are used for CRC calculation as:

- 0x58D43CB2 with bit-reversal done by byte
- 0xD458B23C with bit-reversal done by half-word
- 0xB23CD458 with bit-reversal done on the full word

The output data can also be reversed by setting the REV\_OUT bit in the CRC\_CR register.

The operation is done at bit level. For example, 0x11223344 output data are converted to 0x22CC4488.

The CRC calculator can be initialized to a programmable value using the RESET control bit in the CRC\_CR register (the default value is 0xFFFFFFFF).

The initial CRC value can be programmed with the CRC\_INIT register. The CRC\_DR register is automatically initialized upon CRC\_INIT register write access.

The CRC\_IDR register can be used to hold a temporary value related to CRC calculation. It is not affected by the RESET bit in the CRC\_CR register.

## Polynomial programmability

The polynomial coefficients are fully programmable through the CRC\_POL register, and the polynomial size can be configured to be 7, 8, 16 or 32 bits by programming the POLYSIZE[1:0] bits in the CRC\_CR register. Even polynomials are not supported.

*Note:* The type of an even polynomial is  $X+X^2+\dots+X^n$ , while the type of an odd polynomial is  $1+X+X^2+\dots+X^n$ .

If the CRC data is less than 32-bit, its value can be read from the least significant bits of the CRC\_DR register.

To obtain a reliable CRC calculation, the change on-fly of the polynomial value or size can not be performed during a CRC calculation. As a result, if a CRC calculation is ongoing, the application must either reset it or perform a CRC\_DR read before changing the polynomial.

The default polynomial value is the CRC-32 (Ethernet) polynomial: 0x4C11DB7.

## 12.4 CRC registers

The CRC\_DR register can be accessed by words, right-aligned half-words and right-aligned bytes. For the other registers only 32-bit accesses are allowed.

### 12.4.1 CRC data register (CRC\_DR)

Address offset: 0x00

Reset value: 0xFFFF FFFF

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

Bits 31:0 **DR[31:0]**: Data register bits

This register is used to write new data to the CRC calculator.

It holds the previous CRC calculation result when it is read.

If the data size is less than 32 bits, the least significant bits are used to write/read the correct value.

### 12.4.2 CRC independent data register (CRC\_IDR)

Address offset: 0x04

Reset value: 0x0000 0000

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

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

Bits 7:0 **IDR[7:0]**: General-purpose 8-bit data register bits

These bits can be used as a temporary storage location for one byte.

This register is not affected by CRC resets generated by the RESET bit in the CRC\_CR register

### 12.4.3 CRC control register (CRC\_CR)

Address offset: 0x08

Reset value: 0x0000 0000

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

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

Bit 7 **REV\_OUT**: Reverse output data

This bit controls the reversal of the bit order of the output data.

0: Bit order not affected

1: Bit-reversed output format

Bits 6:5 **REV\_IN[1:0]**: Reverse input data

This bitfield controls the reversal of the bit order of the input data

00: Bit order not affected

01: Bit reversal done by byte

10: Bit reversal done by half-word

11: Bit reversal done by word

Bits 4:3 **POLYSIZE[1:0]**: Polynomial size

These bits control the size of the polynomial.

00: 32 bit polynomial

01: 16 bit polynomial

10: 8 bit polynomial

11: 7 bit polynomial

Bits 2:1 Reserved, must be kept at reset value.

Bit 0 **RESET**: RESET bit

This bit is set by software to reset the CRC calculation unit and set the data register to the value stored in the CRC\_INIT register. This bit can only be set, it is automatically cleared by hardware

### 12.4.4 CRC initial value (CRC\_INIT)

Address offset: 0x10

Reset value: 0xFFFF FFFF

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

Bits 31:0 **CRC\_INIT[31:0]**: Programmable initial CRC value

This register is used to write the CRC initial value.

### 12.4.5 CRC polynomial (CRC\_POL)

Address offset: 0x14

Reset value: 0x04C1 1DB7

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

Bits 31:0 **POL[31:0]**: Programmable polynomial

This register is used to write the coefficients of the polynomial to be used for CRC calculation.

If the polynomial size is less than 32 bits, the least significant bits have to be used to program the correct value.

## 12.4.6 CRC register map

**Table 47. CRC register map and reset values**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	CRC_DR																																
	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	1	1		
0x04	CRC_IDR	Res.																															
	Reset value																																
0x08	CRC_CR	Res.																															
	Reset value																																
0x10	CRC_INIT																																
	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	1	1	1	
0x14	CRC_POL																																
	Reset value	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	1	0	1	1	0	1	1	1	1	

Refer to [Section 2.2: Memory organization](#) for the register boundary addresses.

## 13 Flexible memory controller (FMC)

### 13.1 Introduction

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

### 13.2 FMC main features

The FMC functional block makes the interface with: synchronous and asynchronous static memories, SDRAM memories, and NAND flash memory. Its main purposes are:

- to translate AHB transactions into the appropriate external device protocol
- to meet the access time requirements of the external memory devices

All external memories share the addresses, data and control signals with the controller. Each external device is accessed by means of a unique chip select. The FMC performs only one access at a time to an external device.

The main features of the FMC controller are the following:

- Interface with static-memory mapped devices including:
  - Static random access memory (SRAM)
  - NOR flash memory/OneNAND flash memory
  - PSRAM (4 memory banks)
  - NAND flash memory with ECC hardware to check up to 8 Kbytes of data
- Interface with synchronous DRAM (SDRAM/Mobile LPDDR SDRAM) memories
- Interface with parallel LCD modules, supporting Intel 8080 and Motorola 6800 modes.
- Burst mode support for faster access to synchronous devices such as NOR flash memory, PSRAM and SDRAM)
- Programmable continuous clock output for asynchronous and synchronous accesses
- 8-, 16- or 32-bit wide data bus
- Independent chip select control for each memory bank
- Independent configuration for each memory bank
- Write enable and byte lane select outputs for use with PSRAM, SRAM and SDRAM devices
- External asynchronous wait control
- Write FIFO with 16 x32-bit depth
- Cacheable Read FIFO with 6 x32-bit depth (6 x14-bit address tag) for SDRAM controller.

The Write FIFO is common to all memory controllers and consists of:

- a Write Data FIFO which stores the AHB data to be written to the memory (up to 32 bits) plus one bit for the AHB transfer (burst or not sequential mode)
- a Write Address FIFO which stores the AHB address (up to 28 bits) plus the AHB data size (up to 2 bits). When operating in burst mode, only the start address is stored

except when crossing a page boundary (for PSRAM and SDRAM). In this case, the AHB burst is broken into two FIFO entries.

The Write FIFO can be disabled by setting the WFDIS bit in the FMC\_BCR1 register.

At startup the FMC pins must be configured by the user application. The FMC I/O pins which are not used by the application can be used for other purposes.

The FMC registers that define the external device type and associated characteristics are usually set at boot time and do not change until the next reset or power-up. However, the settings can be changed at any time.

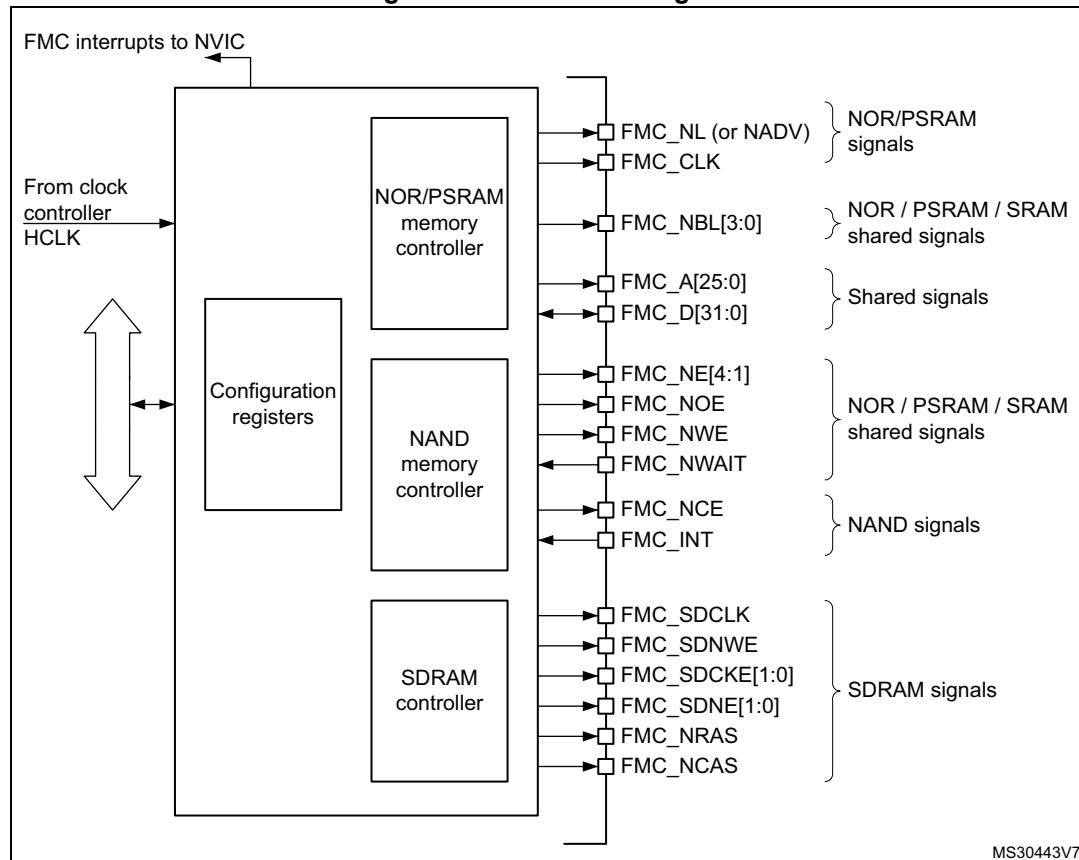
### 13.3 FMC block diagram

The FMC consists of the following main blocks:

- The AHB interface (including the FMC configuration registers)
- The NOR flash/PSRAM/SRAM controller
- The SDRAM controller
- The NAND flash controller

The block diagram is shown in the figure below.

**Figure 33. FMC block diagram**



MS30443V7

## 13.4 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 a FMC bank (Bank 1 to 4) which is not enabled.
- When reading or writing to the NOR flash bank while the FACCEN bit is reset in the FMC\_BCRx register.
- When writing to a write protected SDRAM bank (WP bit set in the SDRAM\_SDCRx register).
- When the SDRAM address range is violated (access to reserved address range)

The effect of an AHB error depends on the AHB master which has attempted the R/W access:

- If the access has been attempted by the Cortex®-M7 CPU, a hard fault interrupt is generated.
- If the access has been performed by a DMA controller, a DMA transfer error is generated and the corresponding DMA channel is automatically disabled.

The AHB clock (HCLK) is the reference clock for the FMC.

### 13.4.1 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 to the right data through its byte lanes NBL[3:0].

Bytes to be written are addressed by NBL[3:0].

All memory bytes are read (NBL[3:0] are driven low during read transaction) and the useless ones are discarded.

- Accesses to devices that do not have the byte select feature (NOR and NAND flash memories)

This situation occurs when a byte access is requested to a 16-bit wide flash memory. Since the device cannot be accessed in Byte mode (only 16-bit words can be read/written from/to the flash memory), Write transactions and Read transactions are allowed (the controller reads the entire 16-bit memory word and uses only the required byte).

### Wrap support for NOR flash/PSRAM and SDRAM

The synchronous memories must be configured in Linear burst mode of undefined length as not all masters can issue a wrap transactions.

If a master generates an AHB wrap transaction:

- The read is split into two linear burst transactions.
- The write is split into two linear burst transactions if the write fifo is enabled and into several linear burst transactions if the write fifo is disabled.

### Configuration registers

The FMC can be configured through a set of registers. Refer to [Section 13.6.6](#), for a detailed description of the NOR flash/PSRAM controller registers. Refer to [Section 13.7.7](#), for a detailed description of the NAND flash registers and to [Section 13.8.5](#) for a detailed description of the SDRAM controller registers.

## 13.5 External device address mapping

From the FMC point of view, the external memory is divided into fixed-size banks of 256 Mbytes each (see [Figure 34](#)):

- Bank 1 used to address up to 4 NOR flash memory or PSRAM devices. This bank is split into 4 NOR/PSRAM subbanks with 4 dedicated chip selects, as follows:
  - Bank 1 - NOR/PSRAM 1
  - Bank 1 - NOR/PSRAM 2
  - Bank 1 - NOR/PSRAM 3
  - Bank 1 - NOR/PSRAM 4
- Bank 3 used to address NAND flash memory devices. The MPU memory attribute for this space must be reconfigured by software to Device.
- Bank 4 and 5 used to address SDRAM devices (1 device per bank).

For each bank the type of memory to be used can be configured by the user application through the Configuration register.

**Figure 34. FMC memory banks**

Address	Bank	Supported memory type
0x6000 0000	Bank 1 4 x 64 MB	NOR/PSRAM/ SRAM
0x6FFF FFFF		
0x7000 0000	Bank 2 Not used	
0x7FFF FFFF		
0x8000 0000	Bank 3 4 x 64 MB	NAND flash memory
0x8FFF FFFF		
0x9000 0000	Bank 4 Not used	
0x9FFF FFFF		
0xC000 0000	SDRAM Bank 1 4 x 64 MB	SDRAM
0xCFFF FFFF		
0xD000 0000	SDRAM Bank 2 4 x 64 MB	
0xDFFF FFFF		

MSv69582V1

### 13.5.1 NOR/PSRAM address mapping

HADDR[27:26] bits are used to select one of the four memory banks as shown in [Table 48](#).

**Table 48. 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 49. NOR/PSRAM External memory address**

<b>Memory width<sup>(1)</sup></b>	<b>Data address issued to the memory</b>	<b>Maximum memory capacity (bits)</b>
8-bit	HADDR[25:0]	64 Mbytes x 8 = 512 Mbits
16-bit	HADDR[25:1] >> 1	64 Mbytes/2 x 16 = 512 Mbits
32-bit	HADDR[25:2] >> 2	64 Mbytes/4 x 32 = 512 Mbits

1. In case of a 16-bit external memory width, the FMC internally uses HADDR[25:1] to generate the address for external memory FMC\_A[24:0]. In case of a 32-bit memory width, the FMC internally uses HADDR[25:2] to generate the external address.  
Whatever the external memory width, FMC\_A[0] must be connected to external memory address A[0].

### 13.5.2 NAND flash memory address mapping

The NAND bank is divided into memory areas as indicated in [Table 50](#).

**Table 50. NAND memory mapping and timing registers**

<b>Start address</b>	<b>End address</b>	<b>FMC bank</b>	<b>Memory space</b>	<b>Timing register</b>
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 51](#) 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 51. NAND bank selection**

<b>Section name</b>	<b>HADDR[17:16]</b>	<b>Address range</b>
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.

### 13.5.3 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 52](#).

**Table 52. 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 53. 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
32-bit	HADDR[27:26]	HADDR[25:13]	HADDR[12:2]	256 Mbytes: 4 x 8K x 2K x 4

- 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].
- The AutoPrecharge is not supported. FMC\_A[10] must be connected to the external memory address A[10] but it is always driven low.

The HADDR[27:0] bits are translated to external SDRAM address depending on the SDRAM controller configuration:

- Data size:8, 16 or 32 bits
- Row size:11, 12 or 13 bits
- Column size: 8, 9, 10 or 11 bits
- Number of internal banks: two or four internal banks

The following tables show the SDRAM address mapping versus the SDRAM controller configuration.

**Table 54. 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
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 55. 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
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				

**Table 55. SDRAM address mapping with 16-bit data bus width<sup>(1)(2)</sup> (continued)**

Row size Configuration	HADDR(AHB address Lines)																												
	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
12-bit row size configuration	Res.			Bank [1:0]		Row[11:0]														Column[7:0]				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 [1:0]		Row[12:0]															Column[10:0]				BM0						

1. BANK[1:0] are the Bank Address BA[1:0]. When only 2 internal banks are used, BA1 must always be set to '0'.

2. Access to Reserved space (Res.) generates an AHB error.

3. BM0: is the byte mask for 16-bit access.

**Table 56. SDRAM address mapping with 32-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]				BM[1:0] <sup>(3)</sup>					
	Res.			Bank [1:0]		Row[10:0]														Column[8:0]				BM[1:0]					
	Res.			Bank [1:0]		Row[10:0]														Column[9:0]				BM[1:0]					
	Res.	Bank [1:0]		Row[10:0]															Column[10:0]				BM[1:0]						
12-bit row size configuration	Res.			Bank [1:0]		Row[11:0]														Column[7:0]				BM[1:0]					
	Res.			Bank [1:0]		Row[11:0]														Column[8:0]				BM[1:0]					
	Res.			Bank [1:0]		Row[11:0]														Column[9:0]				BM[1:0]					
	Res.	Bank [1:0]		Row[11:0]															Column[10:0]				BM[1:0]						

**Table 56. SDRAM address mapping with 32-bit data bus width<sup>(1)(2)</sup> (continued)**

Row size configuration	HADDR(AHB address Lines)																											
	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
13-bit row size configuration	Res.	Bank [1:0]	Row[12:0]																Column[7:0]				BM[1:0]					
	Res.	Bank [1:0]	Row[12:0]																Column[8:0]				BM[1:0]					
	Res.	Bank [1:0]	Row[12:0]																Column[9:0]				BM[1:0]					
	Bank [1:0]	Row[12:0]																Column[10:0]				BM[1:0]						

1. BANK[1:0] are the Bank Address BA[1:0]. When only 2 internal banks are used, BA1 must always be set to '0'.

2. Access to Reserved space (Res.) generates an AHB error.

3. BM[1:0]: is the byte mask for 32-bit access.

## 13.6 NOR flash/PSRAM controller

The FMC generates the appropriate signal timings to drive the following types of memories:

- Asynchronous SRAM and ROM
  - 8 bits
  - 16 bits
  - 32 bits
- PSRAM (CellularRAM™)
  - Asynchronous mode
  - Burst mode for synchronous accesses
  - Multiplexed or non-multiplexed
- NOR flash memory
  - Asynchronous mode
  - Burst mode for synchronous accesses
  - Multiplexed or non-multiplexed

The FMC outputs a unique chip select signal, NE[4:1], per bank. All the other signals (addresses, data and control) are shared.

The FMC supports a wide range of devices through a programmable timings among which:

- Programmable wait states (up to 15)
- Programmable bus turnaround cycles (up to 15)
- Programmable output enable and write enable delays (up to 15)
- Independent read and write timings and protocol to support the widest variety of memories and timings
- Programmable continuous clock (FMC\_CLK) output.

The FMC Clock (FMC\_CLK) is a submultiple of the HCLK clock. It can be delivered to the selected external device either during synchronous accesses only or during asynchronous

and synchronous accesses depending on the CCKEN bit configuration in the FMC\_BCR1 register:

- If the CCLKEN bit is reset, the FMC generates the clock (CLK) only during synchronous accesses (Read/write transactions).
- If the CCLKEN bit is set, the FMC generates a continuous clock during asynchronous and synchronous accesses. To generate the FMC\_CLK continuous clock, Bank 1 must be configured in Synchronous mode (see [Section 13.6.6: 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 (MWD) otherwise the FMC\_CLK frequency is changed depending on AHB data transaction (refer to [Section 13.6.5: 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 13.6.6: NOR/PSRAM controller registers](#)).

The programmable memory parameters include access times (see [Table 57](#)) and support for wait management (for PSRAM and NOR flash accessed in Burst mode).

**Table 57. 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

### 13.6.1 External memory interface signals

[Table 58](#), [Table 59](#) and [Table 60](#) 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 58. Non-multiplexed I/O NOR flash memory**

FMC signal name	I/O	Function
CLK	O	Clock (for synchronous access)
A[25:0]	O	Address bus
D[31:0]	I/O	Bidirectional data bus
NE[x]	O	Chip select, x = 1..4
NOE	O	Output enable
NWE	O	Write enable
NL(=NADV)	O	Latch enable (this signal is called address valid, NADV, by some NOR flash devices)
NWAIT	I	NOR flash wait input signal to the FMC

The maximum capacity is 512 Mbits (26 address lines).

### NOR flash memory, 16-bit multiplexed I/Os

**Table 59. 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 60. Non-multiplexed I/Os PSRAM/SRAM**

FMC signal name	I/O	Function
CLK	O	Clock (only for PSRAM synchronous access)
A[25:0]	O	Address bus
D[31:0]	I/O	Data bidirectional bus
NE[x]	O	Chip select, x = 1..4 (called NCE by PSRAM (CellularRAM™ i.e. CRAM))

**Table 60. Non-multiplexed I/Os PSRAM/SRAM (continued)**

FMC signal name	I/O	Function
NOE	O	Output enable
NWE	O	Write enable
NL(= NADV)	O	Address valid only for PSRAM input (memory signal name: NADV)
NWAIT	I	PSRAM wait input signal to the FMC
NBL[3:0]	O	Byte lane output. Byte 0 to Byte 3 control (Upper and lower byte enable)

The maximum capacity is 512 Mbits.

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

**Table 61. 16-Bit multiplexed I/O PSRAM**

FMC signal name	I/O	Function
CLK	O	Clock (for synchronous access)
A[25:16]	O	Address bus
AD[15:0]	I/O	16-bit multiplexed, bidirectional address/data bus (the 16-bit address A[15:0] and data D[15:0] are multiplexed on the databus)
NE[x]	O	Chip select, x = 1..4 (called NCE by PSRAM (CellularRAM™ i.e. CRAM))
NOE	O	Output enable
NWE	O	Write enable
NL(= NADV)	O	Address valid PSRAM input (memory signal name: NADV)
NWAIT	I	PSRAM wait input signal to the FMC
NBL[1:0]	O	Byte lane output. Byte 0 and Byte 1 control (upper and lower byte enable)

The maximum capacity is 512 Mbits (26 address lines).

### 13.6.2 Supported memories and transactions

*Table 62* 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 62. 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]
	Synchronous	W	16/32	16	Y	-
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]

### 13.6.3 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.

### 13.6.4 NOR flash/PSRAM controller asynchronous transactions

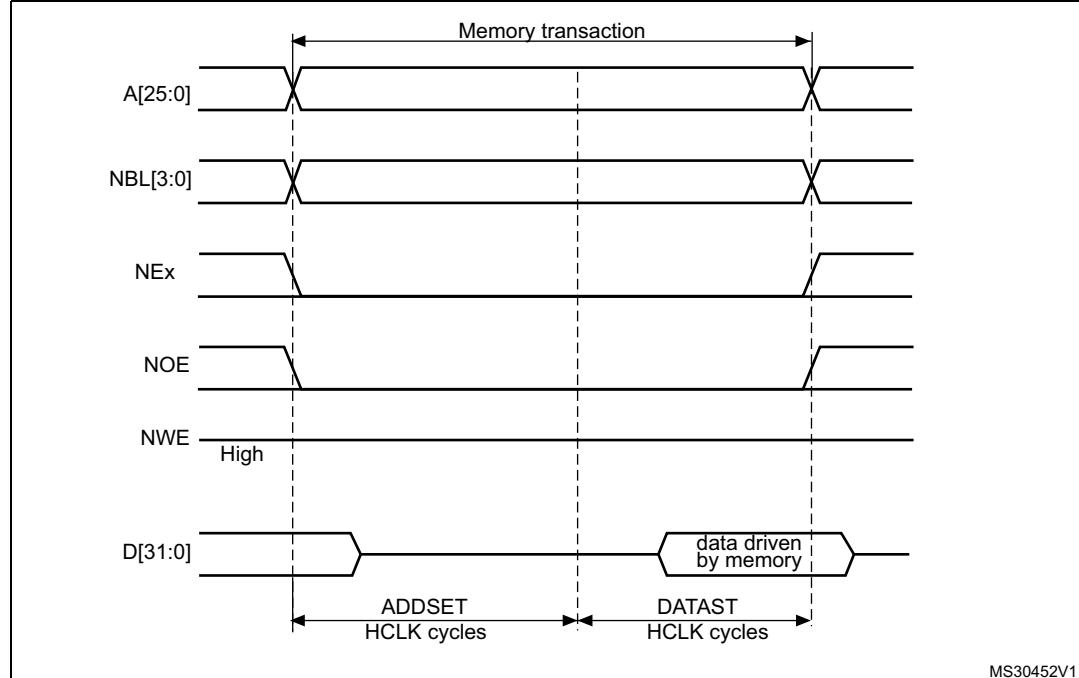
#### Asynchronous static memories (NOR flash, PSRAM, SRAM)

- 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 mode 1 or mode 2 as follows:
  - Mode 1 is the default mode when SRAM/PSRAM memory type is selected (MTYP = 0x0 or 0x01 in the FMC\_BCRx register)
  - Mode 2 is the default mode when NOR memory type is selected (MTYP = 0x10 in the FMC\_BCRx register).

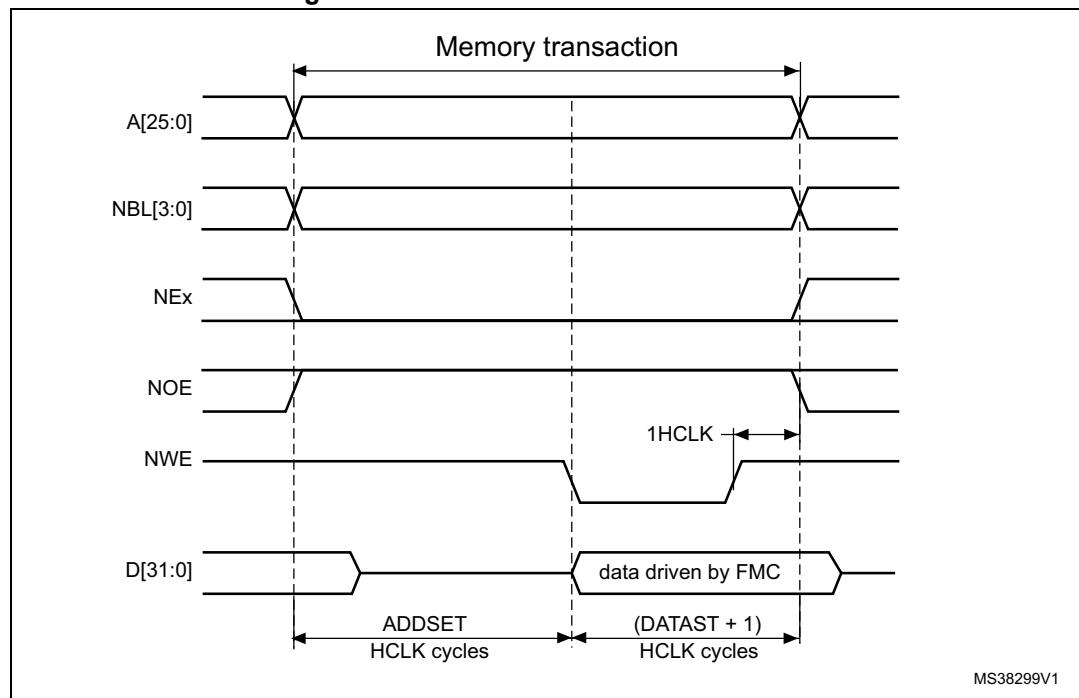
### Mode 1 - SRAM/PSRAM (CRAM)

The next figures show the read and write transactions for the supported modes followed by the required configuration of FMC\_BCRx, and FMC\_BTRx/FMC\_BWTRx registers.

**Figure 35. Mode 1 read access waveforms**



**Figure 36. Mode 1 write access waveforms**



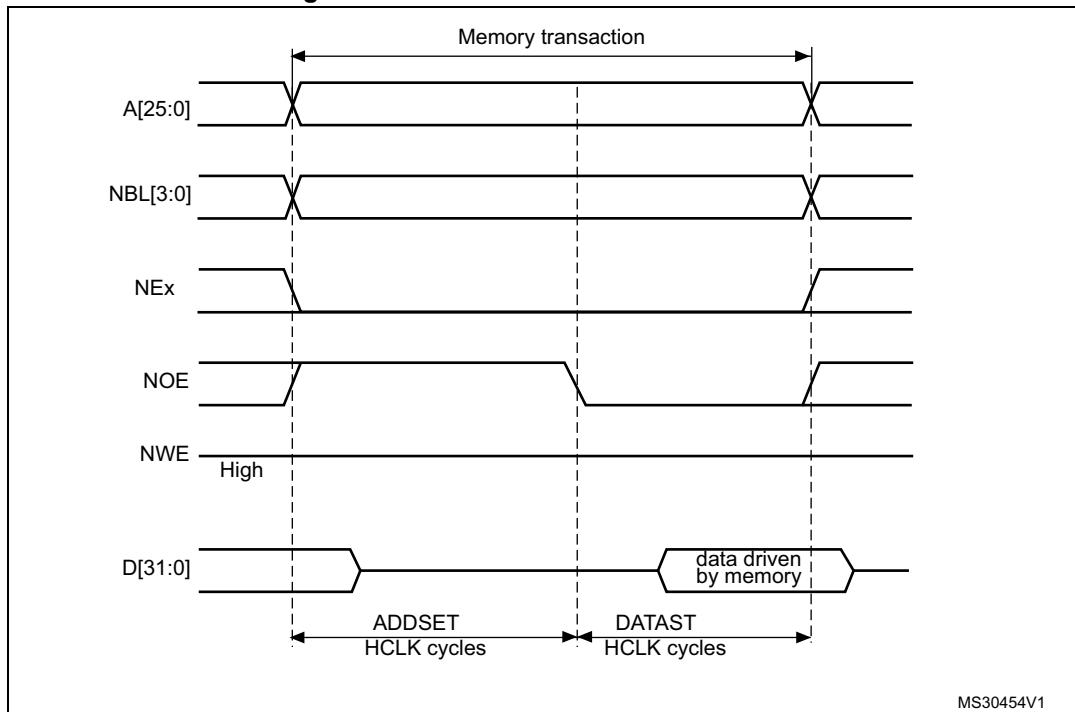
The one HCLK cycle at the end of the write transaction helps guarantee the address and data hold time after the NWE rising edge. Due to the presence of this HCLK cycle, the DATAST value must be greater than zero (DATAST > 0).

**Table 63. FMC\_BCRx bitfields (mode 1)**

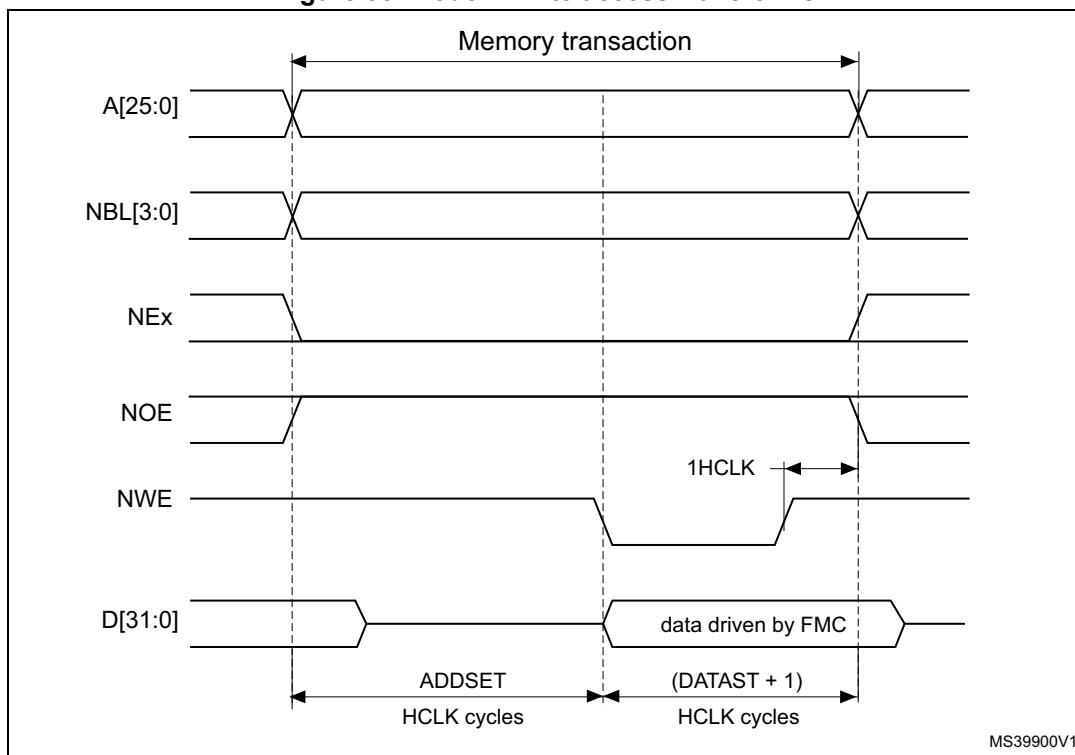
Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x0
13	WAITEN	0x0 (no effect in Asynchronous mode)
12	WREN	As needed
11	WAITCFG	Don't care
10	Reserved	0x0
9	WAITPOL	Meaningful only if bit 15 is 1
8	BURSTEN	0x0
7	Reserved	0x1
6	FACCEN	Don't care
5:4	MWID	As needed
3:2	MTYP	As needed, exclude 0x2 (NOR flash memory)
1	MUXE	0x0
0	MBKEN	0x1

**Table 64. FMC\_BTRx bitfields (mode 1)**

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	Don't care
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST+1 HCLK cycles for write accesses, DATAST HCLK cycles for read accesses).
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles). Minimum value for ADDSET is 0.

**Mode A - SRAM/PSRAM (CRAM) OE toggling****Figure 37. Mode A read access waveforms**

1. NBL[3:0] are driven low during the read access

**Figure 38. Mode A write access waveforms**

The differences compared with Mode 1 are the toggling of NOE and the independent read and write timings.

**Table 65. FMC\_BCRx bitfields (mode A)**

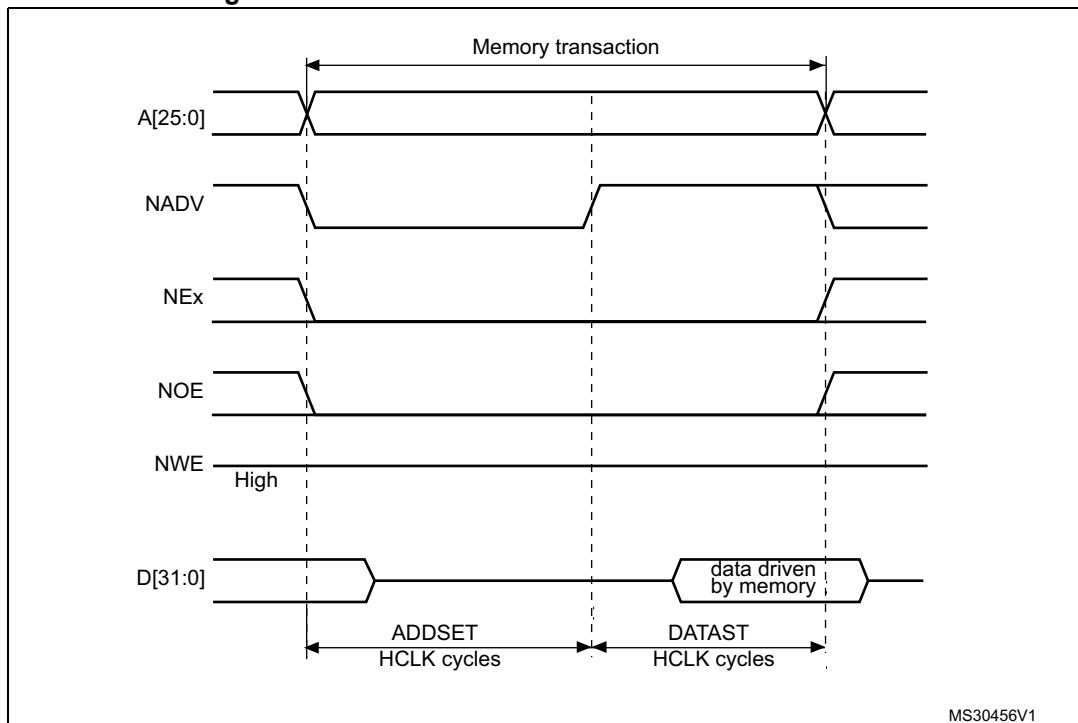
Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1
13	WAITEN	0x0 (no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

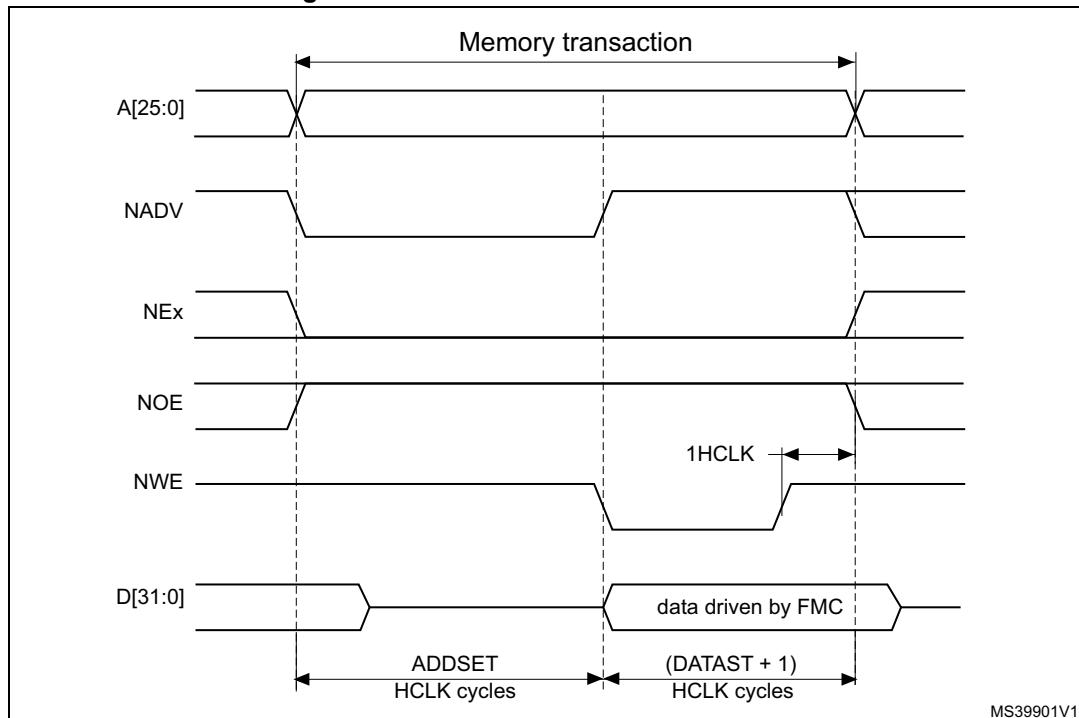
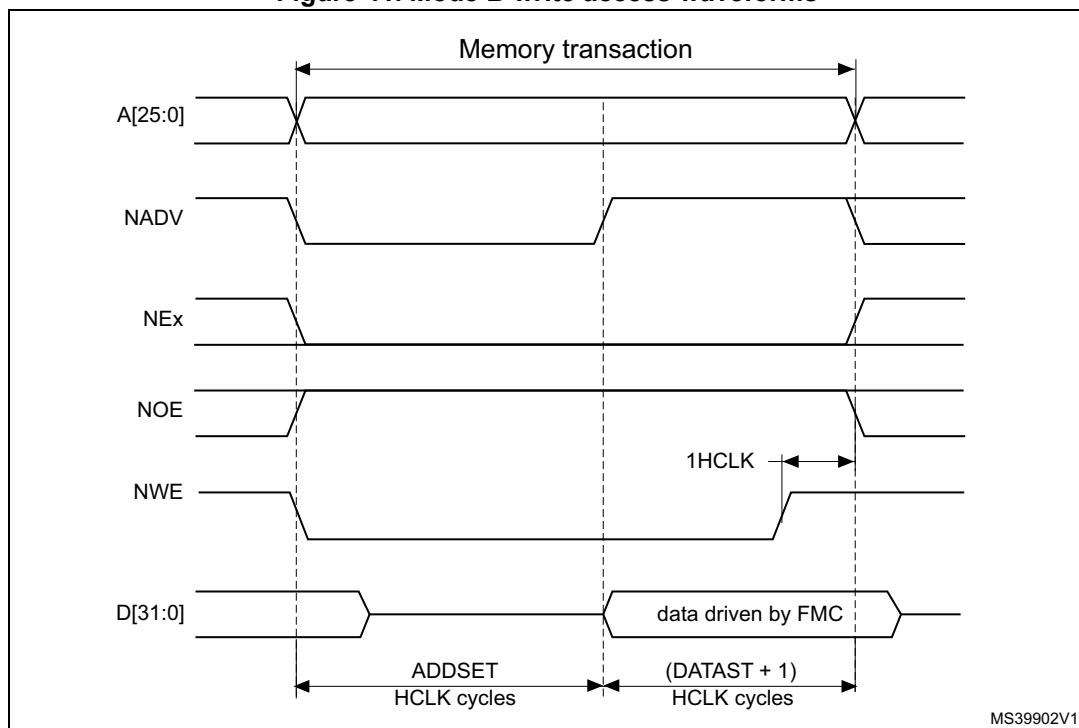
**Table 66. FMC\_BTRx bitfields (mode A)**

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 67. FMC\_BWTRx bitfields (mode A)**

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

**Mode 2/B - NOR flash****Figure 39. Mode 2 and mode B read access waveforms**

**Figure 40. Mode 2 write access waveforms****Figure 41. Mode B write access waveforms**

The differences with mode 1 are the toggling of NWE and the independent read and write timings when extended mode is set (mode B).

**Table 68. FMC\_BCRx bitfields (mode 2/B)**

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 69. FMC\_BTRx bitfields (mode 2/B)**

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 70. FMC\_BWTRx bitfields (mode 2/B)**

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

**Note:** The FMC\_BWTRx register is valid only if the Extended mode is set (mode B), otherwise its content is don't care.

### Mode C - NOR flash - OE toggling

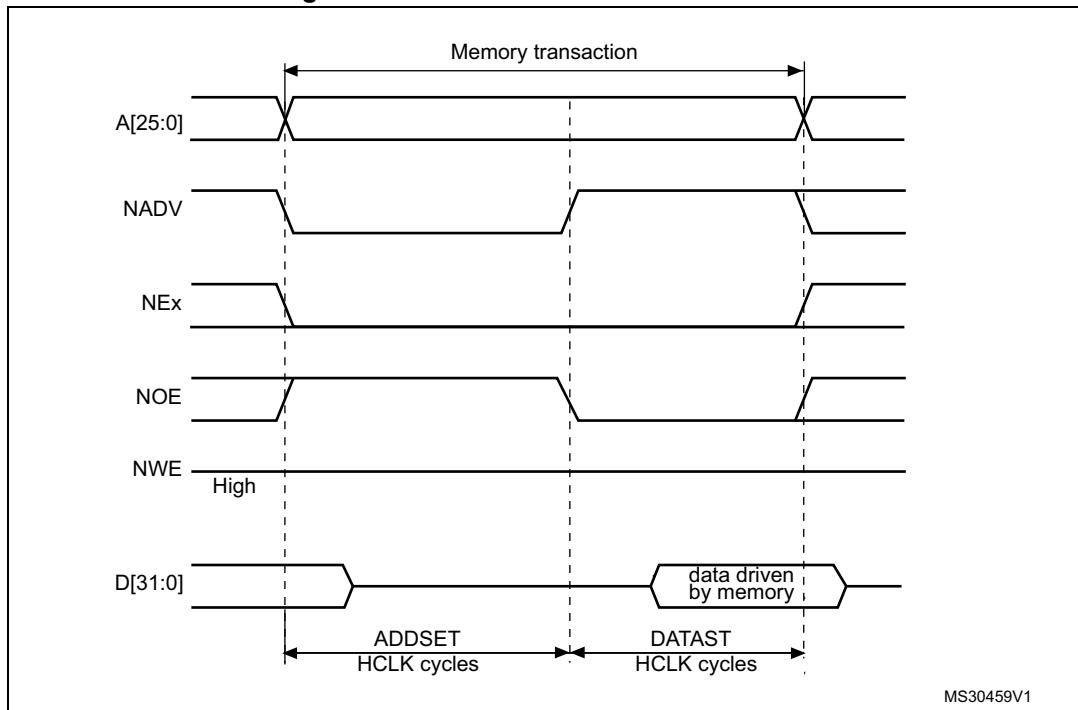
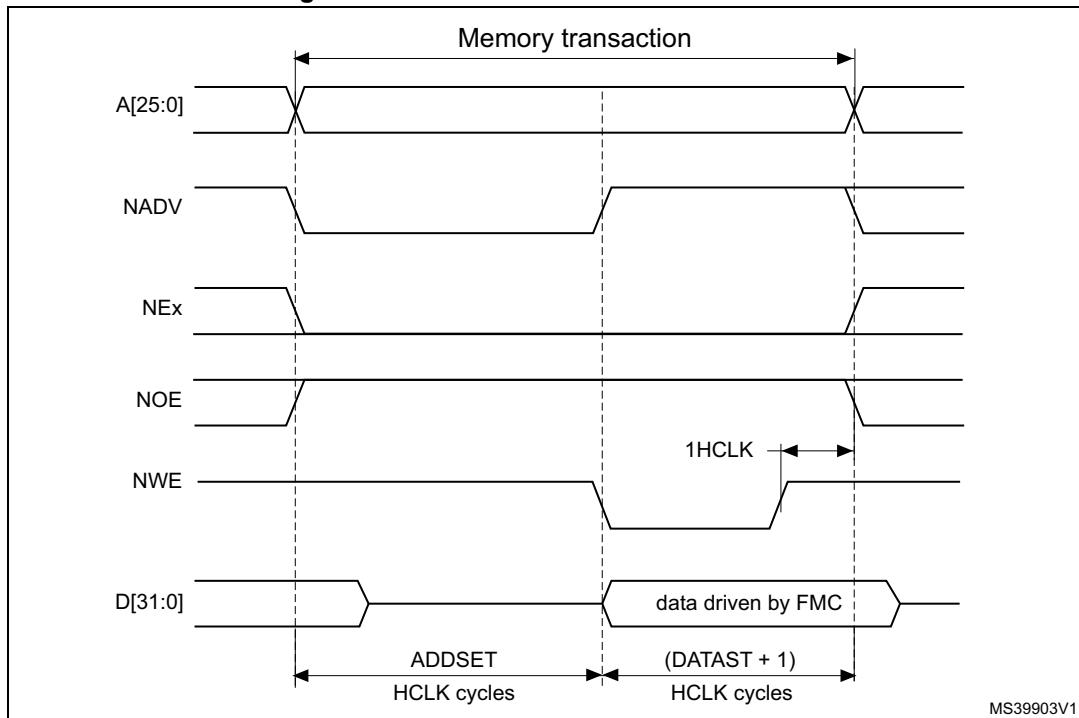
**Figure 42. Mode C read access waveforms**

Figure 43. Mode C write access waveforms



The differences compared with mode 1 are the toggling of NOE and the independent read and write timings.

Table 71. FMC\_BCRx bitfields (mode C)

Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCPWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1
13	WAITEN	0x0 (no effect in Asynchronous mode)
12	WREN	As needed
11	WAITCFG	Don't care
10	Reserved	0x0
9	WAITPOL	Meaningful only if bit 15 is 1
8	BURSTEN	0x0
7	Reserved	0x1
6	FACCEN	0x1
5:4	MWID	As needed

**Table 71. FMC\_BCRx bitfields (mode C) (continued)**

Bit number	Bit name	Value to set
3:2	MTYP	0x02 (NOR flash memory)
1	MUXEN	0x0
0	MBKEN	0x1

**Table 72. FMC\_BTRx bitfields (mode C)**

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x2
27:24	DATLAT	0x0
23:20	CLKDIV	0x0
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles) for read accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for read accesses. Minimum value for ADDSET is 0.

**Table 73. FMC\_BWTRx bitfields (mode C)**

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x2
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles) for write accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for write accesses. Minimum value for ADDSET is 0.

### Mode D - asynchronous access with extended address

Figure 44. Mode D read access waveforms

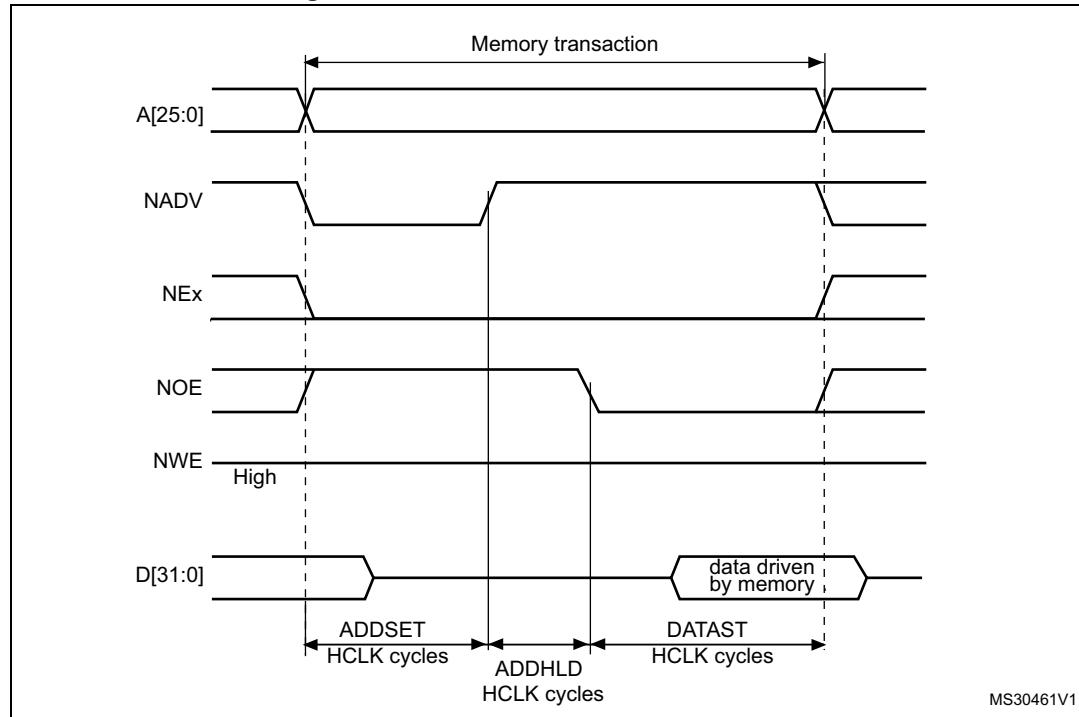
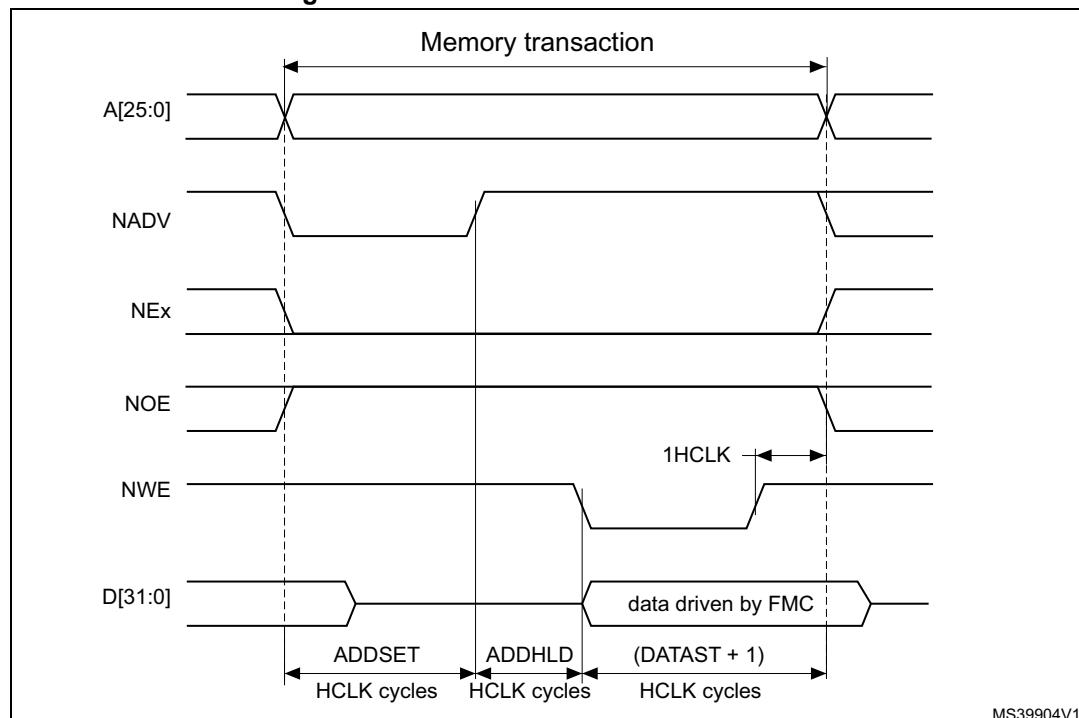


Figure 45. Mode D write access waveforms



The differences with mode 1 are the toggling of NOE that goes on toggling after NADV changes and the independent read and write timings.

**Table 74. FMC\_BCRx bitfields (mode D)**

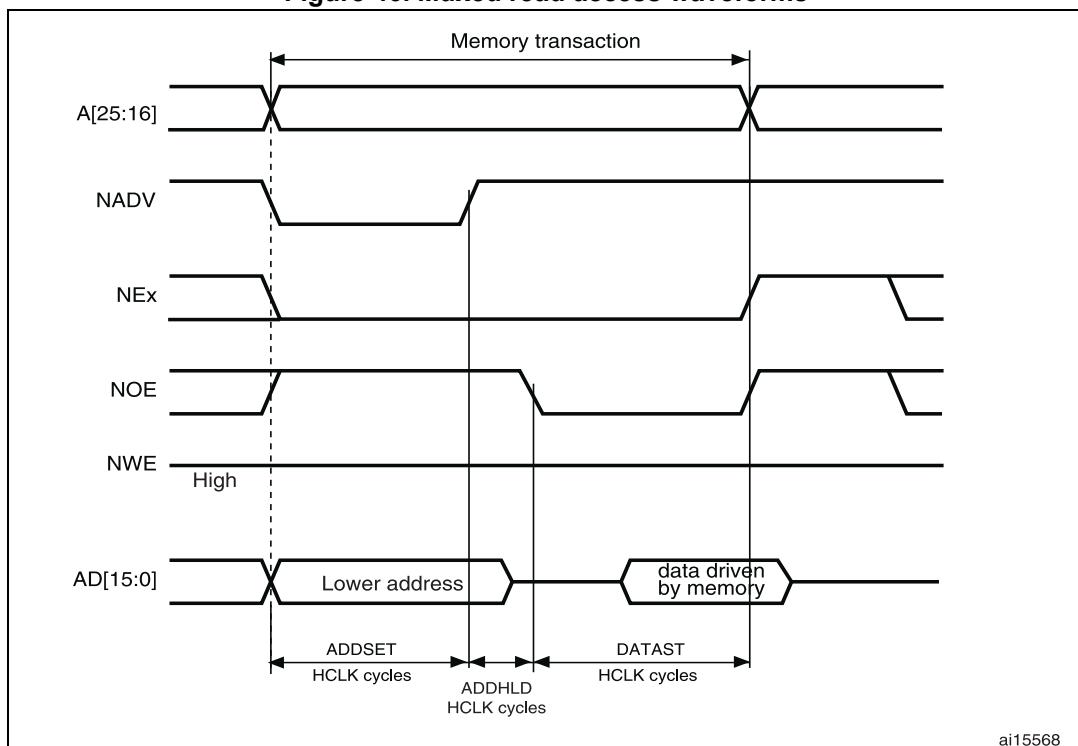
Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1
13	WAITEN	0x0 (no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

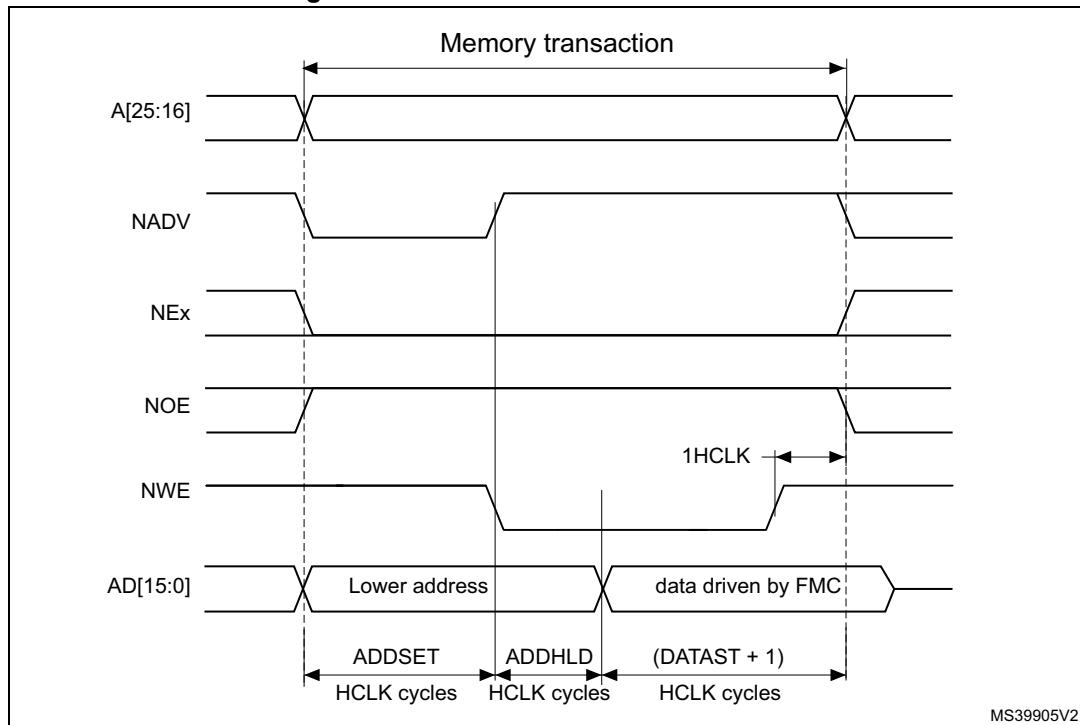
**Table 75. FMC\_BTRx bitfields (mode D)**

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 76. FMC\_BWTRx bitfields (mode D)**

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x3
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST + 1 HCLK cycles) for write accesses.
7:4	ADDHLD	Duration of the middle phase of the write access (ADDHLD HCLK cycles)
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for write accesses. Minimum value for ADDSET is 1.

**Muxed mode - multiplexed asynchronous access to NOR flash memory****Figure 46. Muxed read access waveforms**

**Figure 47. Muxed write access waveforms**

The difference with mode D is the drive of the lower address byte(s) on the data bus.

**Table 77. FMC\_BCRx bitfields (Muxed mode)**

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 77. FMC\_BCRx bitfields (Muxed mode) (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 78. FMC\_BTRx bitfields (Muxed mode)**

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x0
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles for read accesses and DATAST+1 HCLK cycles for write accesses).
7:4	ADDHLD	Duration of the middle phase of the access (ADDHLD HCLK cycles).
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles). Minimum value for ADDSET is 1.

### WAIT management in asynchronous accesses

If the asynchronous memory asserts the WAIT signal to indicate that it is not yet ready to accept or to provide data, the ASYNCWAIT bit has to be set in FMC\_BCRx register.

If the WAIT signal is active (high or low depending on the WAITPOL bit), the second access phase (Data setup phase), programmed by the DATAST bits, is extended until WAIT becomes inactive. Unlike the data setup phase, the first access phases (Address setup and Address hold phases), programmed by the ADDSET and ADDHLD bits, are not WAIT sensitive and so they are not prolonged.

The data setup phase must be programmed so that WAIT can be detected 4 HCLK cycles before the end of the memory transaction. The following cases must be considered:

1. The memory asserts the WAIT signal aligned to NOE/NWE which toggles:

$$\text{DATAST} \geq (4 \times \text{HCLK}) + \text{max\_wait\_assertion\_time}$$

2. The memory asserts the WAIT signal aligned to NEx (or NOE/NWE not toggling):

if

$$\text{max\_wait\_assertion\_time} > \text{address\_phase} + \text{hold\_phase}$$

then:

$$\text{DATAST} \geq (4 \times \text{HCLK}) + (\text{max\_wait\_assertion\_time} - \text{address\_phase} - \text{hold\_phase})$$

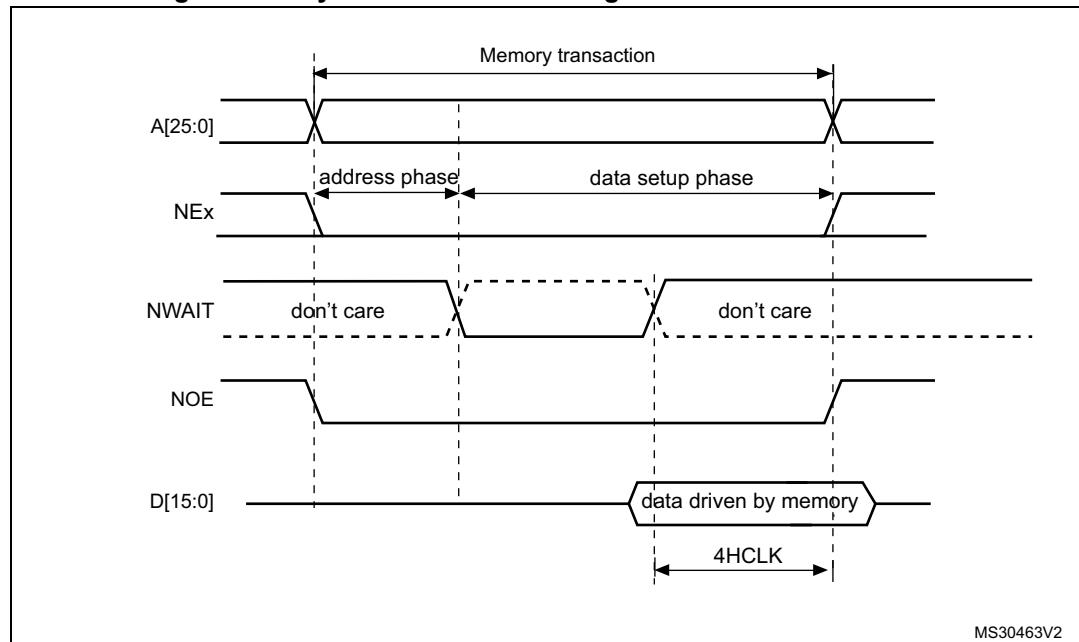
otherwise

$$\text{DATAST} \geq 4 \times \text{HCLK}$$

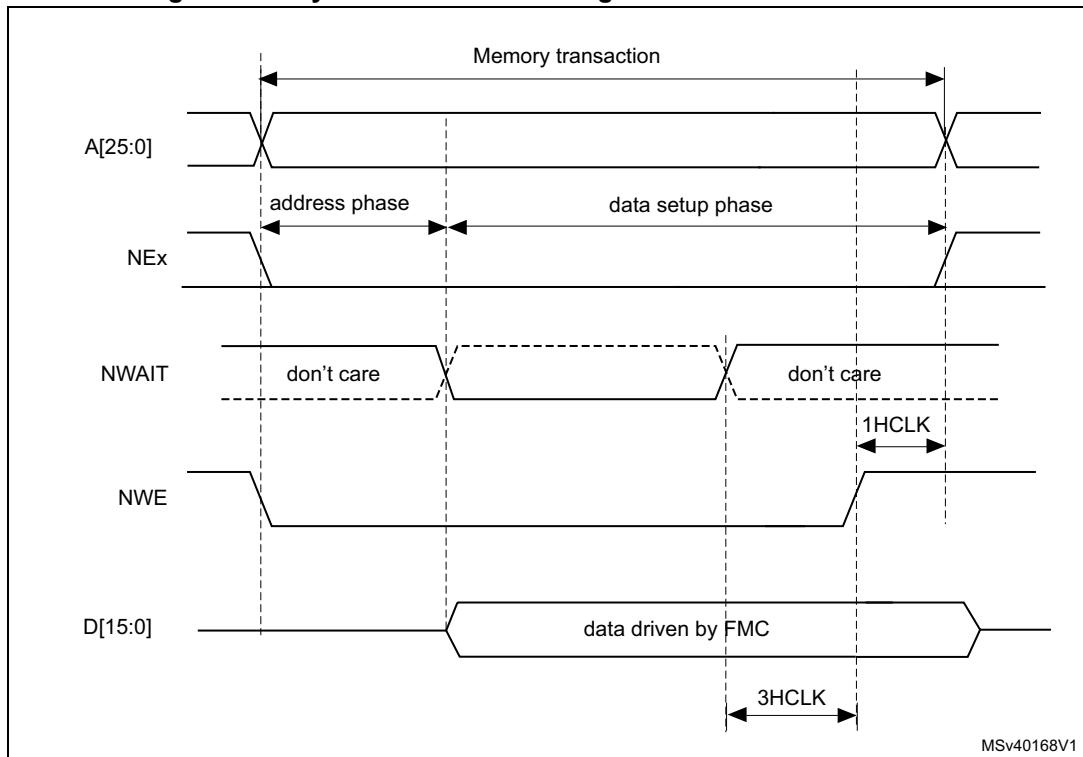
where max\_wait\_assertion\_time is the maximum time taken by the memory to assert the WAIT signal once NEx/NOE/NWE is low.

*Figure 48* and *Figure 49* show the number of HCLK clock cycles that are added to the memory access phase after WAIT is released by the asynchronous memory (independently of the above cases).

**Figure 48. Asynchronous wait during a read access waveforms**



1. NWAIT polarity depends on WAITPOL bit setting in FMC\_BCRx register.

**Figure 49. Asynchronous wait during a write access waveforms**

1. NWAIT polarity depends on WAITPOL bit setting in FMC\_BCRx register.

### 13.6.5 Synchronous transactions

The memory clock, FMC\_CLK, is a submultiple of HCLK. It depends on the value of CLKDIV and the MWID/ AHB data size, following the formula given below:

$$\text{FMC\_CLK divider ratio} = \max(\text{CLKDIV} + 1, \text{MWID(AHB data size)})$$

If MWID is 16 or 8-bit, the FMC\_CLK divider ratio is always defined by the programmed CLKDIV value.

If MWID is 32-bit, the FMC\_CLK divider ratio depends also on AHB data size.

Example:

- If CLKDIV=1, MWID = 32 bits, AHB data size=8 bits, FMC\_CLK=HCLK/4.
- If CLKDIV=1, MWID = 16 bits, AHB data size=8 bits, FMC\_CLK=HCLK/2.

NOR flash memories specify a minimum time from NADV assertion to CLK high. To meet this constraint, the FMC does not issue the clock to the memory during the first internal clock cycle of the synchronous access (before NADV assertion). This guarantees that the rising edge of the memory clock occurs in the middle of the NADV low pulse.

#### Data latency versus NOR memory latency

The data latency is the number of cycles to wait before sampling the data. The DATLAT value must be consistent with the latency value specified in the NOR flash configuration register. The FMC does not include the clock cycle when NADV is low in the data latency count.

**Caution:** Some NOR flash memories include the NADV Low cycle in the data latency count, so that the exact relation between the NOR flash latency and the FMC DATLAT parameter can be either:

- NOR flash latency = (DATLAT + 2) CLK clock cycles
- or NOR flash latency = (DATLAT + 3) CLK clock cycles

Some recent memories assert NWAIT during the latency phase. In such cases DATLAT can be set to its minimum value. As a result, the FMC samples the data and waits long enough to evaluate if the data are valid. Thus the FMC detects when the memory exits latency and real data are processed.

Other memories do not assert NWAIT during latency. In this case the latency must be set correctly for both the FMC and the memory, otherwise invalid data are mistaken for good data, or valid data are lost in the initial phase of the memory access.

### Single-burst transfer

When the selected bank is configured in Burst mode for synchronous accesses, if for example an AHB single-burst transaction is requested on 16-bit memories, the FMC performs a burst transaction of length 1 (if the AHB transfer is 16 bits), or length 2 (if the AHB transfer is 32 bits) and de-assert the chip select signal when the last data is strobed.

Such transfers are not the most efficient in terms of cycles compared to asynchronous read operations. Nevertheless, a random asynchronous access would first require to re-program the memory access mode, which would altogether last longer.

### Cross boundary page for CellularRAM™ 1.5

CellularRAM™ 1.5 does not allow burst access to cross the page boundary. The FMC controller is used to split automatically the burst access when the memory page size is reached by configuring the CPSIZE bits in the FMC\_BCR1 register following the memory page size.

### Wait management

For synchronous NOR flash memories, NWAIT is evaluated after the programmed latency period, which corresponds to (DATLAT+2) CLK clock cycles.

If NWAIT is active (low level when WAITPOL = 0, high level when WAITPOL = 1), wait states are inserted until NWAIT is inactive (high level when WAITPOL = 0, low level when WAITPOL = 1).

When NWAIT is inactive, the data is considered valid either immediately (bit WAITCFG = 1) or on the next clock edge (bit WAITCFG = 0).

During wait-state insertion via the NWAIT signal, the controller continues to send clock pulses to the memory, keeping the chip select and output enable signals valid. It does not consider the data as valid.

In Burst mode, there are two timing configurations for the NOR flash NWAIT signal:

- The flash memory asserts the NWAIT signal one data cycle before the wait state (default after reset).
- The flash memory asserts the NWAIT signal during the wait state

The FMC supports both NOR flash wait state configurations, for each chip select, thanks to the WAITCFG bit in the FMC\_BCRx registers (x = 0..3).

Figure 50. Wait configuration waveforms

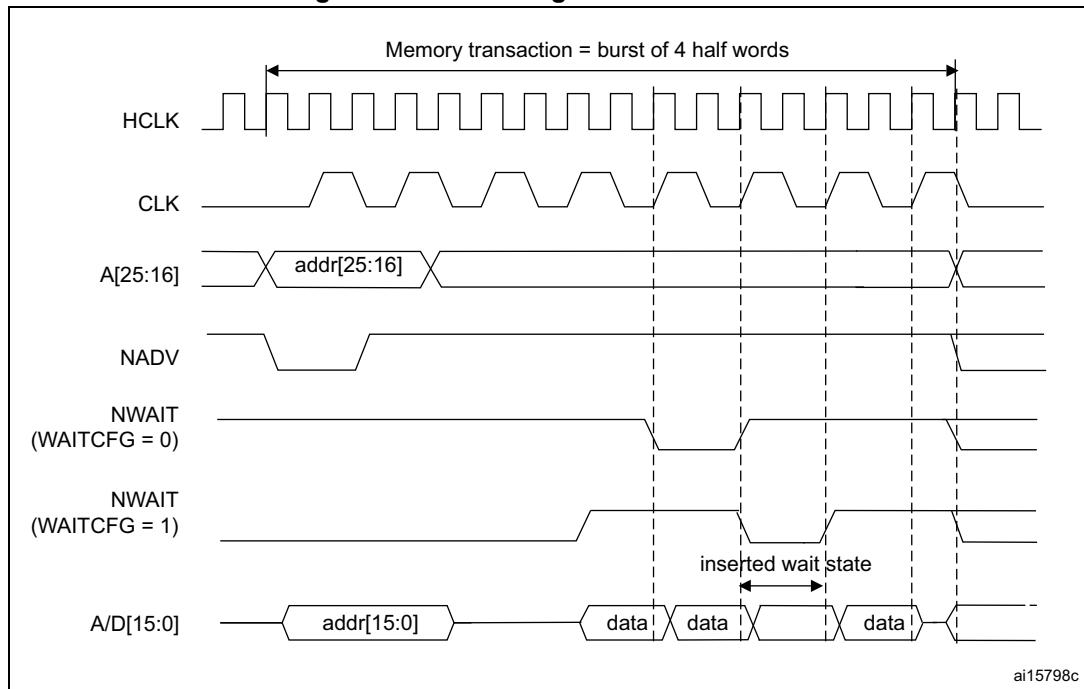
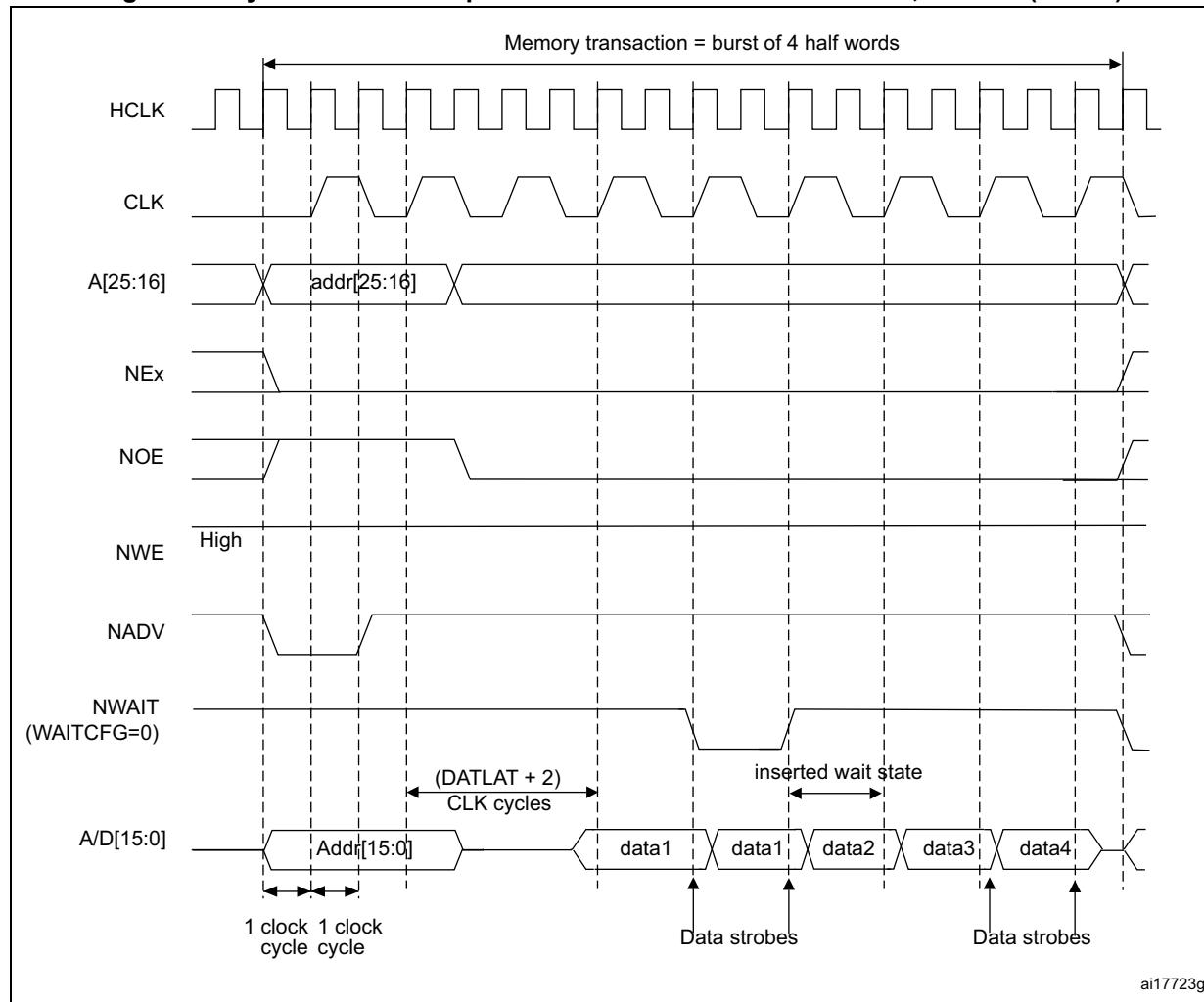


Figure 51. Synchronous multiplexed read mode waveforms - NOR, PSRAM (CRAM)



1. Byte lane outputs (NBL are not shown; for NOR access, they are held high, and, for PSRAM (CRAM) access, they are held low.

Table 79. FMC\_BCRx bitfields (Synchronous multiplexed read mode)

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

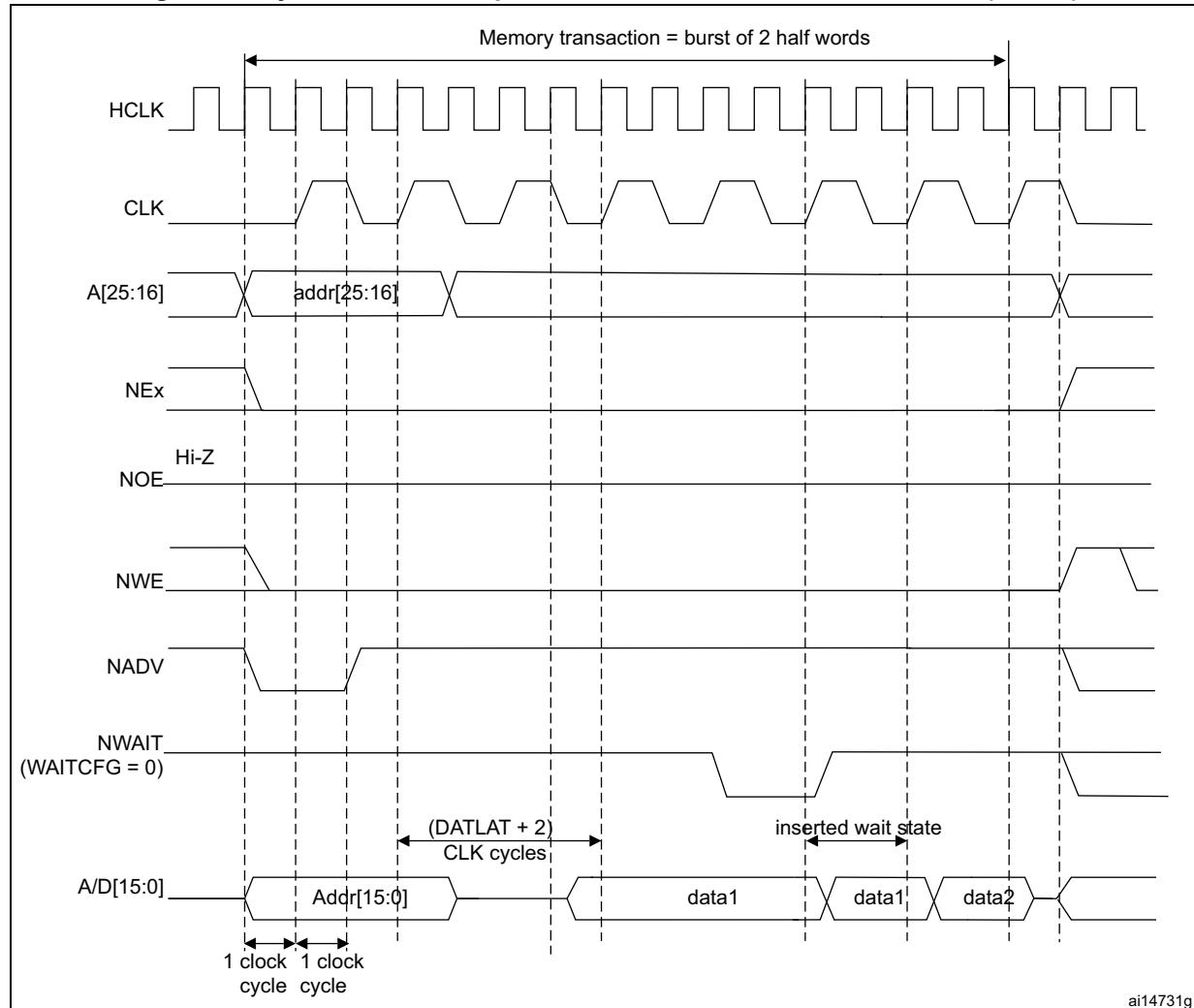
**Table 79. FMC\_BCRx bitfields (Synchronous multiplexed read mode) (continued)**

Bit number	Bit name	Value to set
11	WAITCFG	To be set according to memory
10	Reserved	0x0
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 80. FMC\_BTRx bitfields (Synchronous multiplexed read mode)**

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x0
27-24	DATLAT	Data latency
27-24	DATLAT	Data latency
23-20	CLKDIV	0x0 to get CLK = HCLK (Not supported) 0x1 to get CLK = 2 × HCLK ..
19-16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15-8	DATAST	Don't care
7-4	ADDHLD	Don't care
3-0	ADDSET	Don't care

Figure 52. Synchronous multiplexed write mode waveforms - PSRAM (CRAM)



1. The memory must issue NWAIT signal one cycle in advance, accordingly WAITCFG must be programmed to 0.
2. Byte Lane (NBL) outputs are not shown, they are held low while NEx is active.

Table 81. FMC\_BCRx bitfields (Synchronous multiplexed write mode)

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 81. FMC\_BCRx bitfields (Synchronous multiplexed write mode) (continued)**

Bit number	Bit name	Value to set
12	WREN	0x1
11	WAITCFG	0x0
10	Reserved	0x0
9	WAITPOL	to be set according to memory
8	BURSTEN	no effect on synchronous write
7	Reserved	0x1
6	FACCEN	Set according to memory support
5-4	MWID	As needed
3-2	MTYP	0x1
1	MUXEN	As needed
0	MBKEN	0x1

**Table 82. FMC\_BTRx bitfields (Synchronous multiplexed write mode)**

Bit number	Bit name	Value to set
31-30	Reserved	0x0
29:28	ACCMOD	0x0
27-24	DATLAT	Data latency
23-20	CLKDIV	0x0 to get CLK = HCLK (not supported) 0x1 to get CLK = 2 × HCLK
19-16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15-8	DATAST	Don't care
7-4	ADDHLD	Don't care
3-0	ADDSET	Don't care

### 13.6.6 NOR/PSRAM controller registers

#### SRAM/NOR-flash chip-select control register for bank x (FMC\_BCRx)

Address offset: 0x00 + 0x8 \* (x - 1), (x = 1 to 4)

Reset value: 0x0000 30DB, 0x0000 30D2, 0x0000 30D2, 0x0000 30D2

This register contains the control information of each memory bank, used for SRAMs, PSRAM and NOR flash memories.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	WFDIS	CCLK EN	CBURST RW	CPSIZE[2:0]		
										rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASYNC WAIT	EXT MOD	WAIT EN	WREN	WAIT CFG	Res.	WAIT POL	BURST EN	Res.	FACC EN	MWID[1:0]		MTYP[1:0]		MUX EN	MBK EN
rw	rw	rw	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw

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

##### Bit 21 WFDIS: Write FIFO disable

This bit disables the Write FIFO used by the FMC controller.

0: Write FIFO enabled (Default after reset)

1: Write FIFO disabled

*Note: The WFDIS bit of the FMC\_BCR2..4 registers is don't care. It is only enabled through the FMC\_BCR1 register.*

##### Bit 20 CCLKEN: Continuous clock enable

This bit enables the FMC\_CLK clock output to external memory devices.

0: The FMC\_CLK is only generated during the synchronous memory access (read/write transaction). The FMC\_CLK clock ratio is specified by the programmed CLKDIV value in the FMC\_BCRx register (default after reset).

1: The FMC\_CLK is generated continuously during asynchronous and synchronous access. The FMC\_CLK clock is activated when the CCLKEN is set.

*Note: The CCLKEN bit of the FMC\_BCR2..4 registers is don't care. It is only enabled through the FMC\_BCR1 register. Bank 1 must be configured in Synchronous mode to generate the FMC\_CLK continuous clock.*

*Note: If CCLKEN bit is set, the FMC\_CLK clock ratio is specified by CLKDIV value in the FMC\_BTR1 register. CLKDIV in FMC\_BWTR1 is don't care.*

*Note: If the Synchronous mode is used and CCLKEN bit is set, the synchronous memories connected to other banks than Bank 1 are clocked by the same clock (the CLKDIV value in the FMC\_BTR2..4 and FMC\_BWTR2..4 registers for other banks has no effect.)*

##### Bit 19 CBURSTRW: Write burst enable

For PSRAM (CRAM) operating in Burst mode, the bit enables synchronous accesses during write operations. The enable bit for synchronous read accesses is the BURSTEN bit in the FMC\_BCRx register.

0: Write operations are always performed in Asynchronous mode.

1: Write operations are performed in Synchronous mode.

**Bits 18:16 CPSIZE[2:0]: CRAM page size**

These are used for CellularRAM™ 1.5 which does not allow burst access to cross the address boundaries between pages. When these bits are configured, the FMC controller splits automatically the burst access when the memory page size is reached (refer to memory datasheet for page size).

000: No burst split when crossing page boundary (default after reset)

001: 128 bytes

010: 256 bytes

011: 512 bytes

100: 1024 bytes

Others: Reserved, must not be used

**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 mode 1 or mode 2 as follows:*

- *Mode 1 is the default mode when the SRAM/PSRAM memory type is selected (MTYP = 0x0 or 0x01)*
- *Mode 2 is the default mode when the NOR memory type is selected (MTYP = 0x10).*

**Bit 13 WAITEN: Wait enable bit**

This bit enables/disables wait-state insertion via the NWAIT signal when accessing the memory in Synchronous mode.

0: NWAIT signal is disabled (its level not taken into account, no wait state inserted after the programmed flash latency period).

1: NWAIT signal is enabled (its level is taken into account after the programmed latency period to insert wait states if asserted) (default after reset).

**Bit 12 WREN: Write enable bit**

This bit indicates whether write operations are enabled/disabled in the bank by the FMC.

0: Write operations are disabled in the bank by the FMC, an AHB error is reported.

1: Write operations are enabled for the bank by the FMC (default after reset).

**Bit 11 WAITCFG: Wait timing configuration**

The NWAIT signal indicates whether the data from the memory are valid or if a wait state must be inserted when accessing the memory in Synchronous mode. This configuration bit determines if NWAIT is asserted by the memory one clock cycle before the wait state or during the wait state:

0: NWAIT signal is active one data cycle before wait state (default after reset).

1: NWAIT signal is active during wait state (not used for PSRAM).

**Bit 10 Reserved, must be kept at reset value.****Bit 9 WAITPOL: Wait signal polarity bit**

Defines the polarity of the wait signal from memory used for either in Synchronous or Asynchronous mode.

0: NWAIT active low (default after reset)

1: NWAIT active high

**Bit 8 BURSTEN:** Burst enable bit

This bit enables/disables synchronous accesses during read operations. It is valid only for synchronous memories operating in Burst mode.

- 0: Burst mode disabled (default after reset). Read accesses are performed in Asynchronous mode.
- 1: Burst mode enable. Read accesses are performed in Synchronous mode.

**Bit 7 Reserved, must be kept at reset value.****Bit 6 FACCEN:** Flash access enable

Enables NOR flash memory access operations.

- 0: Corresponding NOR flash memory access is disabled.
- 1: Corresponding NOR flash memory access is enabled (default after reset).

**Bits 5:4 MWID[1:0]:** Memory data bus width

Defines the external memory device width, valid for all type of memories.

- 00: 8 bits
- 01: 16 bits (default after reset)
- 10: 32 bits
- 11: reserved

**Bits 3:2 MTYP[1:0]:** Memory type

Defines the type of external memory attached to the corresponding memory bank.

- 00: SRAM (default after reset for Bank 2...4)
- 01: PSRAM (CRAM)
- 10: NOR flash/OneNAND flash (default after reset for Bank 1)
- 11: reserved

**Bit 1 MUXEN:** Address/data multiplexing enable bit

When this bit is set, the address and data values are multiplexed on the data bus, valid only with NOR and PSRAM memories:

- 0: Address/data non multiplexed
- 1: Address/data multiplexed on databus (default after reset)

**Bit 0 MBKEN:** Memory bank enable bit

Enables the memory bank. After reset Bank1 is enabled, all others are disabled. Accessing a disabled bank causes an ERROR on AHB bus.

- 0: Corresponding memory bank is disabled.
- 1: Corresponding memory bank is enabled.

**SRAM/NOR-flash chip-select timing register for bank x (FMC\_BTRx)**

Address offset:  $0x04 + 0x8 * (x - 1)$ , ( $x = 1$  to 4)

Reset value: 0xFFFF FFFF

This register contains the control information of each memory bank, used for SRAMs, PSRAM and NOR flash memories. If the EXTMOD bit is set in the FMC\_BCRx register, then this register is partitioned for write and read access, that is, 2 registers are available: one to

configure read accesses (this register) and one to configure write accesses (FMC\_BWTRx registers).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	ACCMOD[1:0]		DATLAT[3:0]				CLKDIV[3:0]				BUSTURN[3:0]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATAST[7:0]								ADDHLD[3:0]				ADDSET[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

#### Bits 29:28 **ACCMOD[1:0]: Access mode**

Specifies the asynchronous access modes as shown in the timing diagrams. These bits are taken into account only when the EXTMOD bit in the FMC\_BCRx register is 1.

- 00: Access mode A
- 01: Access mode B
- 10: Access mode C
- 11: Access mode D

#### Bits 27:24 **DATLAT[3:0]: (see note below bit descriptions): Data latency for synchronous memory**

For synchronous access with read/write Burst mode enabled (BURSTEN / CBURSTRW bits set), defines the number of memory clock cycles (+2) to issue to the memory before reading/writing the first data:

This timing parameter is not expressed in HCLK periods, but in FMC\_CLK periods.

For asynchronous access, this value is don't care.

0000: Data latency of 2 CLK clock cycles for first burst access

1111: Data latency of 17 CLK clock cycles for first burst access (default value after reset)

#### Bits 23:20 **CLKDIV[3:0]: Clock divide ratio (for FMC\_CLK signal)**

Defines the period of FMC\_CLK clock output signal, expressed in number of HCLK cycles:

0000: Reserved

0001: FMC\_CLK period = 2 × HCLK periods

0010: FMC\_CLK period = 3 × HCLK periods

1111: FMC\_CLK period = 16 × HCLK periods (default value after reset)

In asynchronous NOR flash, SRAM or PSRAM accesses, this value is don't care.

*Note: Refer to [Section 13.6.5: 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 enables to match the minimum time between consecutive transactions (tEHEL from NEx high to NEx low) and the maximum time needed by the memory to free the data bus after a read access (tEHQZ). The programmed bus turnaround delay is inserted between an asynchronous read (muxed or mode D) or write transaction and any other asynchronous /synchronous read or write to or from a static bank. The bank can be the same or different in case of read, in case of write the bank can be different except for muxed or mode D.

In some cases, whatever the programmed BUSTURN values, the bus turnaround delay is fixed

as follows:

- The bus turnaround delay is not inserted between two consecutive asynchronous write transfers to the same static memory bank except for muxed mode and mode 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 muxed mode and mode D.
  - An asynchronous read to an asynchronous or synchronous write to any static bank or dynamic bank except for muxed mode and mode 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 35](#) to [Figure 47](#)), used in asynchronous accesses:

0000 0000: Reserved

0000 0001: DATAST phase duration = 1 × HCLK clock cycles

0000 0010: DATAST phase duration = 2 × HCLK clock cycles

...

1111 1111: DATAST phase duration = 255 × HCLK clock cycles (default value after reset)

For each memory type and access mode data-phase duration, refer to the respective figure ([Figure 35](#) to [Figure 47](#)).

Example: Mode 1, write access, DATAST=1: Data-phase duration= DATAST+1 = 2 HCLK clock cycles.

*Note: In synchronous accesses, this value is don't care.*

**Bits 7:4 ADDHLD[3:0]: Address-hold phase duration**

These bits are written by software to define the duration of the *address hold* phase (refer to [Figure 35](#) to [Figure 47](#)), used in mode D or multiplexed accesses:

0000: Reserved

0001: ADDHLD phase duration = 1 × HCLK clock cycle

0010: ADDHLD phase duration = 2 × HCLK clock cycle

...

1111: ADDHLD phase duration = 15 × HCLK clock cycles (default value after reset)

For each access mode address-hold phase duration, refer to the respective figure ([Figure 35](#) to [Figure 47](#)).

*Note: In synchronous accesses, this value is not used, the address hold phase is always 1 memory clock period duration.*

**Bits 3:0 ADDSET[3:0]: Address setup phase duration**

These bits are written by software to define the duration of the *address setup* phase (refer to [Figure 35](#) to [Figure 47](#)), used in SRAMs, ROMs, asynchronous NOR flash and PSRAM:

0000: ADDSET phase duration = 0 × HCLK clock cycle

...

1111: ADDSET phase duration = 15 × HCLK clock cycles (default value after reset)

For each access mode address setup phase duration, refer to the respective figure ([Figure 35](#) to [Figure 47](#)).

*Note: In synchronous accesses, this value is don't care.*

*In Muxed mode or mode D, the minimum value for ADDSET is 1.*

*In mode 1 and PSRAM memory, the minimum value for ADDSET is 1.*

**Note:**

*PSRAMs (CRAMs) have a variable latency due to internal refresh. Therefore these memories issue the NWAIT signal during the whole latency phase to prolong the latency as needed.*

*With PSRAMs (CRAMs) the filled DATLAT must be set to 0, so that the FMC exits its latency phase soon and starts sampling NWAIT from memory, then starts to read or write when the memory is ready.*

*This method can be used also with the latest generation of synchronous flash memories that issue the NWAIT signal, unlike older flash memories (check the datasheet of the specific flash memory being used).*

### SRAM/NOR-flash write timing registers x (FMC\_BWTRx)

Address offset: 0x104 + 0x8 \* (x - 1), (x = 1 to 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[1:0]		Res	Res	Res	Res	Res	Res	Res	Res	BUSTURN[3:0]					
		rw	rw									rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
DATAST[7:0]								ADDHLD[3:0]				ADDSET[3:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		

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

Bits 29:28 **ACCMOD[1:0]**: Access mode.

Specifies the asynchronous access modes as shown in the next timing diagrams. These bits are taken into account only when the EXTMOD bit in the FMC\_BCRx register is 1.

- 00: Access mode A
- 01: Access mode B
- 10: Access mode C
- 11: Access mode D

Bits 27:20 Reserved, must be kept at reset value.

Bits 19:16 **BUSTURN[3:0]**: Bus turnaround phase duration

The programmed bus turnaround delay is inserted between an asynchronous write transfer and any other asynchronous /synchronous read or write transfer to or from a static bank. The bank can be the same or different in case of read, in case of write the bank can be different expect for muxed or mode D.

In some cases, whatever the programmed BUSTURN values, the bus turnaround delay is fixed as follows:

- The bus turnaround delay is not inserted between two consecutive asynchronous write transfers to the same static memory bank except for muxed and D modes.
- There is a bus turnaround delay of 2 HCLK clock cycle between:
  - Two consecutive synchronous writes (burst or single) to the same bank.
  - A synchronous write (burst or single) transfer and an asynchronous write or read transfer to or from static memory bank.
- There is a bus turnaround delay of 3 HCLK clock cycle between:
  - Two consecutive synchronous writes (burst or single) to different static bank.
  - A synchronous write (burst or single) transfer and a synchronous read from the same or a different bank.

0000: BUSTURN phase duration = 0 HCLK clock cycle added

...

1111: BUSTURN phase duration = 15 HCLK clock cycles added (default value after reset)

Bits 15:8 **DATAST[7:0]**: Data-phase duration.

These bits are written by software to define the duration of the data phase (refer to [Figure 35](#) to [Figure 47](#)), used in asynchronous SRAM, PSRAM and NOR flash memory accesses:

0000 0000: Reserved

0000 0001: DATAST phase duration = 1 × HCLK clock cycles

0000 0010: DATAST phase duration = 2 × HCLK clock cycles

...

1111 1111: DATAST phase duration = 255 × HCLK clock cycles (default value after reset)

Bits 7:4 **ADDHLD[3:0]**: Address-hold phase duration.

These bits are written by software to define the duration of the *address hold* phase (refer to [Figure 44](#) to [Figure 47](#)), used in asynchronous multiplexed accesses:

0000: Reserved

0001: ADDHLD phase duration = 1 × HCLK clock cycle

0010: ADDHLD phase duration = 2 × HCLK clock cycle

...

1111: ADDHLD phase duration = 15 × HCLK clock cycles (default value after reset)

*Note: In synchronous NOR flash accesses, this value is not used, the address hold phase is always 1 flash clock period duration.*

Bits 3:0 **ADDSET[3:0]**: Address setup phase duration.

These bits are written by software to define the duration of the *address setup* phase in HCLK cycles (refer to [Figure 35](#) to [Figure 47](#)), used in asynchronous accesses:

0000: ADDSET phase duration = 0 × HCLK clock cycle

...

1111: ADDSET phase duration = 15 × HCLK clock cycles (default value after reset)

*Note: In synchronous accesses, this value is not used, the address setup phase is always 1 flash clock period duration. In muxed mode, the minimum ADDSET value is 1.*

## 13.7 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 13.7.7](#)). The programmable memory parameters include access timings (shown in [Table 83](#)) and ECC configuration.

**Table 83. 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

### 13.7.1 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 84. 8-bit NAND flash**

FMC signal name	I/O	Function
A[17]	O	NAND flash address latch enable (ALE) signal
A[16]	O	NAND flash command latch enable (CLE) signal

**Table 84. 8-bit NAND flash (continued)**

FMC signal name	I/O	Function
D[7:0]	I/O	8-bit multiplexed, bidirectional address/data bus
NCE	O	Chip select
NOE(= NRE)	O	Output enable (memory signal name: read enable, NRE)
NWE	O	Write enable
NWAIT/INT	I	NAND flash ready/busy input signal to the FMC

Theoretically, there is no capacity limitation as the FMC can manage as many address cycles as needed.

### 16-bit NAND flash memory

**Table 85. 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.*

### 13.7.2 NAND flash supported memories and transactions

[Table 86](#) shows the supported devices, access modes and transactions. Transactions not allowed (or not supported) by the NAND flash controller are shown in gray.

**Table 86. 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

**Table 86. Supported memories and transactions (continued)**

Device	Mode	R/W	AHB data size	Memory data size	Allowed/not allowed	Comments
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

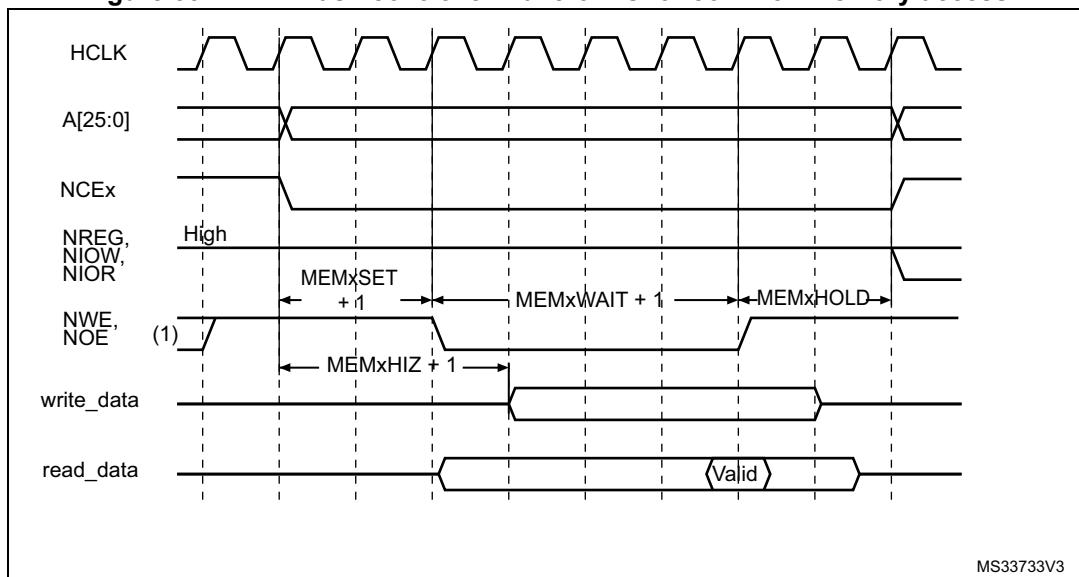
### 13.7.3 Timing diagrams for NAND flash memory

The NAND flash memory bank is managed through a set of registers:

- Control register: FMC\_PCR
- Interrupt status register: FMC\_SR
- ECC register: FMC\_ECCR
- Timing register for Common memory space: FMC\_PMEM
- Timing register for Attribute memory space: FMC\_PATT

Each timing configuration register contains three parameters used to define number of HCLK cycles for the three phases of any NAND flash access, plus one parameter that defines the timing for starting driving the data bus when a write access is performed.

*Figure 53* shows the timing parameter definitions for common memory accesses, knowing that Attribute memory space access timings are similar.

**Figure 53. NAND flash controller waveforms for common memory access**

1. NOE remains high (inactive) during write accesses. NWE remains high (inactive) during read accesses.
2. For write access, the hold phase delay is (MEMHOLD) HCLK cycles and for read access is (MEMHOLD + 2) HCLK cycles.

### 13.7.4 NAND flash operations

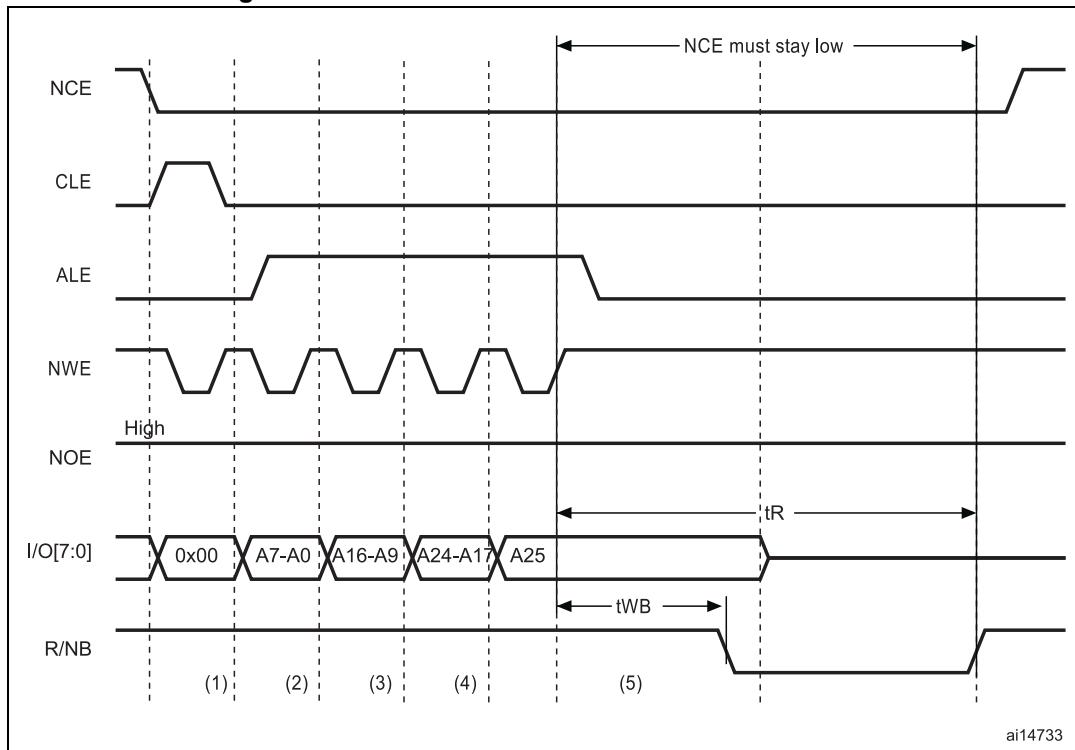
The command latch enable (CLE) and address latch enable (ALE) signals of the NAND flash memory device are driven by address signals from the FMC controller. This means that to send a command or an address to the NAND flash memory, the CPU has to perform a write to a specific address in its memory space.

A typical page read operation from the NAND flash device requires the following steps:

1. Program and enable the corresponding memory bank by configuring the FMC\_PCR and FMC\_PMEM (and for some devices, FMC\_PATT, see [Section 13.7.5: 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 13.5.2: NAND flash memory address mapping](#) for timing configuration).
2. The CPU performs a byte write to the common memory space, with data byte equal to one flash command byte (for example 0x00 for Samsung NAND flash devices). The LE input of the NAND flash memory is active during the write strobe (low pulse on NWE), thus the written byte is interpreted as a command by the NAND flash memory. Once the command is latched by the memory device, it does not need to be written again for the following page read operations.
3. The CPU can send the start address (STARTAD) for a read operation by writing four bytes (or three for smaller capacity devices), STARTAD[7:0], STARTAD[16:9], STARTAD[24:17] and finally STARTAD[25] (for 64 Mb x 8 bit NAND flash memories) in the common memory or attribute space. The ALE input of the NAND flash device is active during the write strobe (low pulse on NWE), thus the written bytes are interpreted as the start address for read operations. Using the attribute memory space makes it possible to use a different timing configuration of the FMC, which can be used to implement the prewait functionality needed by some NAND flash memories (see details in [Section 13.7.5: NAND flash prewait functionality](#)).
4. The controller waits for the NAND flash memory to be ready (R/NB signal high), before starting a new access to the same or another memory bank. While waiting, the controller holds the NCE signal active (low).
5. The CPU can then perform byte read operations from the common memory space to read the NAND flash page (data field + Spare field) byte by byte.
6. The next NAND flash page can be read without any CPU command or address write operation. This can be done in three different ways:
  - by simply performing the operation described in step 5
  - a new random address can be accessed by restarting the operation at step 3
  - a new command can be sent to the NAND flash device by restarting at step 2

### 13.7.5 NAND flash prewait functionality

Some NAND flash devices require that, after writing the last part of the address, the controller waits for the R/NB signal to go low. (see [Figure 54](#)).

**Figure 54. Access to non 'CE don't care' NAND-flash**

1. CPU wrote byte 0x00 at address 0x7001 0000.
2. CPU wrote byte A7~A0 at address 0x7002 0000.
3. CPU wrote byte A16~A9 at address 0x7002 0000.
4. CPU wrote byte A24~A17 at address 0x7002 0000.
5. CPU wrote byte A25 at address 0x7802 0000: FMC performs a write access using FMC\_PATT timing definition, where  $\text{ATTHOLD} \geq 7$  (providing that  $(7+1) \times \text{HCLK} = 112 \text{ ns} > t_{WB} \text{ max}$ ). This guarantees that NCE remains low until R/NB goes low and high again (only requested for NAND flash memories where NCE is not don't care).

When this functionality is required, it can be ensured by programming the MEMHOLD value to meet the  $t_{WB}$  timing. However any CPU read access to the NAND flash memory has a hold delay of  $(\text{MEMHOLD} + 2)$  HCLK cycles and CPU write access has a hold delay of  $(\text{MEMHOLD})$  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.

### 13.7.6 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 must 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.

### 13.7.7 NAND flash controller registers

#### NAND flash control registers (FMC\_PCR)

Address offset: 0x80

Reset value: 0x0000 0018

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ECCPS[2:0]	TAR3	
													rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAR[2:0]			TCLR[3:0]				Res.	Res.	ECCEN	PWID[1:0]		PTYP	PBKEN	PWAITEN	Res.
rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	

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

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 is 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 is set.*

### Common memory space timing register (FMC\_PMEM)

Address offset: 0x88

Reset value: 0xFCFC FCFC

The FMC\_PMEM read/write register contains the timing information for NAND flash memory bank. This information is used to access either the common memory space of the NAND flash for command, address write access and data read/write access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEMHIZ[7:0]								MEMHOLD[7:0]							
<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
MEMWAIT[7:0]								MEMSET[7:0]							
<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 31:24 **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 register (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 13.7.5: NAND flash prewait functionality](#)).

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

Bits 31:24 **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: 1 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 13.7.6: 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 must be cleared after being read by setting the ECCEN bit to 0. To compute a new data block, the ECCEN bit must be set to 1.

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

Bits 31:0 **ECC[31:0]**: ECC result

This field contains the value computed by the ECC computation logic. [Table 87](#) describes the contents of these bitfields.

**Table 87. 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]

## 13.8 SDRAM controller

### 13.8.1 SDRAM controller main features

The main features of the SDRAM controller are the following:

- Two SDRAM banks with independent configuration
- 8-bit, 16-bit, 32-bit data bus width
- 13-bits Address Row, 11-bits Address Column, 4 internal banks: 4x16Mx32bit (256 MB), 4x16Mx16bit (128 MB), 4x16Mx8bit (64 MB)
- Word, half-word, byte access
- SDRAM clock can be HCLK/2 or HCLK/3
- Automatic row and bank boundary management
- Multibank ping-pong access
- Programmable timing parameters
- Automatic Refresh operation with programmable Refresh rate
- Self-refresh mode
- Power-down mode
- SDRAM power-up initialization by software
- CAS latency of 1,2,3
- Cacheable Read FIFO with depth of 6 lines x32-bit (6 x14-bit address tag)

### 13.8.2 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 88. SDRAM signals**

SDRAM signal	I/O type	Description	Alternate function
SDCLK	O	SDRAM clock	-
SDCKE[1:0]	O	SDCKE0: SDRAM Bank 1 Clock Enable SDCKE1: SDRAM Bank 2 Clock Enable	-
SDNE[1:0]	O	SDNE0: SDRAM Bank 1 Chip Enable SDNE1: SDRAM Bank 2 Chip Enable	-
A[12:0]	O	Address	FMC_A[12:0]
D[31:0]	I/O	Bidirectional data bus	FMC_D[31:0]
BA[1:0]	O	Bank Address	FMC_A[15:14]
NRAS	O	Row Address Strobe	-
NCAS	O	Column Address Strobe	-
SDNWE	O	Write Enable	-
NBL[3:0]	O	Output Byte Mask for write accesses (memory signal name: DQM[3:0])	FMC_NBL[3:0]

### 13.8.3 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.

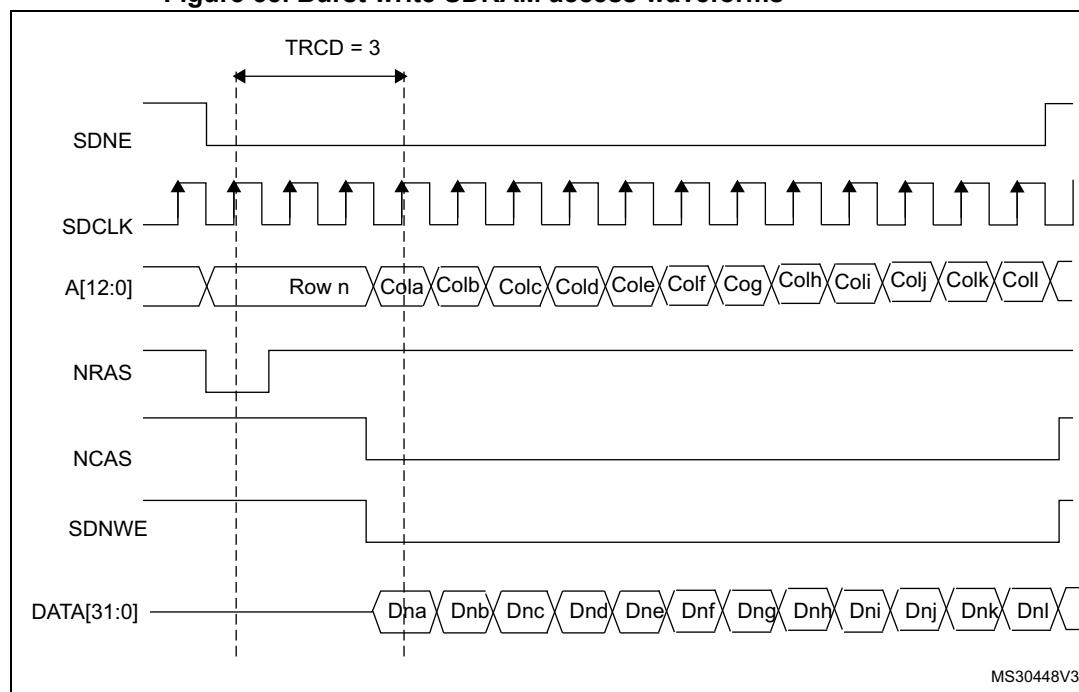
**Note:** If two SDRAM devices are connected to the FMC, all the accesses performed at the same time to both devices by the Command Mode register (Load Mode Register command) are issued using the timing parameters configured for SDRAM Bank 1 (TMRD and TRAS timings) in the FMC\_SDTR1 register.

### 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 55. Burst write SDRAM access waveforms**



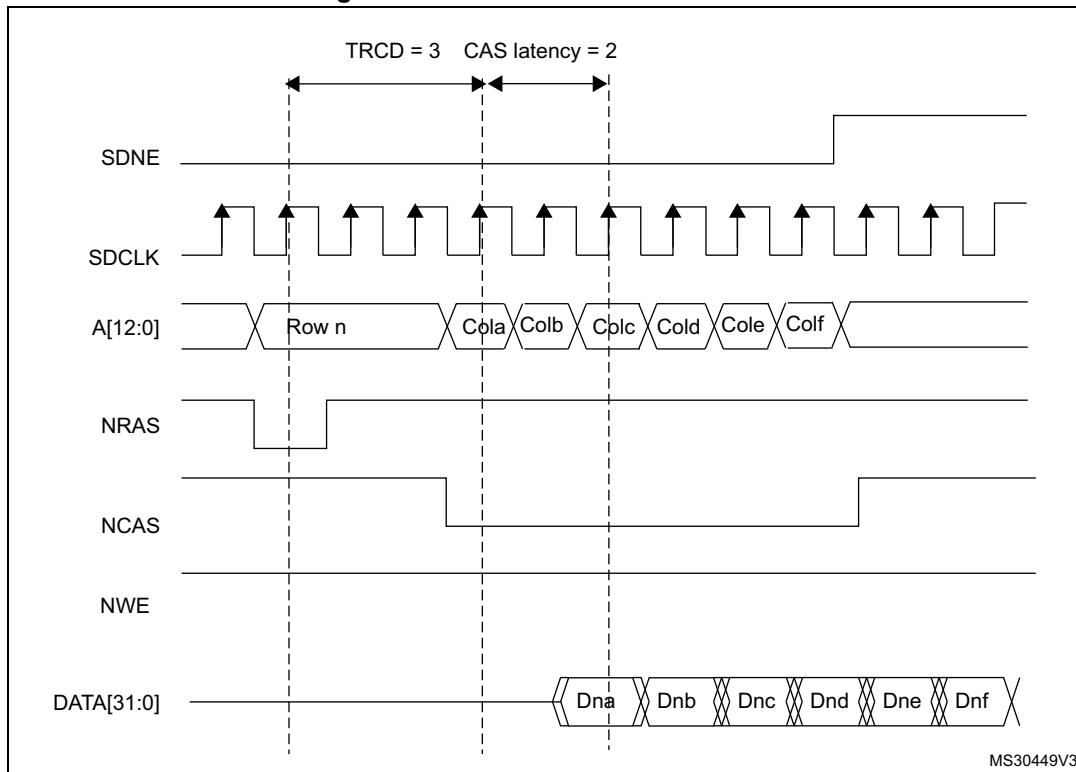
The SDRAM controller always checks the next access.

- If the next access is in the same row or in another active row, the write operation is carried out,
- if the next access targets another row (not active), the SDRAM controller generates a precharge command, activates the new row and initiates a write command.

### SDRAM controller read cycle

The SDRAM controller accepts single and burst read requests and translates them into single memory accesses. In both cases, the SDRAM controller keeps track of the active row in each bank to be able to perform consecutive read accesses in different banks (Multibank ping-pong access).

Figure 56. Burst read SDRAM access



The FMC SDRAM controller features a Cacheable read FIFO (6 lines x 32 bits). It is used to store data read in advance during the CAS latency period and the RPIPE delay following the below formula. The RBURST bit must be set in the FMC\_SDCR1 register to anticipate the next read access.

$$\text{Number of anticipated data} = \text{CAS latency} + 1 + (\text{RPIPE delay})/2$$

Examples:

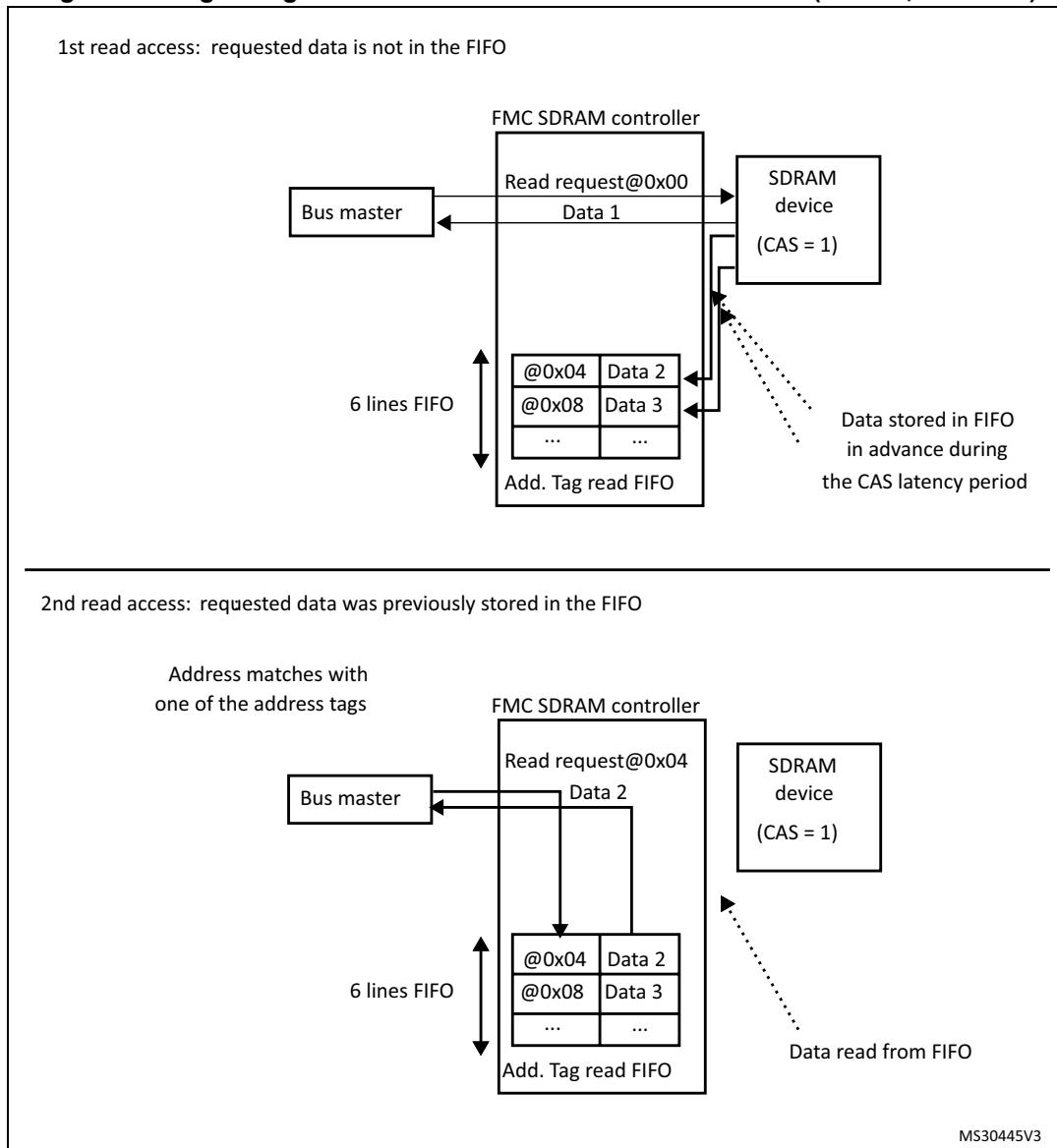
- CAS latency = 3, RPIPE delay = 0: Four data (not committed) are stored in the FIFO.
- CAS latency = 3, RPIPE delay = 2: Five data (not committed) are stored in the FIFO.

The read FIFO features a 14-bit address tag to each line to identify its content: 11 bits for the column address, 2 bits to select the internal bank and the active row, and 1 bit to select the SDRAM device

When the end of the row is reached in advance during an AHB burst read, the data read in advance (not committed) are not stored in the read FIFO. For single read access, data are correctly stored in the FIFO.

Each time a read request occurs, the SDRAM controller checks:

- If the address matches one of the address tags, data are directly read from the FIFO and the corresponding address tag/ line content is cleared and the remaining data in the FIFO are compacted to avoid empty lines.
- Otherwise, a new read command is issued to the memory and the FIFO is updated with new data. If the FIFO is full, the older data are lost.

**Figure 57. Logic diagram of Read access with RBURST bit set (CAS=1, RPIPE=0)**

MS30445V3

During a write access or a Precharge command, the read FIFO is flushed and ready to be filled with new data.

After the first read request, if the current access was not performed to a row boundary, the SDRAM controller anticipates the next read access during the CAS latency period and the RPIPE delay (if configured). This is done by incrementing the memory address. The following condition must be met:

- RBURST control bit should be set to '1' in the FMC\_SDCR1 register.

The address management depends on the next AHB request:

- Next AHB request is sequential (AHB Burst)  
In this case, the SDRAM controller increments the address.
- Next AHB request is not sequential
  - If the new read request targets the same row or another active row, the new address is passed to the memory and the master is stalled for the CAS latency period, waiting for the new data from memory.
  - If the new read request does not target an active row, the SDRAM controller generates a Precharge command, activates the new row, and initiates a read command.

If the RURST is reset, the read FIFO is not used.

### Row and bank boundary management

When a read or write access crosses a row boundary, if the next read or write access is sequential and the current access was performed to a row boundary, the SDRAM controller executes the following operations:

1. Precharge of the active row,
2. Activation of the new row
3. Start of a read/write command.

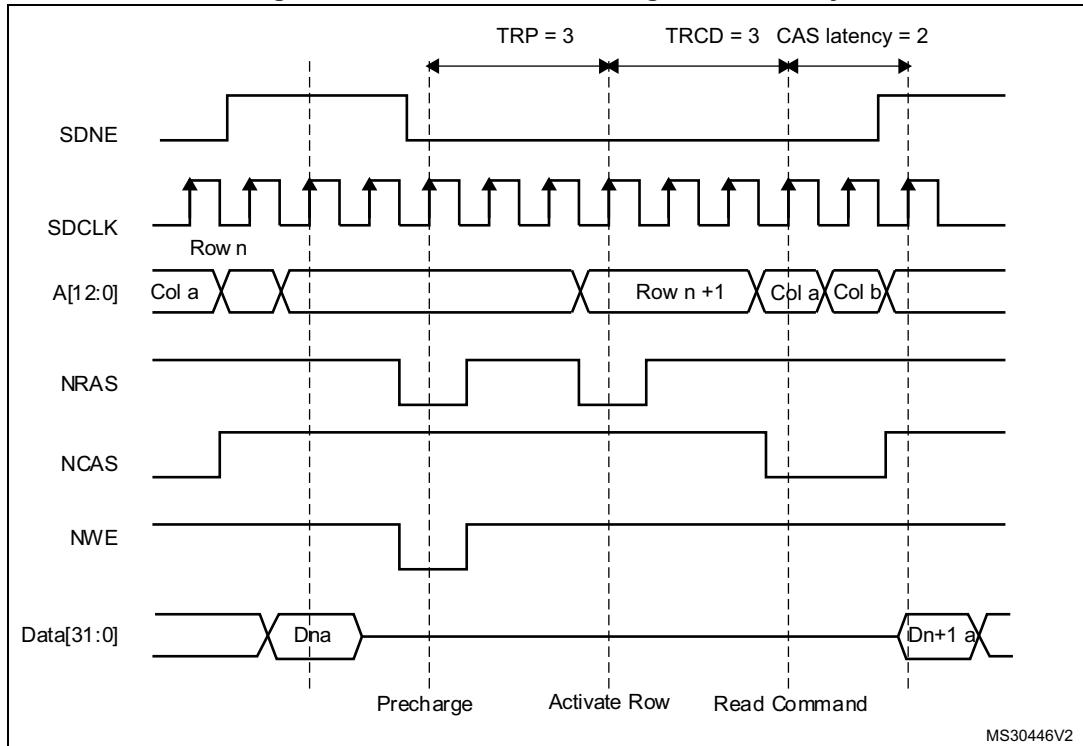
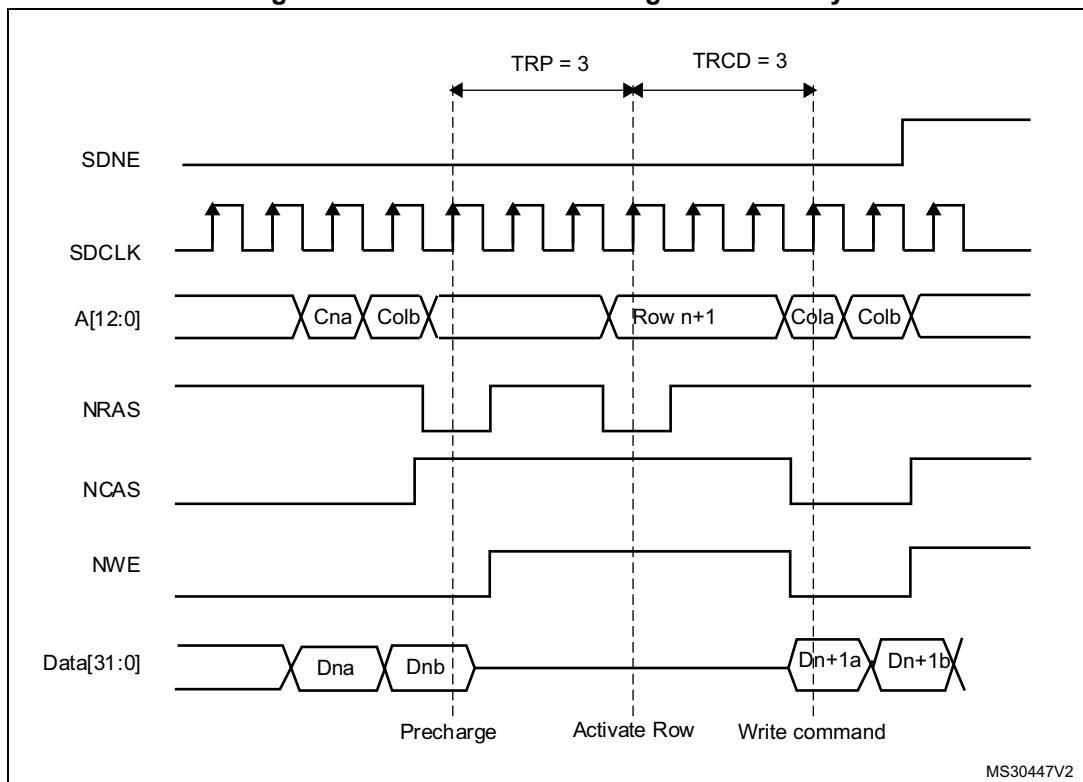
At a row boundary, the automatic activation of the next row is supported for all columns and data bus width configurations.

If necessary, the SDRAM controller inserts additional clock cycles between the following commands:

- Between Precharge and Active commands to match TRP parameter (only if the next access is in a different row in the same bank),
- Between Active and Read commands to match the TRCD parameter.

These parameters are defined into the FMC\_SDTRx register.

Refer to [Figure 55](#) and [Figure 56](#) for read and burst write access crossing a row boundary.

**Figure 58. Read access crossing row boundary****Figure 59. Write access crossing row boundary**

If the next access is sequential and the current access crosses a bank boundary, the SDRAM controller activates the first row in the next bank and initiates a new read/write command. Two cases are possible:

- If the current bank is not the last one, the active row in the new bank must be precharged. At a bank boundary, the automatic activation of the next row is supported for all rows/columns and data bus width configuration.
- If the current bank is the last one, the automatic activation of the next row is supported only when addressing 13-bit rows, 11-bit columns, 4 internal banks and 32-bit data bus SDRAM devices. Otherwise, the SDRAM address range is violated and an AHB error is generated.
- In case of 13-bit row address, 11-bit column address, 4 internal banks and bus width 32-bit SDRAM memories, at boundary bank, the SDRAM controller continues to read/write from the second SDRAM device (assuming it has been initialized):
  - a) The SDRAM controller activates the first row (after precharging the active row, if there is already an active row in the first internal bank, and initiates a new read/write command).
  - b) If the first row is already activated, the SDRAM controller just initiates a read/write command.

#### SDRAM controller refresh cycle

The Auto-refresh command is used to refresh the SDRAM device content. The SDRAM controller periodically issues auto-refresh commands. An internal counter is loaded with the COUNT value in the register FMC\_SDRTR. This value defines the number of memory clock cycles between the refresh cycles (refresh rate). When this counter reaches zero, an internal pulse is generated.

If a memory access is ongoing, the auto-refresh request is delayed. However, if the memory access and the auto-refresh requests are generated simultaneously, the auto-refresh request takes precedence.

If the memory access occurs during an auto-refresh operation, the request is buffered and processed when the auto-refresh is complete.

If a new auto-refresh request occurs while the previous one was not served, the RE (Refresh Error) bit is set in the Status register. An Interrupt is generated if it has been enabled (REIE = '1').

If SDRAM lines are not in idle state (not all row are closed), the SDRAM controller generates a PALL (Precharge ALL) command before the auto-refresh.

If the Auto-refresh command is generated by the FMC\_SDCMR Command Mode register (Mode bits = '011'), a PALL command (Mode bits = '010') must be issued first.

#### 13.8.4 Low-power modes

Two low-power modes are available:

- Self-refresh mode
  - The auto-refresh cycles are performed by the SDRAM device itself to retain data without external clocking.
- Power-down mode
  - The auto-refresh cycles are performed by the SDRAM controller.

### Self-refresh mode

This mode is selected by setting the MODE bits to '101' and by configuring the Target Bank bits (CTB1 and/or CTB2) in the FMC\_SDCMR register.

The SDRAM clock stops running after a TRAS delay and the internal refresh timer stops counting only if one of the following conditions is met:

- A Self-refresh command is issued to both devices
- One of the devices is not activated (SDRAM bank is not initialized).

Before entering Self-Refresh mode, the SDRAM controller automatically issues a PALL command.

If the Write data FIFO is not empty, all data are sent to the memory before activating the Self-refresh mode and the BUSY status flag remains set.

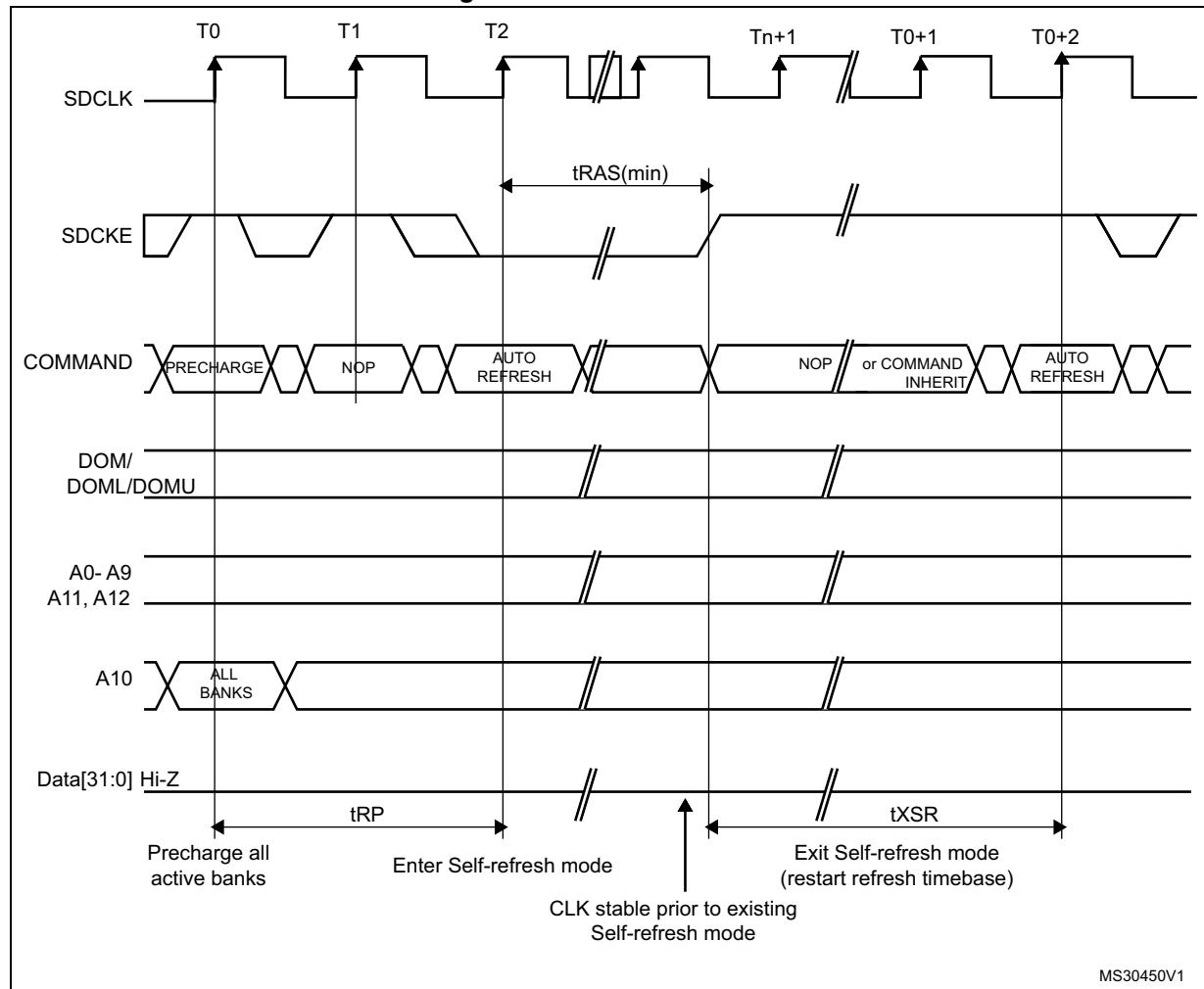
In Self-refresh mode, all SDRAM device inputs become don't care except for SDCKE which remains low.

The SDRAM device must remain in Self-refresh mode for a minimum period of time of TRAS and can remain in Self-refresh mode for an indefinite period beyond that. To guarantee this minimum period, the BUSY status flag remains high after the Self-refresh activation during a TRAS delay.

As soon as an SDRAM device is selected, the SDRAM controller generates a sequence of commands to exit from Self-refresh mode. After the memory access, the selected device remains in Normal mode.

To exit from Self-refresh, the MODE bits must be set to '000' (Normal mode) and the Target Bank bits (CTB1 and/or CTB2) must be configured in the FMC\_SDCMR register.

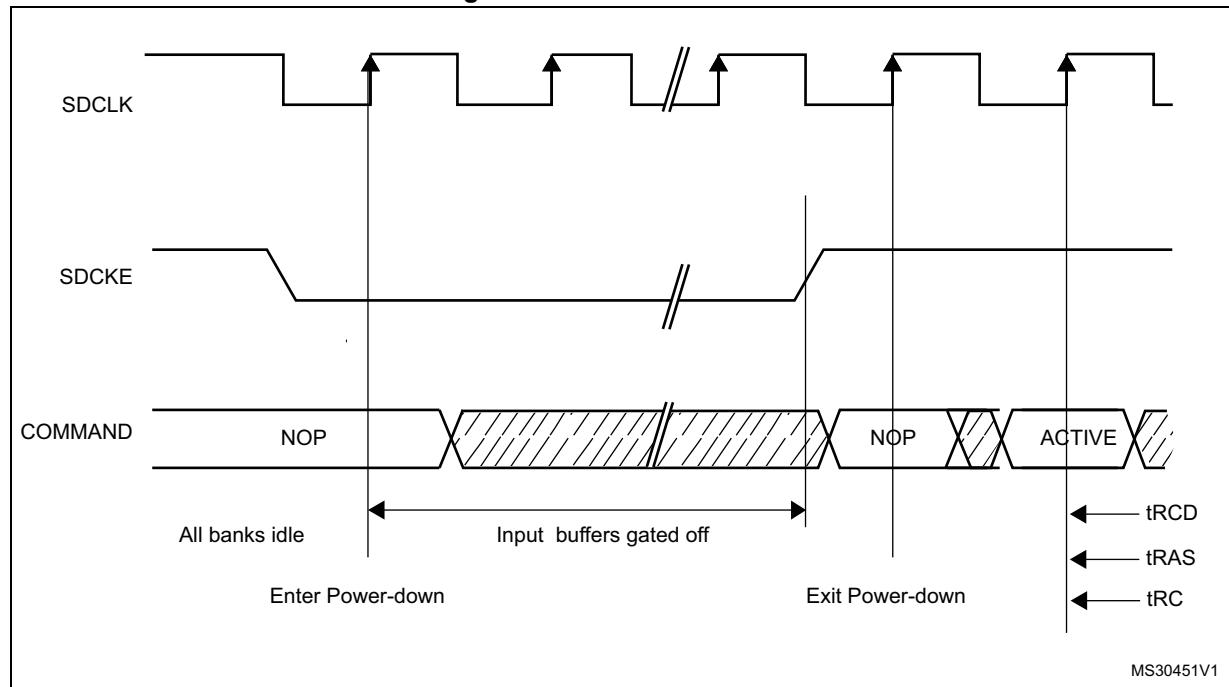
Figure 60. Self-refresh mode



### Power-down mode

This mode is selected by setting the MODE bits to '110' and by configuring the Target Bank bits (CTB1 and/or CTB2) in the FMC\_SDCMR register.

Figure 61. Power-down mode



If the Write data FIFO is not empty, all data are sent to the memory before activating the Power-down mode.

As soon as an SDRAM device is selected, the SDRAM controller exits from the Power-down mode. After the memory access, the selected SDRAM device remains in Normal mode.

During Power-down mode, all SDRAM device input and output buffers are deactivated except for the SDCKE which remains low.

The SDRAM device cannot remain in Power-down mode longer than the refresh period and cannot perform the Auto-refresh cycles by itself. Therefore, the SDRAM controller carries out the refresh operation by executing the operations below:

1. Exit from Power-down mode and drive the SDCKE high
2. Generate the PALL command only if a row was active during Power-down mode
3. Generate the auto-refresh command
4. Drive SDCKE low again to return to Power-down mode.

To exit from Power-down mode, the MODE bits must be set to '000' (Normal mode) and the Target Bank bits (CTB1 and/or CTB2) must be configured in the FMC\_SDCMR register.

### 13.8.5 SDRAM controller registers

#### SDRAM control register x (FMC\_SDCRx)

Address offset: 0x140+ 0x4\* (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[1:0]	WP	CAS[1:0]	NB	MWID[1:0]	NR[1:0]	NC[1:0]						
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

##### Bits 14:13 RPIPE[1:0]: Read pipe

These bits define the delay, in HCLK clock cycles, for reading data after CAS latency.

- 00: No HCLK clock cycle delay
- 01: One HCLK clock cycle delay
- 10: Two HCLK clock cycle delay
- 11: reserved.

*Note:* The corresponding bits in the FMC\_SDCR2 register is read only.

##### Bit 12 RBURST: Burst read

This bit enables Burst read mode. The SDRAM controller anticipates the next read commands during the CAS latency and stores data in the Read FIFO.

- 0: single read requests are not managed as bursts
- 1: single read requests are always managed as bursts

*Note:* The corresponding bit in the FMC\_SDCR2 register is don't care.

##### Bits 11:10 SDCLK[1:0]: SDRAM clock configuration

These bits define the SDRAM clock period for both SDRAM banks and allow disabling the clock before changing the frequency. In this case the SDRAM must be re-initialized.

- 00: SDCLK clock disabled
- 10: SDCLK period = 2 x HCLK periods
- 11: SDCLK period = 3 x HCLK periods
- Others: Reserved, must not be used

*Note:* The corresponding bits in the FMC\_SDCR2 register are don't care.

##### Bit 9 WP: Write protection

This bit enables write mode access to the SDRAM bank.

- 0: Write accesses allowed
- 1: Write accesses ignored

##### Bits 8:7 CAS[1:0]: CAS Latency

This bits sets the SDRAM CAS latency in number of memory clock cycles

- 00: reserved.
- 01: 1 cycle
- 10: 2 cycles
- 11: 3 cycles

**Bit 6 NB:** Number of internal banks

This bit sets the number of internal banks.

0: Two internal Banks

1: Four internal Banks

**Bits 5:4 MWID[1:0]:** Memory data bus width.

These bits define the memory device width.

00: 8 bits

01: 16 bits

10: 32 bits

11: reserved.

**Bits 3:2 NR[1:0]:** Number of row address bits

These bits define the number of bits of a row address.

00: 11 bit

01: 12 bits

10: 13 bits

11: reserved.

**Bits 1:0 NC[1:0]:** Number of column address bits

These bits define the number of bits of a column address.

00: 8 bits

01: 9 bits

10: 10 bits

11: 11 bits.

**Note:** Before modifying the RBURST or RPIPE settings or disabling the SDCLK clock, the user must first send a PALL command to make sure ongoing operations are complete.

**SDRAM timing register x (FMC\_SDTRx)**

Address offset:  $0x148 + 0x4 * (x - 1)$ , ( $x = 1,2$ )

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[3:0]				TRP[3:0]				TWR[3:0]			
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRC[3:0]				TRAS[3:0]				TXSR[3:0]				TMRD[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

**Bits 27:24 TRCD[3:0]:** Row to column delay

These bits define the delay between the Activate command and a Read/Write command in number of memory clock cycles.

0000: 1 cycle.

0001: 2 cycles

....

1111: 16 cycles

Bits 23:20 **TRP[3:0]:** Row precharge delay

These bits define the delay between a Precharge command and another command in number of memory clock cycles. The TRP timing is only configured in the FMC\_SDTR1 register. If two SDRAM devices are used, the TRP must be programmed with the timing of the slowest device.

0000: 1 cycle

0001: 2 cycles

....

1111: 16 cycles

*Note: The corresponding bits in the FMC\_SDTR2 register are don't care.*

Bits 19:16 **TWR[3:0]:** Recovery delay

These bits define the delay between a Write and a Precharge command in number of memory clock cycles.

0000: 1 cycle

0001: 2 cycles

....

1111: 16 cycles

*Note: TWR must be programmed to match the write recovery time ( $t_{WR}$ ) defined in the SDRAM datasheet, and to guarantee that:*

$TWR \geq TRAS - TRCD$  and  $TWR \geq TRC - TRCD - TRP$

*Example: TRAS= 4 cycles, TRCD= 2 cycles. So, TWR >= 2 cycles. TWR must be programmed to 0x1.*

*If two SDRAM devices are used, the FMC\_SDTR1 and FMC\_SDTR2 must be programmed with the same TWR timing corresponding to the slowest SDRAM device.*

*If only one SDRAM device is used, the TWR timing must be kept at reset value (0xF) for the not used bank.*

Bits 15:12 **TRC[3:0]:** Row cycle delay

These bits define the delay between the Refresh command and the Activate command, as well as the delay between two consecutive Refresh commands. It is expressed in number of memory clock cycles. The TRC timing is only configured in the FMC\_SDTR1 register. If two SDRAM devices are used, the TRC must be programmed with the timings of the slowest device.

0000: 1 cycle

0001: 2 cycles

....

1111: 16 cycles

*Note: TRC must match the TRC and TRFC (Auto Refresh period) timings defined in the SDRAM device datasheet.*

*Note: The corresponding bits in the FMC\_SDTR2 register are don't care.*

Bits 11:8 **TRAS[3:0]:** Self refresh time

These bits define the minimum Self-refresh period in number of memory clock cycles.

0000: 1 cycle

0001: 2 cycles

....

1111: 16 cycles

**Bits 7:4 TXSR[3:0]: Exit Self-refresh delay**

These bits define the delay from releasing the Self-refresh command to issuing the Activate command in number of memory clock cycles.

0000: 1 cycle  
0001: 2 cycles  
....

1111: 16 cycles

*Note: If two SDRAM devices are used, the FMC\_SDTR1 and FMC\_SDTR2 must be programmed with the same TXSR timing corresponding to the slowest SDRAM device.*

**Bits 3:0 TMRD[3:0]: Load Mode Register to Active**

These bits define the delay between a Load Mode Register command and an Active or Refresh command in number of memory clock cycles.

0000: 1 cycle  
0001: 2 cycles  
....

1111: 16 cycles

*Note: If two SDRAM devices are connected, all the accesses performed simultaneously to both devices by the Command Mode register (Load Mode Register command) are issued using the timing parameters configured for Bank 1 (TMRD and TRAS timings) in the FMC\_SDTR1 register.*

*The TRP and TRC timings are only configured in the FMC\_SDTR1 register. If two SDRAM devices are used, the TRP and TRC timings must be programmed with the timings of the slowest device.*

### SDRAM Command Mode register (FMC\_SDCMR)

Address offset: 0x150

Reset value: 0x0000 0000

This register contains the command issued when the SDRAM device is accessed. This register is used to initialize the SDRAM device, and to activate the Self-refresh and the Power-down modes. As soon as the MODE field is written, the command is 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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MRD[12:7]													
										rw	rw	rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
MRD[6:0]								NRFS[3:0]				CTB1	CTB2	MODE[2:0]									
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	w	w	w	w	w								

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 is 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 is 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 is 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 is ignored.*

*Note: If only one SDRAM bank is used and a command is issued with it's associated CTB bit set, the other CTB bit of the the unused bank must be kept to 0.*

**SDRAM refresh timer register (FMC\_SDRTR)**

Address offset:0x154

Reset value: 0x0000 0000

This register sets the refresh rate in number of SDCLK clock cycles between the refresh cycles by configuring the Refresh Timer Count value.

$$\text{Refresh rate} = (\text{COUNT} + 1) \times \text{SDRAM clock frequency}$$

$$\text{COUNT} = (\text{SDRAM refresh period} / \text{Number of rows}) - 20$$

**Example**

$$\text{Refresh rate} = 64 \text{ ms} / (8196 \text{ rows}) = 7.81 \mu\text{s}$$

where 64 ms is the SDRAM refresh period.

$$7.81\mu\text{s} \times 60\text{MHz} = 468.6$$

The refresh rate must be increased by 20 SDRAM clock cycles (as in the above example) to obtain a safe margin if an internal refresh request occurs when a read request has been accepted. It corresponds to a COUNT value of '0000111000000' (448).

This 13-bit field is loaded into a timer which is decremented using the SDRAM clock. This timer generates a refresh pulse when zero is reached. The COUNT value must be set at least to 41 SDRAM clock cycles.

As soon as the FMC\_SDRTR register is programmed, the timer starts counting. If the value programmed in the register is '0', no refresh is carried out. This register must not be reprogrammed after the initialization procedure to avoid modifying the refresh rate.

Each time a refresh pulse is generated, this 13-bit COUNT field is reloaded into the counter.

If a memory access is in progress, the Auto-refresh request is delayed. However, if the memory access and Auto-refresh requests are generated simultaneously, the Auto-refresh takes precedence. If the memory access occurs during a refresh operation, the request is buffered to be processed when the refresh is complete.

This register is common to SDRAM bank 1 and bank 2.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	REIE	COUNT[12:0]												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[1:0]	MODES1[1:0]	RE											
										r	r	r	r	r	r

Bits 31:6 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

**13.8.6 FMC register map****Table 89. FMC register map and reset values**

Offset	Register name reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0x00	FMC_BCR1	Res.	WFDIS	CCLKEN	CBURSTRW	0	0	0	0	ASYNCWAIT	EXTMOD	WAITEN	WREN	WAITCFG	0	0	0	0	0	0	0	0	0	0	0	0									
	Reset value											0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0			
0x08	FMC_BCR2	Res.	Res.	CBURSTRW	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

Table 89. FMC register map and reset values (continued)

Offset	Register name reset value	31	30	29	28	27	CBURSTRW[0]	CPSIZE [2:0]	CBURSTRW[0]	19	18	17	16	ASYNCPWAIT[0]	ASYNCPWAIT[15]	EXTMOD[0]	EXTMOD[14]	WAITEN[0]	WAITEN[13]	WREN[1]	WREN[12]	WAITCFG[0]	WAITCFG[11]	Res.[0]	Res.[10]	MWID[1:0]	MWID[1:0]	MTYP[1:0]	MTYP[2]	MUXEN[1]	MUXEN[3]	MBKEN[0]				
0x10	FMC_BCR3	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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
	Reset value																																			
0x18	FMC_BCR4	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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
	Reset value																																			
0x04	FMC_BTR1	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	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				
0x0C	FMC_BTR2	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	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				
0x14	FMC_BTR3	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	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				
0x1C	FMC_BTR4	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	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				
0x104	FMC_BWTR1	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	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				
0x10C	FMC_BWTR2	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	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					
0x114	FMC_BWTR3	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	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]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	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				

**Table 89. FMC register map and reset values (continued)**

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 14 Quad-SPI interface (QUADSPI)

### 14.1 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.
- automatic 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 device 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.

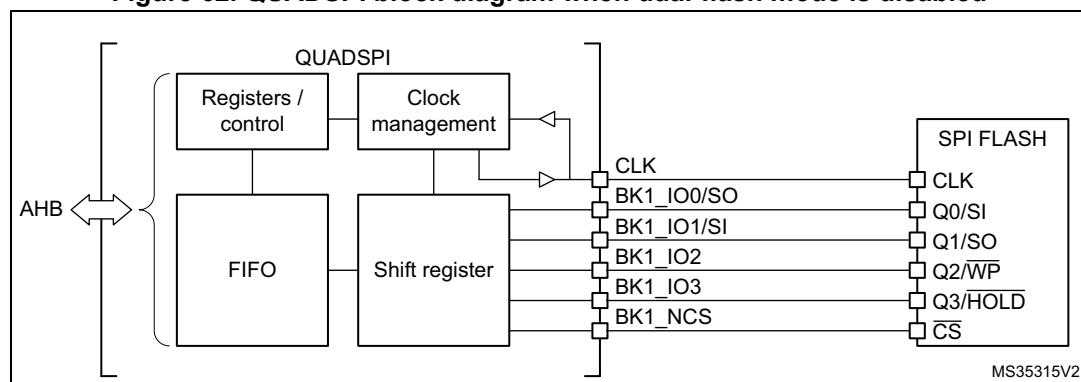
### 14.2 QUADSPI main features

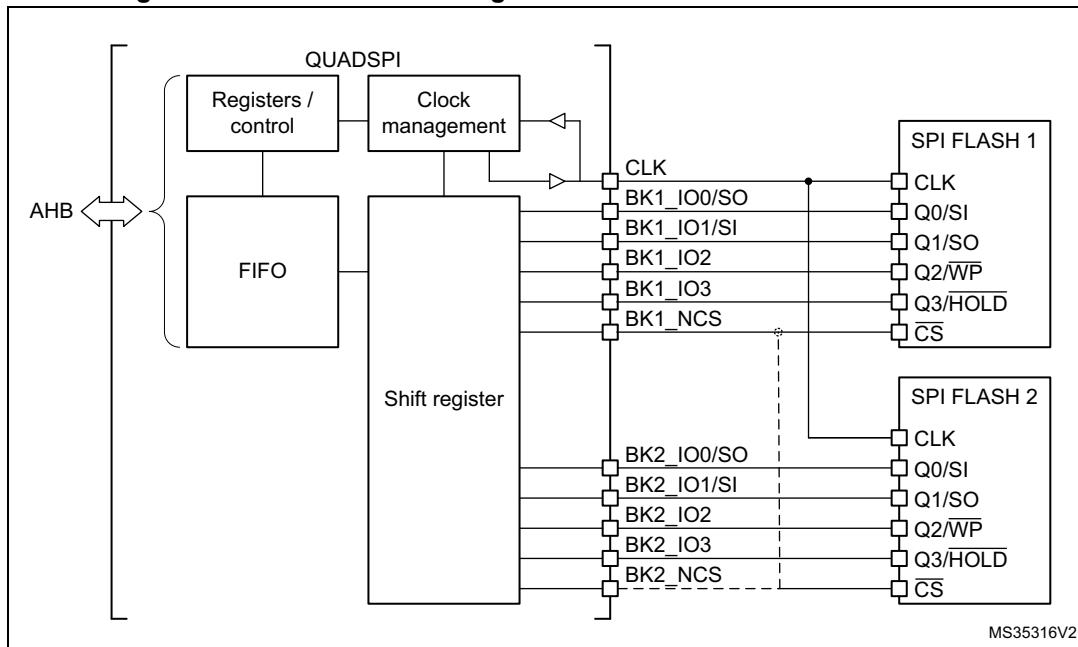
- Three functional modes: indirect, automatic status-polling, and memory-mapped
- Dual-flash mode, where 8 bits can be sent/received simultaneously by accessing two flash memories in parallel
- SDR and DDR support
- Fully programmable opcode for both indirect and memory-mapped modes
- Fully programmable frame format for both indirect and memory-mapped modes
- Integrated FIFO for reception and transmission
- 8-, 16-, and 32-bit data accesses allowed
- DMA channel for indirect mode operations
- Interrupt generation on FIFO threshold, timeout, operation complete, and access error

### 14.3 QUADSPI functional description

#### 14.3.1 QUADSPI block diagram

Figure 62. QUADSPI block diagram when dual-flash mode is disabled



**Figure 63. QUADSPI block diagram when dual-flash mode is enabled**

### 14.3.2 QUADSPI pins

The table below 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 90. QUADSPI pins**

Signal name	Signal type	Description
CLK	Digital output	Clock to FLASH 1 and FLASH 2
BK1_IO0/SO	Digital input/output	Bidirectional I/O in dual/quad modes or serial output in single mode, for FLASH 1
BK1_IO1/SI	Digital input/output	Bidirectional I/O in dual/quad modes or serial input in single mode, for FLASH 1
BK1_IO2	Digital input/output	Bidirectional I/O in quad mode, for FLASH 1
BK1_IO3	Digital input/output	Bidirectional I/O in quad mode, for FLASH 1
BK2_IO0/SO	Digital input/output	Bidirectional I/O in dual/quad modes or serial output in single mode, for FLASH 2
BK2_IO1/SI	Digital input/output	Bidirectional I/O in dual/quad modes or serial input in single mode, for FLASH 2
BK2_IO2	Digital input/output	Bidirectional I/O in quad mode, for FLASH 2
BK2_IO3	Digital input/output	Bidirectional I/O 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.

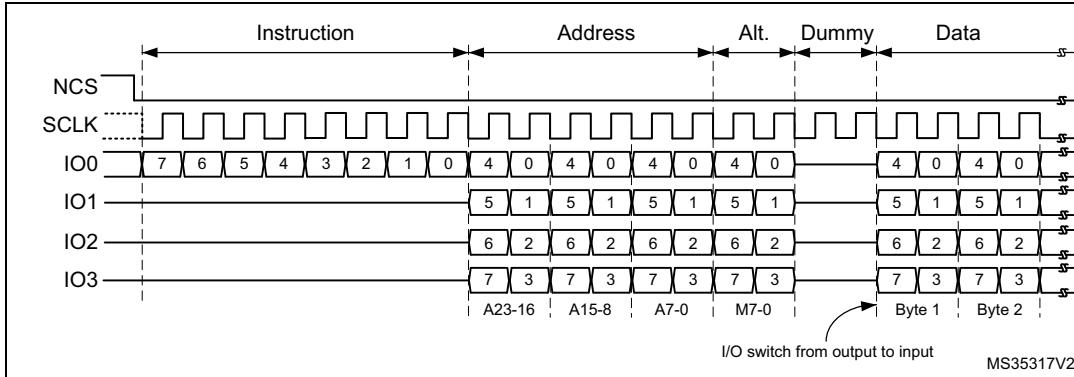
### 14.3.3 QUADSPI command sequence

The QUADSPI communicates with the flash memory using commands. Each command can include five phases: instruction, address, alternate byte, dummy, data. Any of these phases

can be configured to be skipped, but at least one of the instruction, address, alternate byte, or data phase must be present.

NCS falls before the start of each command and rises again after each command finishes.

**Figure 64. Example of read command in quad-SPI mode**



### Instruction phase

During this phase, an 8-bit instruction, configured in INSTRUCTION bitfield of QUADSPI\_CCR[7:0] register, is sent to the flash memory, specifying the type of operation to be performed.

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] bitfield 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] bitfield of QUADSPI\_CCR[13:12] register. In indirect and automatic status-polling modes, address bytes to be sent are specified in the ADDRESS[31:0] bitfield 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] bitfield 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-byte phase

In the alternate-byte 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 [1:0] bitfield 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] bitfield of QUADSPI\_CCR[15:14] register.

When ABMODE = 00, the alternate-byte 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 the dual-mode is used and only two cycles are used for the alternate bytes. In this case, the 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 must be set to 0x8A (1000\_1010).

### Dummy-cycle phase

In the dummy-cycle phase, 1-31 cycles are given without any data being sent or received, in order to give time to the flash memory 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] bitfield 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 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 status-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. In indirect-read mode the data received from the flash memory is obtained by reading 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] bitfield 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.

#### 14.3.4 QUADSPI signal interface protocol modes

##### Single-SPI mode

This legacy SPI mode allows just one single bit to be sent/received serially. In this mode, data are sent to the flash memory over the SO signal (whose I/O shared with IO0). Data received from the flash memory arrive via SI (whose I/O shared with IO1).

The different phases can each be configured separately to use this mode by setting the IMODE/ADMODE/ABMODE/DMODE fields (in QUADSPI\_CCR) to 01.

In each phase which is configured in single-SPI mode:

- IO0 (SO) is in output mode.
- IO1 (SI) is in input mode (high impedance).
- IO2 is in output mode and forced to 0.
- 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-SPI 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 this mode, IO0/IO1/IO2/IO3 are all 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 the 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 drives 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 flash memories also send the data using CLK falling edge. By default (when SSHIFT = 0), the signals are sampled using the following (rising) edge of CLK.

### DDR mode

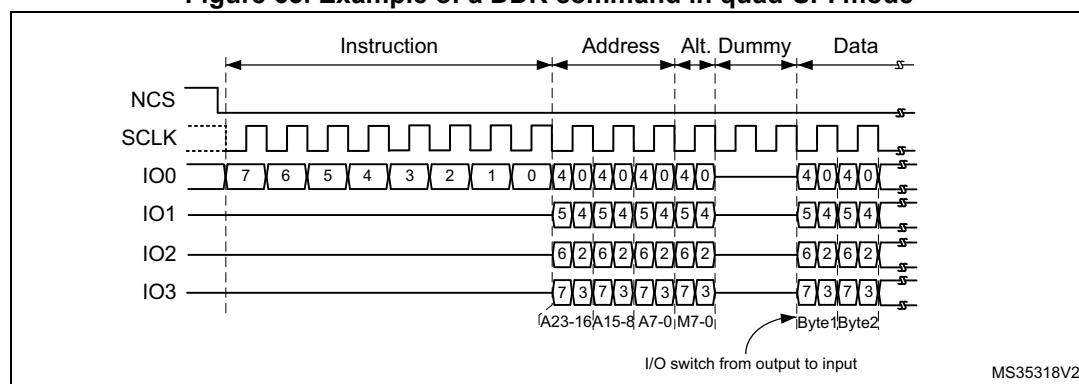
When the DDRM bit (QUADSPI\_CCR[31]) is set to 1, the QUADSPI operates in double-data rate (DDR) mode.

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 CLK falling and rising edges.

The instruction phase is not affected by DDRM. The instruction is always sent using CLK falling edge.

When receiving data in DDR mode, the QUADSPI assumes that flash memories also send the data using both CLK rising and falling edges. When DDRM = 1, the firmware must clear SSHIFT (bit 4 of QUADSPI\_CR). Thus, the signals are sampled one half of a CLK cycle later (on the following, opposite edge).

**Figure 65. Example of a DDR command in quad-SPI 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 uses the same CLK and optionally the same NCS signals, but each have separate IO0, IO1, IO2, and IO3 signals.

The dual-flash mode can be used in conjunction with single-, dual-, and quad-SPI modes, as well as with either SDR or DDR mode.

The flash memory size, as specified in FSIZE[4:0] (QUADSPI\_DCR[20:16]), must 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 must 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 receives 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 bitfield (QUADSPI\_DLR[0]) is stuck at 1 when DFM = 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.

#### 14.3.5 QUADSPI indirect mode

When in indirect mode, commands are started by writing to QUADSPI registers, and data are 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 1s), 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]) must be set to 00.

If QUADSPI\_DLR = 0xFFFF\_FFFF and FSIZE = 0x1F (max value indicating a 4-Gbyte flash memory), then in this special case, 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 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 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)
- after 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)
- after a write is performed to DATA[31:0] (QUADSPI\_DR), 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, BUSY (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), the 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 the 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[4:0] is used to define a FIFO threshold. When the threshold is reached, 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 hardware as soon as the threshold condition is no longer true (after enough data is 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. The reading of the flash memory does not restart until 4 bytes become vacant in the FIFO (when FLEVEL ≤ 28). Thus, when FTHRES ≥ 29, the application must take care to read enough bytes to assure that the QUADSPI starts retrieving data from the flash memory again. Otherwise, the FTF flag stays at 0 as long as 28 < FLEVEL < FTHRES.

#### 14.3.6 QUADSPI automatic status-polling mode

In automatic status-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.

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 status-polling mode. If 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 status-polling mode stop (APMS) bit is set, the 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.

#### 14.3.7 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 the quad-SPI flash bank area before having properly configured and enabled the QUADSPI peripheral.

No more than 256 Mbytes can 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 256-Mbyte 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 an HardFault 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 access and load in advance the byte at the following address. If the

subsequent access is indeed made at a continuous address, the access is 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 may 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.

#### 14.3.8 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] bitfield 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 4 Gbytes (addressed using 32 bits) in indirect mode, but the addressable space in memory-mapped mode is limited to 256 Mbytes.

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 NCS high between the two commands for only one CLK cycle by default. If the flash memory requires more time between commands, the CSHT bitfield 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).

#### 14.3.9 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 SSHIFT (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: SSHIFT must be clear when DDRM bit is set.

#### 14.3.10 QUADSPI configuration

The QUADSPI configuration is done in two phases:

1. QUADSPI peripheral configuration
2. QUADSPI flash memory configuration

Once configured and enabled, the QUADSPI can be used in one of its three operating modes: indirect, automatic status-polling, or memory-mapped mode.

## QUADSPI configuration

The QUADSPI is configured using QUADSPI\_CR. The user must configure the clock prescaler division factor and the sample shifting settings for the incoming data.

The DDR mode can be set through the DDRM bit. When setting the quad-SPI interface in DDR mode, the internal divider of kernel clock must be set with a division ratio of two or more. 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.

DMA requests are enabled setting the DMAEN bit. In case of interrupt usage, their respective enable bit can be also set during this phase.

The FIFO level for either DMA request generation or interrupt generation is programmed in the FTHRES bits.

If a timeout counter is needed, the TCEN bit can be set and the timeout value programmed in the QUADSPI\_LPTR register.

The 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 must program the flash memory size in the FSIZE bits, the chip-select minimum high time in CSHT bits, and the functional mode (Mode 0 or Mode 3) in the MODE bit.

### 14.3.11 QUADSPI use

The operating mode is selected using FMODE[1:0] (QUADSPI\_CCR[27:26]).

#### Indirect mode

When FMODE is programmed to 00, the indirect-write mode is selected and data can be sent to the flash memory. With FMODE = 01, the 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 QUADSPI\_DLR.
2. Specify the frame format, mode and instruction code in QUADSPI\_CCR.
3. Specify optional alternate byte to be sent right after the address phase in QUADSPI\_ABR.
4. Specify the operating mode in QUADSPI\_CR. If FMODE = 00 (indirect-write mode) and DMAEN = 1, then QUADSPI\_AR must be specified before QUADSPI\_CR. Otherwise QUADSPI\_DR can be written by the DMA before QUADSPI\_AR is updated (if the DMA controller has already been enabled).
5. Specify the targeted address in QUADSPI\_AR.
6. Read/write the data from/to the FIFO through QUADSPI\_DR.

When writing QUADSPI\_CR, the user specifies the following settings:

- enable bit (EN) set to 1
- 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 must be set
- interrupt enables
- automatic status-polling mode parameters: match mode and stop mode (valid when FMODE = 11)
- clock prescaler

When writing QUADSPI\_CCR, the user specifies the following parameters:

- instruction byte through 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)
- address size (8/16/24/32-bit) through ADSIZE bits
- the way the alternate bytes have to be sent through the ABMODE (None/1/2/4 lines)
- alternate bytes number (1/2/3/4) through the ABSIZE bits
- presence or not of dummy bytes through the DBMODE bit
- number of dummy bytes through the DCYC bits
- the way data have to be sent/received (none/1/2/4 lines) through DMODE bits

If neither QUADSPI\_AR nor 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.

### Automatic status-polling mode

This mode is enabled setting the FMODE bitfield (QUADSPI\_CCR[27:26]) to 10. In this mode, the programmed frame is 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 is ignored and only 4 bytes are read.

The periodicity is specified in the QUADSPI\_PISR register.

Once the status data is retrieved, it can internally be processed:

- to set the status match flag and generate an interrupt if enabled
- to stop automatically the periodic retrieving of the status bytes

The received value can be masked with the value stored in QUADSPI\_PSMKR and ORed or ANDed with the value stored in 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 QUADSPI\_DR.

### Memory-mapped mode

In memory-mapped mode, the external flash memory is seen as an internal memory but with some latency during accesses. Only read operations are allowed to the external flash memory in this mode.

The memory-mapped mode is entered by setting FMODE to 11 in QUADSPI\_CCR.

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.

QUADSPI\_DLR has no meaning in memory-mapped mode.

### 14.3.12 Sending the instruction only once

Some flash memories (for example: Winbound) 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 status-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).

### 14.3.13 QUADSPI error management

An error can be generated in the following case:

- In indirect mode or automatic status-polling mode when a wrong address is programmed in QUADSPI\_AR (according to the flash memory size defined by FSIZE[4:0] in the QUADSPI\_DCR), TEF is set and an interrupt is generated if enabled. Also in indirect mode, if the address plus the data length exceeds the flash memory size, TEF is 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, a bus error is generated 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, a bus error is generated as a response to the faulty bus master request.

### 14.3.14 QUADSPI busy bit and abort functionality

Once the QUADSPI starts an operation with the flash memory, the BUSY bit is automatically set in QUADSPI\_SR.

In indirect mode, BUSY is reset once the QUADSPI has completed the requested command sequence, and the FIFO is empty.

In automatic status-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 QUADSPI\_CR. Once the abort is completed, BUSY and ABORT are automatically reset, and the FIFO is flushed.

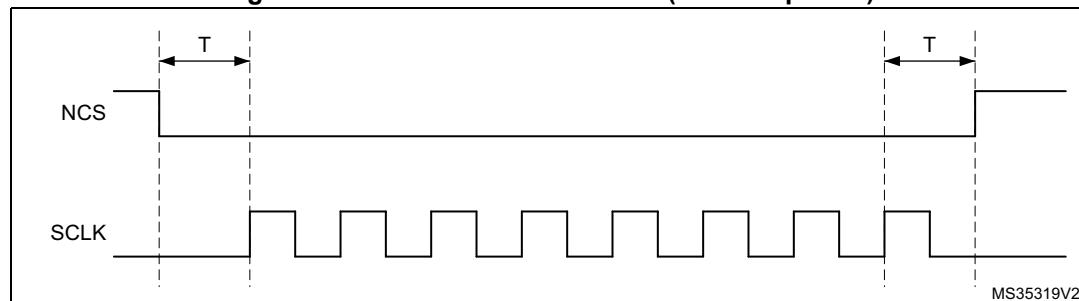
*Note:* Some flash memories may misbehave if a write operation to a status registers is aborted.

#### 14.3.15 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.

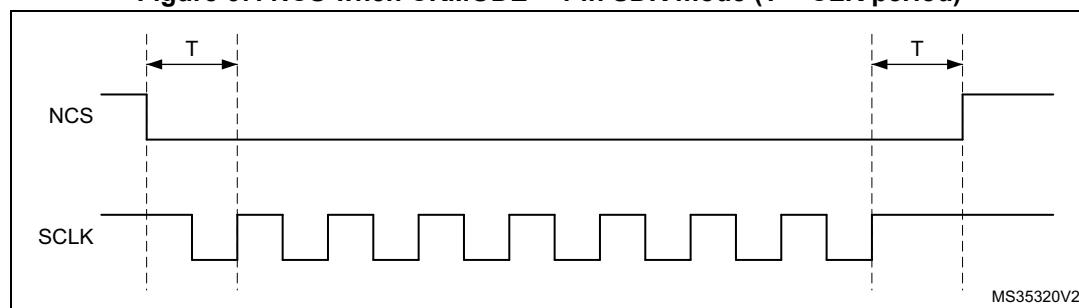
When CKMODE = 0 (“mode0”, where CLK stays low when no operation is in progress), NCS falls one CLK cycle before an operation first rising CLK edge, and NCS rises one CLK cycle after the operation final rising CLK edge, as shown in the figure below.

Figure 66. NCS when CKMODE = 0 ( $T = \text{CLK period}$ )



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 the figure below.

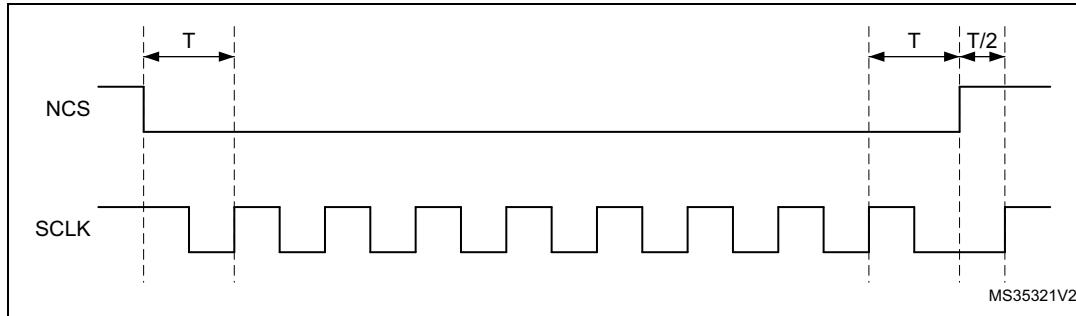
Figure 67. NCS when CKMODE = 1 in SDR mode ( $T = \text{CLK period}$ )



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 the figure below. Because DDR operations must

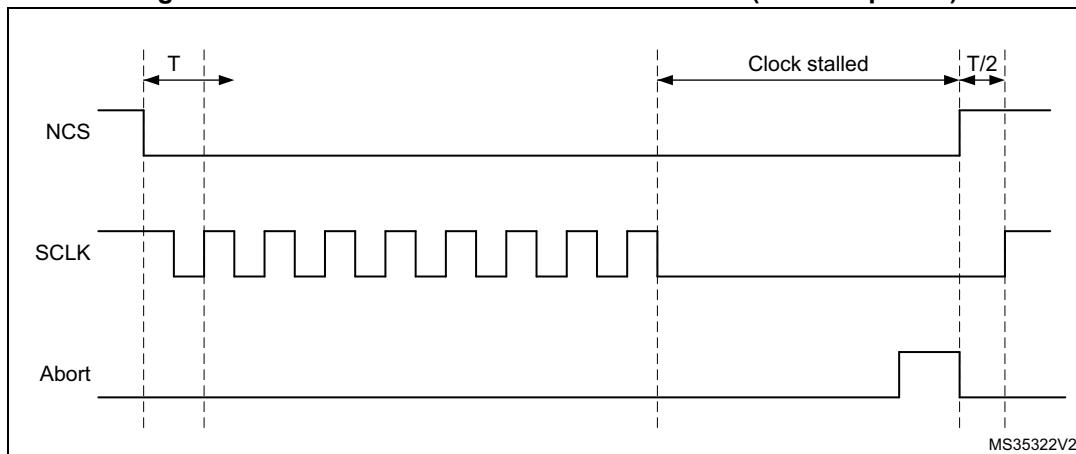
finish with a falling edge, CLK is low when NCS rises, and CLK rises back up one half of a CLK cycle afterwards.

**Figure 68. NCS when CKMODE = 1 in DDR mode (T = CLK period)**



When the FIFO stays full in a read operation or if the FIFO stays empty in a write operation, the operation stalls and CLK stays low until firmware services the FIFO. If an abort occurs when an operation is stalled, NCS rises just after the abort is requested and then CLK rises one half of a CLK cycle later, as shown in [Figure 69](#).

**Figure 69. NCS when CKMODE = 1 with an abort (T = CLK period)**



When not in dual-flash mode (DFM = 0) and FSEL = 0 (default value), only FLASH 1 is accessed. Thus BK2\_NCS stays high, if FSEL = 1, only FLASH 2 is accessed and BK1\_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 is dual-flash mode only, then BK1\_NCS signal can be used for FLASH 2 as well, and the pin devoted to BK2\_NCS can be used for other functions.

## 14.4 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 91. 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

## 14.5 QUADSPI registers

### 14.5.1 QUADSPI control register (QUADSPI\_CR)

Address offset: 0x000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRESCALER[7:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	Res.	TOIE	SMIE	FTIE	TCIE	TEIE
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	FTHRES[4:0]				FSEL	DFM	Res.	SSHIFT	TCEN	DMAEN	ABORT	EN	
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **PRESCALER[7:0]**: Clock prescaler

This bitfield 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 duty cycle is not 50%. The clock signal remains low one cycle longer than it stays high.

When setting quad-SPI interface in DDR mode, the prescaler must be set with a division ratio of two or more.

*Note: This bitfield can be modified only when BUSY = 0.*

**Bit 23 PMM:** Polling match mode

This bit indicates which method must be used for determining a “match” during automatic status-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.

*Note: This bit can be modified only when BUSY = 0.*

**Bit 22 APMS:** Automatic status-polling mode stop

This bit determines if automatic status-polling is stopped after a match.

0: Automatic status-polling mode is stopped only by abort or by disabling the QUADSPI.

1: Automatic status-polling mode stops as soon as there is a match.

*Note: 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 disabled

1: Interrupt enabled

**Bit 19 SMIE:** Status match interrupt enable

This bit enables the status match interrupt.

0: Interrupt disabled

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 disabled

1: Interrupt enabled

Bits 15:13 Reserved, must be kept at reset value.

**Bits 12:8 FTHRES[4:0]: FIFO threshold level**

This bitfield defines, in indirect mode, the threshold number of bytes in the FIFO that causes the FIFO threshold flag (bit FTF in register QUADSPI\_SR) to be set.

0: In indirect-write mode (FMODE = 00), FTF is set if there are one or more free bytes location left in the FIFO or indirect-read mode (FMODE = 01), FTF is set if there are one or more valid bytes that can be read from the FIFO.

1: In indirect-write mode (FMODE = 00), FTF is set if there are two or more free bytes location left in the FIFO, or indirect-read mode (FMODE = 01), FTF is set if there are two or more valid bytes that can be read from the FIFO

...

31: In indirect-write mode (FMODE = 00), FTF is set if there are 32 free bytes location left in the FIFO, or indirect-read mode (FMODE = 01), 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

*Note: 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

*Note: 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

The firmware must assure that SSHIFT = 0 when in DDR mode (when DDRM = 1).

*Note: This bitfield 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 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). This bit enables 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 may 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 NCS remains active indefinitely after an access in memory-mapped mode.

1: Timeout counter is enabled, and thus the NCS is released in memory-mapped mode after TIMEOUT[15:0] cycles of flash memory inactivity.

*Note: This bit can be modified only when BUSY = 0.*

Bit 2 **DMAEN**: DMA enable

In indirect mode, the 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 ongoing command sequence. It is automatically reset once the abort is complete. This bit stops the current transfer.

In automatic status-polling or memory-mapped mode, this bit also reset APM or DM bit.

- 0: No abort requested
- 1: Abort requested

Bit 0 **EN**: QUADSPI enable

- 0: QUADSPI disabled
- 1: QUADSPI enabled

**14.5.2 QUADSPI device configuration register (QUADSPI\_DCR)**

Address offset: 0x004

Reset value: 0x0000 0000

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

Bits 31:21 Reserved, must be kept at reset value.

Bits 20:16 **FSIZE[4:0]**: Flash memory size

This bitfield defines the size of external memory using the following formula:

$$\text{Number of bytes in flash memory} = 2^{\lceil \text{FSIZE}+1 \rceil}$$

FSIZE+1 is effectively the number of address bits required to address the flash memory.

The flash memory capacity can be up to 4 Gbytes (addressed using 32 bits) in indirect mode, but the addressable space in memory-mapped mode is limited to 256 Mbytes.

If DFM = 1, FSIZE indicates the total capacity of the two flash memories together.

*Note: This bitfield 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

*Note: This bitfield 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.

*Note: This bitfield can be modified only when BUSY = 0.*

### 14.5.3 QUADSPI status register (QUADSPI\_SR)

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	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 bitfield 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 status-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 is 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 status-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 is transferred or in any mode when the transfer is 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.

#### 14.5.4 QUADSPI flag clear register (QUADSPI\_FCR)

Address offset: 0x00C

Reset value: 0x0000 0000

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

Bits 31:5 Reserved, must be kept at reset value.

Bit 4 **CTOF**: Clear timeout flag

Writing 1 clears the TOF flag in QUADSPI\_SR.

Bit 3 **CSMF**: Clear status match flag

Writing 1 clears the SMF flag in QUADSPI\_SR.

Bit 2 Reserved, must be kept at reset value.

Bit 1 **CTCF**: Clear transfer complete flag

Writing 1 clears the TCF flag in QUADSPI\_SR.

Bit 0 **CTEF**: Clear transfer error flag

Writing 1 clears the TEF flag in QUADSPI\_SR.

#### 14.5.5 QUADSPI data length register (QUADSPI\_DLR)

Address offset: 0x010

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 automatic status-polling modes. A value no greater than 3 (indicating 4 bytes) must be used for automatic status-polling mode.

All 1s in indirect mode means undefined length, where the QUADSPI continues 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 bitfield has no effect when in memory-mapped mode (FMODE = 10).

*Note: This bitfield can be written only when BUSY = 0.*

**14.5.6 QUADSPI communication configuration register (QUADSPI\_CCR)**

Address offset: 0x014

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DDRM	DHHC	Res.	SIOO	FMODE[1:0]		DMODE[1:0]		Res.	DCYC[4:0]				ABSIZE[1:0]		
rw	rw		rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABMODE[1:0]		ADSIZE[1:0]		ADMODE[1:0]		IMODE[1:0]		INSTRUCTION[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **DDRM**: Double data rate mode

This bit sets the DDR mode for the address, alternate byte and data phase:

0: DDR mode disabled

1: DDR mode enabled

*Note: This bit can be written only when BUSY = 0.*

Bit 30 **DHHC**: DDR hold

This bit delays the data output by 1/4 of the QUADSPI output clock cycle in DDR mode:

0: Data output delayed using analog delay

1: Data output delayed by 1/4 of a QUADSPI output clock cycle

This feature is only active in DDR mode.

*Note: This bit can be written only when BUSY = 0.*

*PRESCALER > 0 is mandatory when DHHC = 1.*

Bit 29 Reserved, must be kept at reset value.

Bit 28 **SIOO**: Send instruction only once mode

This bit has no effect when IMODE = 00. See [Section 14.3.12](#) for more details.

0: Instruction sent on every transaction

1: Instruction sent only for the first command

*Note: This bit can be written only when BUSY = 0.*

Bits 27:26 **FMODE[1:0]**: Functional mode

This bitfield defines the QUADSPI functional mode of operation.

00: Indirect-write mode

01: Indirect-read mode

10: Automatic status-polling mode

11: Memory-mapped mode

If DMAEN = 1 already, the DMA controller for the corresponding channel must be disabled before changing the FMODE value.

*Note: This bitfield can be written only when BUSY = 0.*

Bits 25:24 **DMODE[1:0]**: Data mode

This bitfield defines the data phase mode of operation:

00: No data

01: Data on a single line

10: Data on two lines

11: Data on four lines

This bitfield also determines the dummy phase mode of operation.

*Note: This bitfield 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 bitfield defines the duration of the dummy phase. In both SDR and DDR modes, it specifies a number of CLK cycles (0-31).

*Note: This bitfield can be written only when BUSY = 0.*

Bits 17:16 **ABSIZE[1:0]**: Alternate-byte size

This bit defines the size of alternate bytes.

00: 8-bit alternate byte

01: 16-bit alternate bytes

10: 24-bit alternate bytes

11: 32-bit alternate bytes

*Note: This bitfield can be written only when BUSY = 0.*

Bits 15:14 **ABMODE[1:0]**: Alternate byte mode

This bitfield defines the alternate-byte 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

*Note: This bitfield 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

*Note: This bitfield can be written only when BUSY = 0.*

**Bits 11:10 ADMODE[1:0]: Address mode**

This bitfield 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

*Note: This bitfield can be written only when BUSY = 0.*

**Bits 9:8 IMODE[1:0]: Instruction mode**

This bitfield 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

*Note: This bitfield can be written only when BUSY = 0.*

**Bits 7:0 INSTRUCTION[7:0]: Instruction**

Instruction to be sent to the external SPI device.

*Note: This bitfield can be written only when BUSY = 0.*

**14.5.7 QUADSPI address register (QUADSPI\_AR)**

Address offset: 0x018

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**

This bitfield contains the address to be sent to the external flash memory.

Writes to this bitfield are ignored when BUSY = 1 or when FMODE = 11 (memory-mapped mode).

In dual flash mode, ADDRESS[0] is automatically stuck to 0 as the address must always be even

**14.5.8 QUADSPI alternate-byte register (QUADSPI\_ABR)**

Address offset: 0x01C

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.

*Note:* This bitfield can be written only when BUSY = 0.

### 14.5.9 QUADSPI data register (QUADSPI\_DR)

Address offset: 0x020

Reset value: 0x0000 0000

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

Bits 31:0 **DATA[31:0]**: Data

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 status-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].

### 14.5.10 QUADSPI polling status mask register (QUADSPI\_PSMKR)

Address offset: 0x024

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 automatic status-polling mode.

For bit n:

0: Bit n of the data received in automatic status-polling mode is masked and its value is not considered in the matching logic

1: Bit n of the data received in automatic status-polling mode is unmasked and its value is considered in the matching logic

*Note: This bitfield can be written only when BUSY = 0.*

#### 14.5.11 QUADSPI polling status match register (QUADSPI\_PSMAR)

Address offset: 0x028

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.

*Note: This bitfield can be written only when BUSY = 0.*

#### 14.5.12 QUADSPI polling interval register (QUADSPI\_PIR)

Address offset: 0x02C

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 two read during automatic status-polling phases.

*Note: This bitfield can be written only when BUSY = 0.*

### 14.5.13 QUADSPI low-power timeout register (QUADSPI\_LPTR)

Address offset: 0x030

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.
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 bitfield 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.

*Note: This bitfield can be written only when BUSY = 0.*

### 14.5.14 QUADSPI register map

Table 92. QUADSPI register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x000	QUADSPI_CR	PRESCALER[7:0]							PMM	APMS	22	21	20	19	18	17	16
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x004	QUADSPI_DCR	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
0x008	QUADSPI_SR	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
0x00C	QUADSPI_FCR	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
0x010	QUADSPI_DLR	DL[31:0]															
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x014	QUADSPI_CCR	DDRM	DHFC	Res.	STOO	FMODE[1:0]	DMODE[1:0]	Res.	DCYC[4:0]	ABSIZE[1:0]	AMODE[1:0]	ADSIZ[1:0]	ADMODE[1:0]	IMODE[1:0]	INSTRUCTION[7:0]		Res.
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x018	QUADSPI_AR	ADDRESS[31:0]															
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x01C	QUADSPI_ABR	ALTERNATE[31:0]															
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x020	QUADSPI_DR	DATA[31:0]															
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Table 92. QUADSPI register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x024	QUADSPI_PSMKR																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x028	QUADSPI_PSMAR																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x02C	QUADSPI_PIR	Res.																															
	Reset value																																
0x030	QUADSPI_LPTR	Res.																															
	Reset value																																

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

## 15 Analog-to-digital converter (ADC)

### 15.1 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.

### 15.2 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 input range:  $V_{REF-} \leq V_{IN} \leq V_{REF+}$
- DMA request generation during regular channel conversion

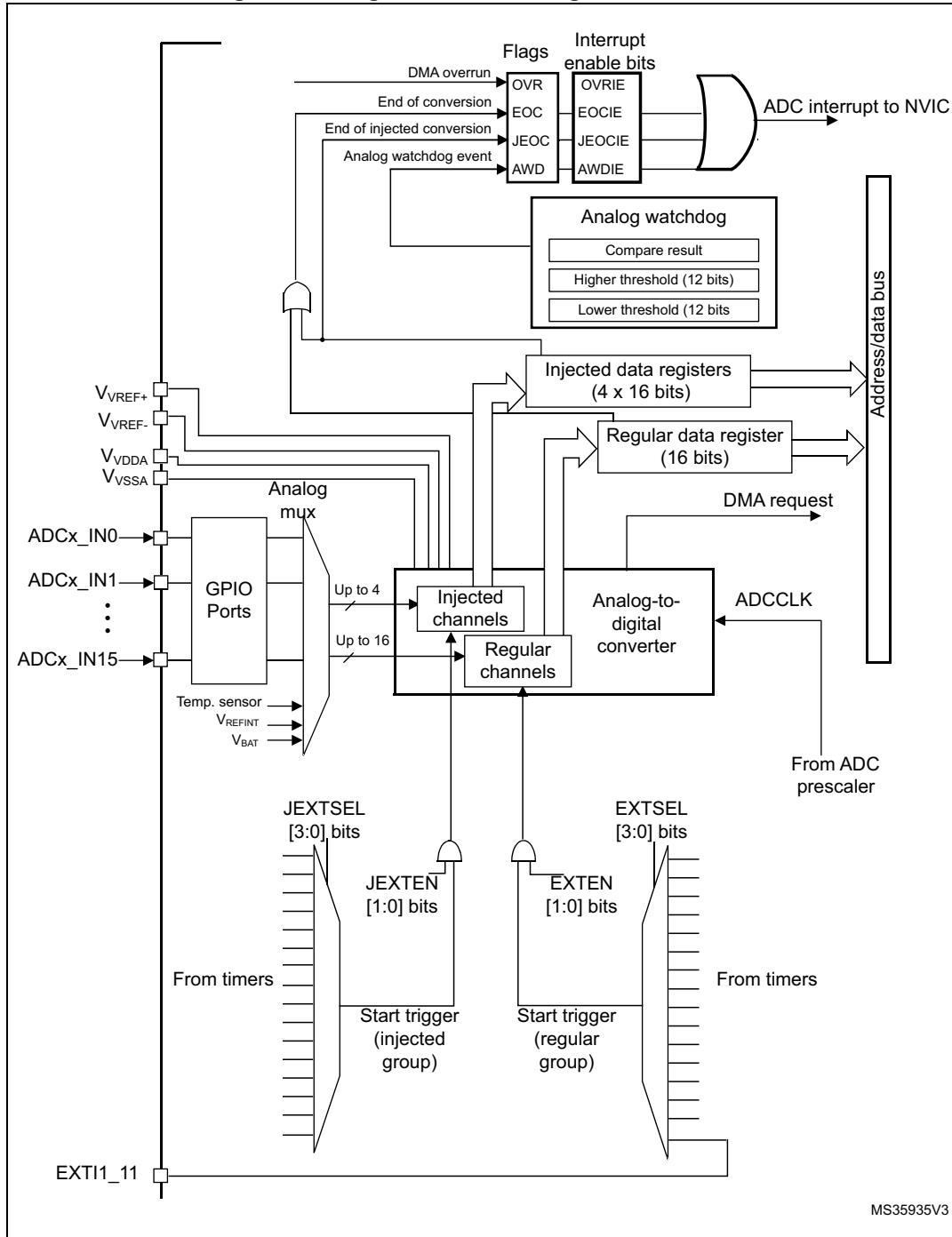
*Figure 70* shows the block diagram of the ADC.

*Note:*  $V_{REF-}$ , if available (depending on package), must be tied to  $V_{SSA}$ .

## 15.3 ADC functional description

Figure 70 shows a single ADC block diagram and Table 93 gives the ADC pin description.

**Figure 70. Single ADC block diagram**



**Table 93.** ADC pins

Name	Signal type	Remarks
$V_{REF+}$	Input, analog reference positive	The higher/positive reference voltage for the ADC
$V_{DDA}$	Input, analog supply	Analog power supply equal to $V_{DD}$
$V_{REF-}$	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

### 15.3.1 ADC on-off control

The ADC is powered on by setting the ADON bit in the ADC\_CR2 register. When the ADON bit is set for the first time, it wakes up the ADC from the Power-down mode.

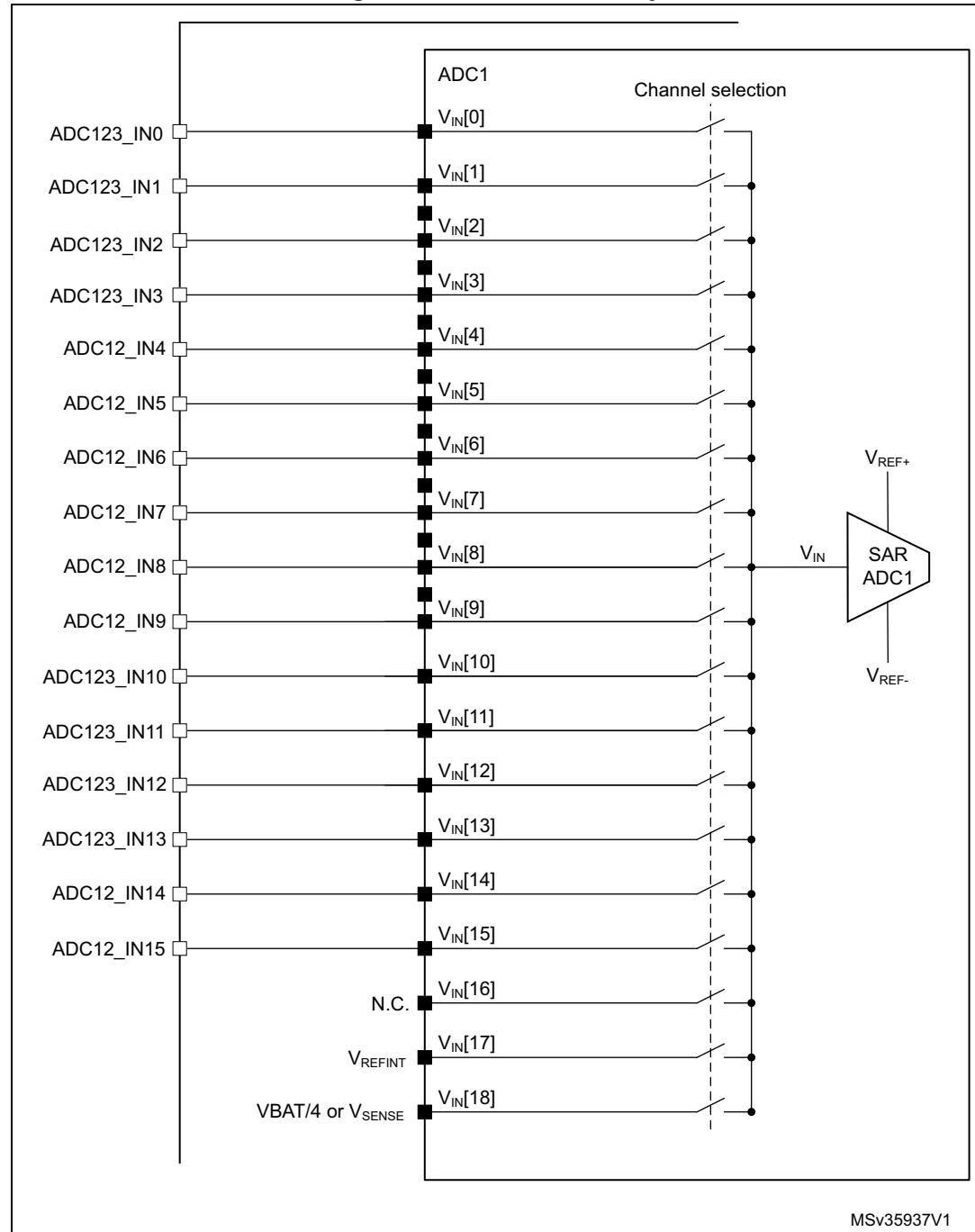
The conversion starts when either the SWSTART or the JSWSTART bit is set.

The user can stop conversion and put the ADC in power down mode by clearing the ADON bit. In this mode the ADC consumes almost no power (only a few  $\mu A$ ).

### 15.3.2 ADC1/2 and ADC3 connectivity

ADC1, ADC2 and ADC3 are tightly coupled and share some external channels as described in [Figure 71](#), [Figure 72](#) and [Figure 73](#).

**Figure 71. ADC1 connectivity**



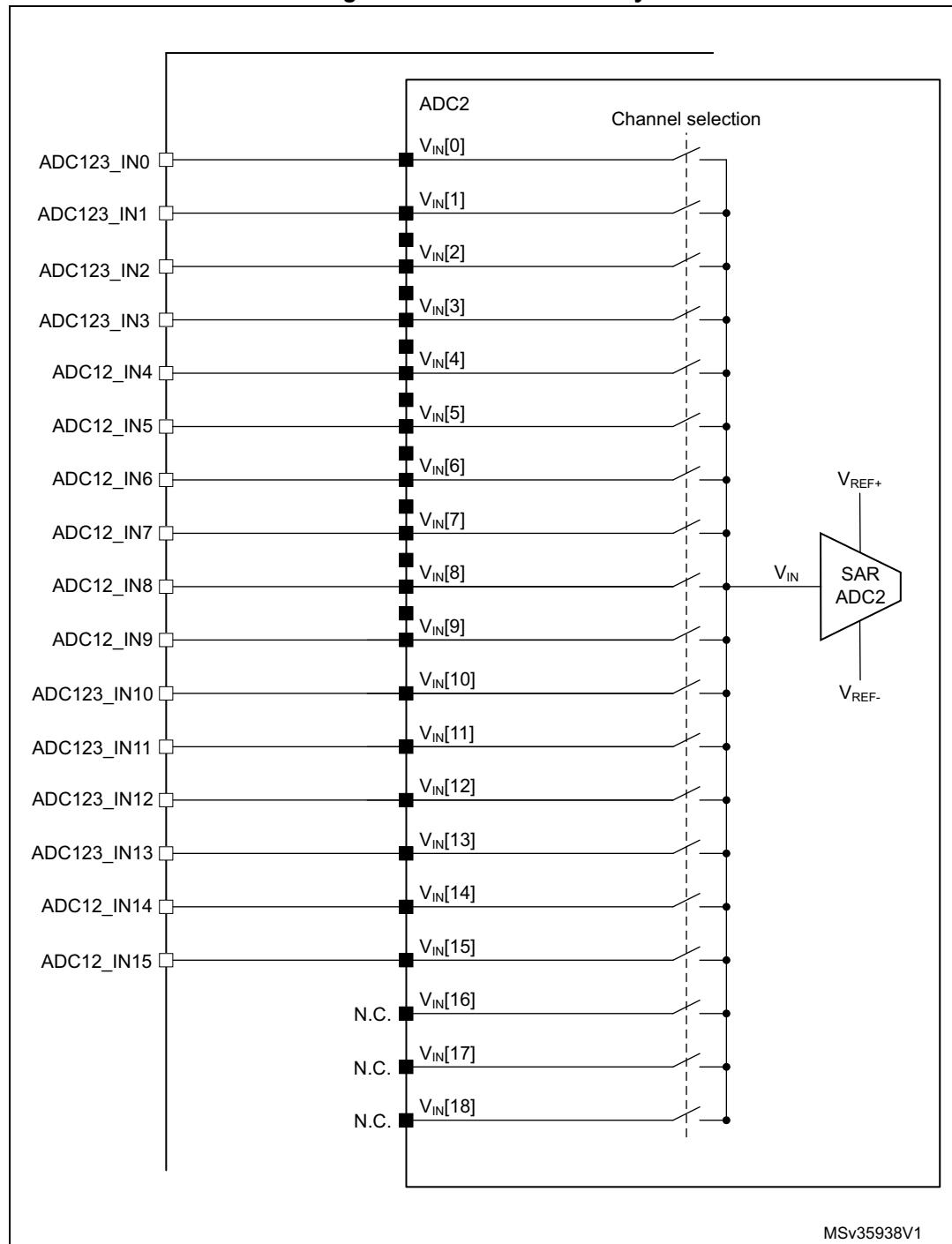
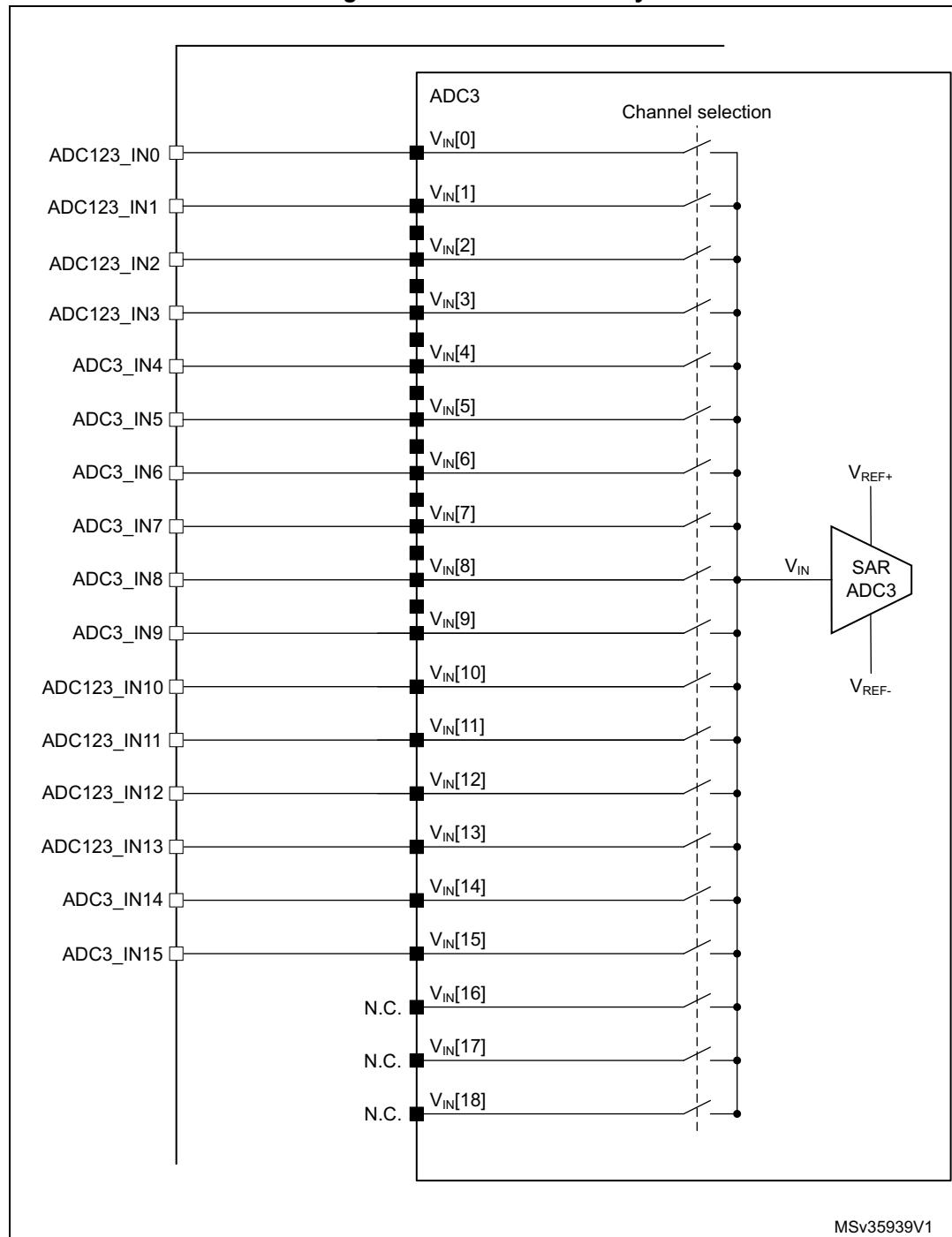
**Figure 72. ADC2 connectivity**

Figure 73. ADC3 connectivity



### 15.3.3 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).

### 15.3.4 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.

### 15.3.5 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.

### 15.3.6 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 SWSTART bit in the ADC\_CR2 register (for regular channels only).

After each conversion:

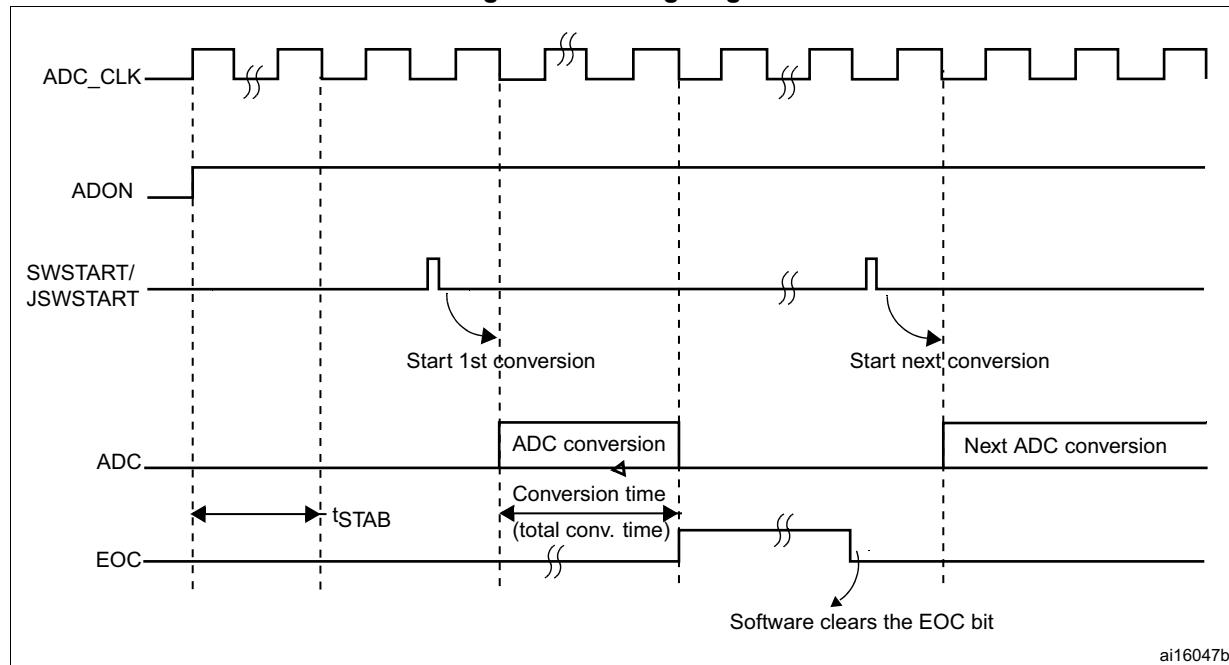
- If a regular group of channels was converted:
  - The last converted data are stored into the 16-bit ADC\_DR register
  - The EOC (end of conversion) flag is set
  - An interrupt is generated if the EOCIE bit is set

**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.*

### 15.3.7 Timing diagram

As shown in [Figure 74](#), the ADC needs a stabilization time of  $t_{STAB}$  before it starts converting accurately. After the start of the ADC conversion and after 15 clock cycles, the EOC flag is set and the 16-bit ADC data register contains the result of the conversion.

Figure 74. Timing diagram



ai16047b

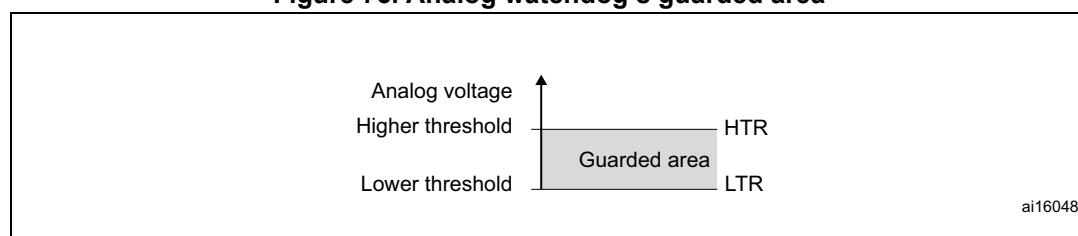
### 15.3.8 Analog watchdog

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 94](#) shows how the ADC\_CR1 register should be configured to enable the analog watchdog on one or more channels.

Figure 75. Analog watchdog's guarded area



ai16048

Table 94. 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 94. 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

### 15.3.9 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.

### 15.3.10 Injected channel management

#### Triggered injection

To use triggered injection, the JAUTO bit must be cleared in the ADC\_CR1 register.

1. Start the conversion of a group of regular channels either by external trigger or by setting the SWSTART bit in the ADC\_CR2 register.
2. If an external injected trigger occurs or if the JSWSTART bit is set during the conversion of a regular group of channels, the current conversion is reset and the injected channel sequence switches to Scan-once mode.
3. Then, the regular conversion of the regular group of channels is resumed from the last interrupted regular conversion.

If a regular event occurs during an injected conversion, the injected conversion is not interrupted but the regular sequence is executed at the end of the injected sequence.

*Figure 76* shows the corresponding timing diagram.

**Note:** When using triggered injection, one must ensure that the interval between trigger events is longer than the injection sequence. For instance, if the sequence length is 30 ADC clock cycles (that is two conversions with a sampling time of 3 clock periods), the minimum interval between triggers must be 31 ADC clock cycles.

### Auto-injection

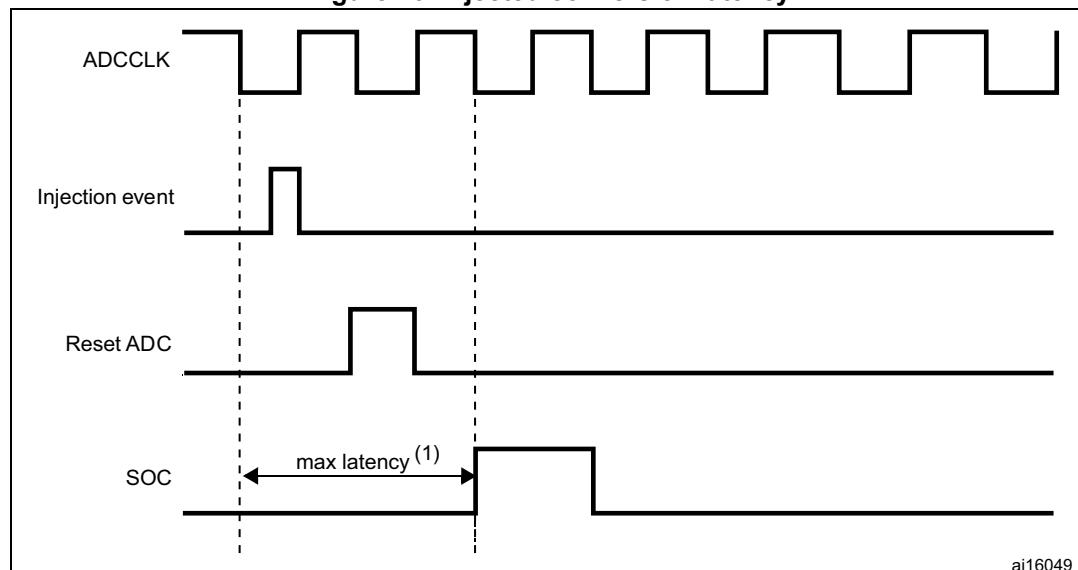
If the JAUTO bit is set, then the channels in the injected group are automatically converted after the regular group of channels. This can be used to convert a sequence of up to 20 conversions programmed in the ADC\_SQRx and ADC\_JSQR registers.

In this mode, external trigger on injected channels must be disabled.

If the CONT bit is also set in addition to the JAUTO bit, regular channels followed by injected channels are continuously converted.

**Note:** It is not possible to use both the auto-injected and discontinuous modes simultaneously.

**Figure 76. Injected conversion latency**



ai16049

1. The maximum latency value can be found in the electrical characteristics of the STM32F75xxx and STM32F74xxx datasheets.

### 15.3.11 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.*

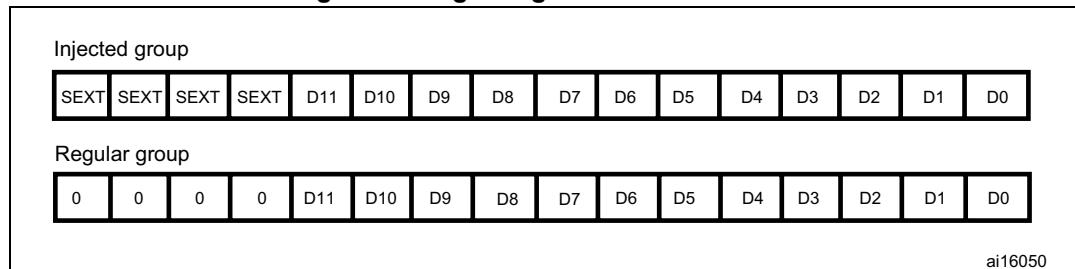
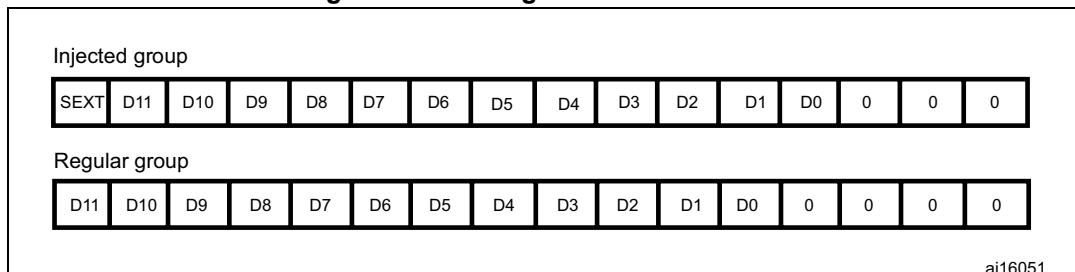
*Discontinuous mode must be enabled only for the conversion of one group.*

## 15.4 Data alignment

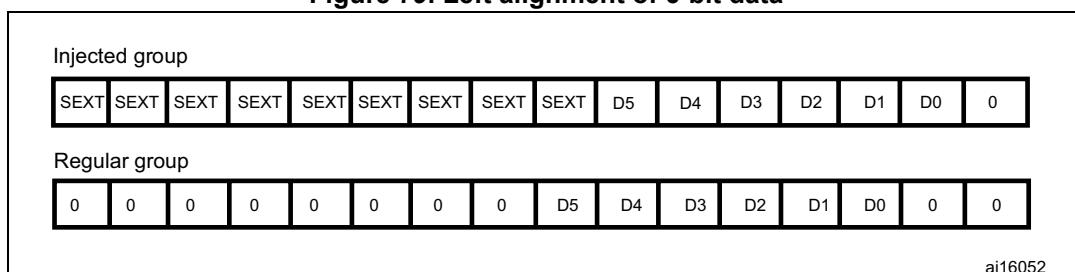
The ALIGN bit in the ADC\_CR2 register selects the alignment of the data stored after conversion. Data can be right- or left-aligned as shown in [Figure 77](#) and [Figure 78](#).

The converted data value from the injected group of channels is decreased by the user-defined offset written in the ADC\_JOFRx registers so the result can be a negative value. The SEXT bit represents the extended sign value.

For channels in a regular group, no offset is subtracted so only twelve bits are significant.

**Figure 77. Right alignment of 12-bit data****Figure 78. Left alignment of 12-bit data**

Special case: when left-aligned, the data are aligned on a half-word basis except when the resolution is set to 6-bit. in that case, the data are aligned on a byte basis as shown in [Figure 79](#).

**Figure 79. Left alignment of 6-bit data**

## 15.5 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:

$$T_{\text{conv}} = 3 + 12 = 15 \text{ cycles} = 0.5 \mu\text{s with APB2 at 60 MHz}$$

## 15.6 Conversion on external trigger and trigger polarity

Conversion can be triggered by an external event (such as timer trigger/compare, 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 95](#) provides the correspondence between the EXTEN[1:0] and JEXTEN[1:0] values and the trigger polarity.

**Table 95. 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 96](#) gives the possible external trigger for regular conversion.

**Table 96. External trigger for regular channels**

Source	Type	EXTSEL[3:0]
tim1_oc1	Internal signal from on-chip timers	0000
tim1_oc2		0001
tim1_oc3		0010
tim2_oc2		0011
tim5_trgo		0100
tim4_oc4		0101
tim3_oc4		0110
tim8_trgo		0111
tim8_trgo2		1000
tim1_trgo		1001
tim1_trgo2		1010
tim2_trgo		1011
tim4_trgo		1100
tim6_trgo		1101
exti11	External pin	1111

*Table 97* gives the possible external trigger for injected conversion.

**Table 97. External trigger for injected channels**

Source	Connection type	JEXTSEL[3:0]
tim1_trgo	Internal signal from on-chip timers	0000
tim1_oc4		0001
tim2_trgo		0010
tim2_oc1		0011
tim3_oc4		0100
tim4_trgo		0101
tim8_oc4	Internal signal from on-chip timers	0111
tim1_trgo2		1000
tim8_trgo		1001
tim8_trgo2		1010
tim3_oc3		1011
tim5_trgo		1100
tim3_oc1		1101
tim6_trgo		1110

Software source trigger events can be generated by setting SWSTART (for regular conversion) or JSWSTART (for injected conversion) in ADC\_CR2.

A regular group conversion can be interrupted by an injected trigger.

**Note:**

*The trigger selection can be changed on the fly. However, when the selection changes, there is a time frame of 1 APB clock cycle during which the trigger detection is disabled. This is to avoid spurious detection during transitions.*

## 15.7 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

## 15.8 Data management

### 15.8.1 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\_SxNDTR 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.

### 15.8.2 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.

### 15.8.3 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.

## 15.9 Multi ADC mode

In devices with two ADCs or more, the Dual (with two ADCs) and Triple (with three ADCs) ADC modes can be used (see [Figure 80](#)).

In multi ADC mode, the start of conversion is triggered alternately or simultaneously by the ADC1 master to the ADC2 and ADC3 slaves, depending on the mode selected by the MULTI[4:0] bits in the ADC\_CCR register.

**Note:** *In multi ADC mode, when configuring conversion trigger by an external event, the application must set trigger by the master only and disable trigger by slaves to prevent spurious triggers that would start unwanted slave conversions.*

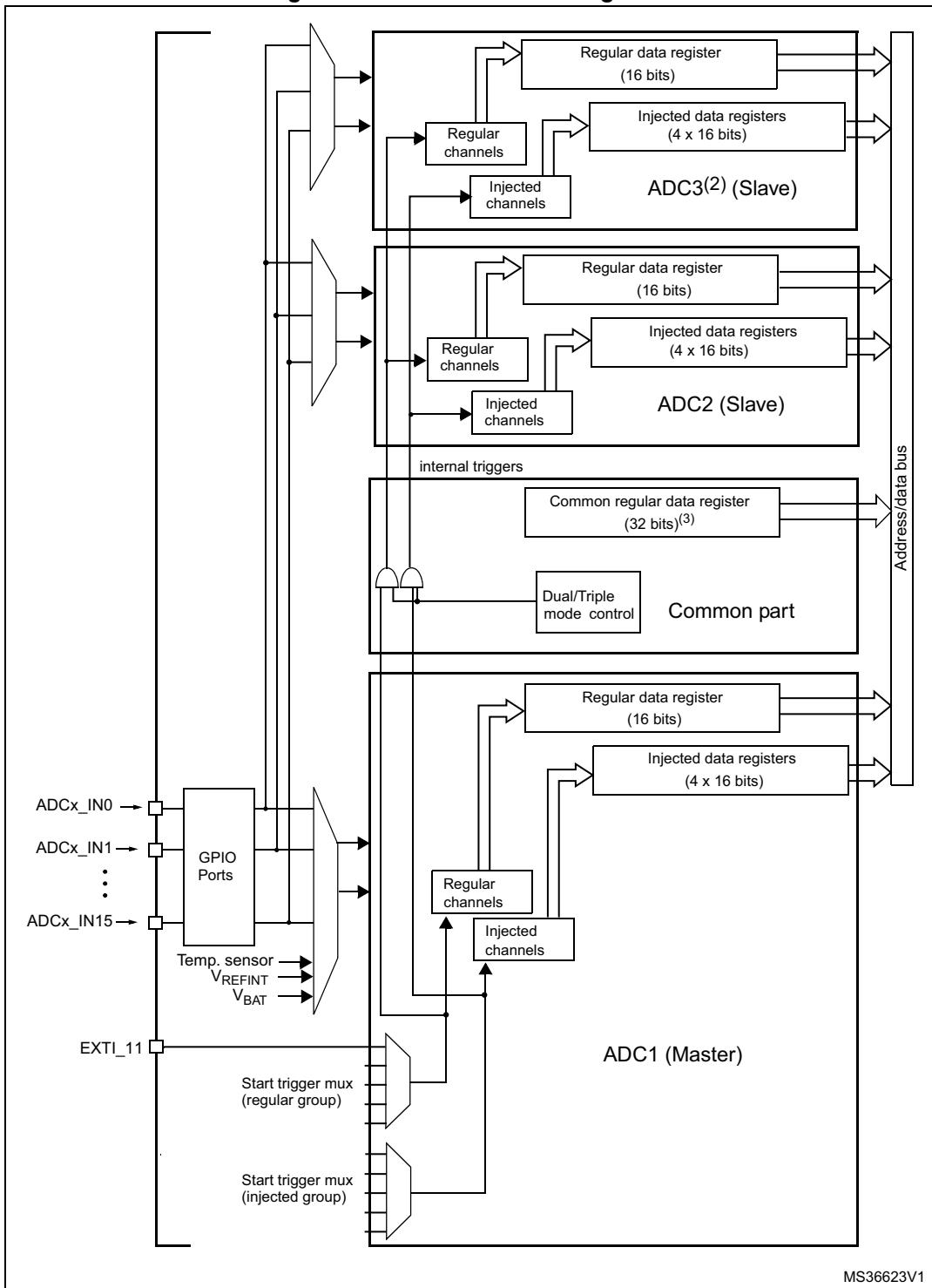
The four possible modes below are implemented:

- Injected simultaneous mode
- Regular simultaneous mode
- Interleaved mode
- Alternate trigger mode

It is also possible to use the previous modes combined in the following ways:

- Injected simultaneous mode + Regular simultaneous mode
- Regular simultaneous mode + Alternate trigger mode

**Note:** *In multi ADC mode, the converted data can be read on the multi-mode data register (ADC\_CDR). The status bits can be read in the multi-mode status register (ADC\_CSR).*

Figure 80. Multi ADC block diagram<sup>(1)</sup>

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, DMA mode 1 is not supported.

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 can be 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:**

- 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:  $\text{ADC\_CDR}[31:0] = \text{ADC2\_DR}[15:0] | \text{ADC1\_DR}[15:0]$

- 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. Interleaved dual and triple modes are supported:

**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:  $\text{ADC\_CDR}[15:0] = \text{ADC1\_DR}[7:0] | \text{ADC3\_DR}[7:0]$

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.

### 15.9.1 Injected simultaneous mode

This mode converts an injected group of channels. The external trigger source comes from the injected group multiplexer of ADC1 (selected by the JEXTSEL[3:0] bits in the ADC1\_CR2 register). A simultaneous trigger is provided to ADC2 and ADC3.

**Note:** *Do not convert the same channel on the two/three ADCs (no overlapping sampling times for the two/three ADCs when converting the same channel).*

*In simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longer of the 2 sequences (Dual ADC mode) /3 sequences (Triple ADC mode). Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.*

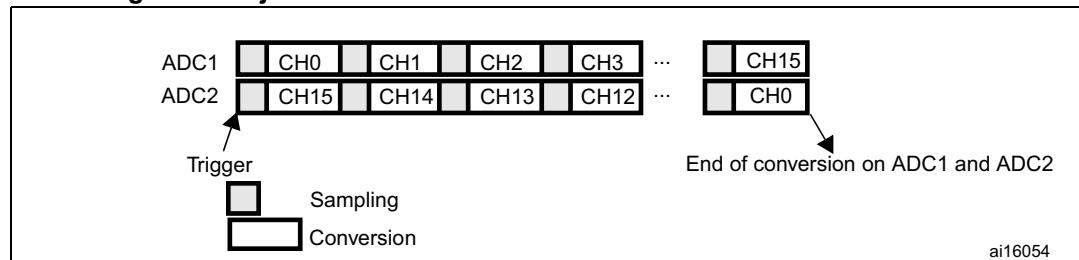
*Regular conversions can be performed on one or all ADCs. In that case, they are independent of each other and are interrupted when an injected event occurs. They are resumed at the end of the injected conversion group.*

### Dual ADC mode

At the end of conversion event on ADC1 or ADC2:

- The converted data are stored into the ADC\_JDRx registers of each ADC interface.
- A JEOC interrupt is generated (if enabled on one of the two ADC interfaces) when the ADC1/ADC2's injected channels have all been converted.

**Figure 81. Injected simultaneous mode on 4 channels: dual ADC mode**

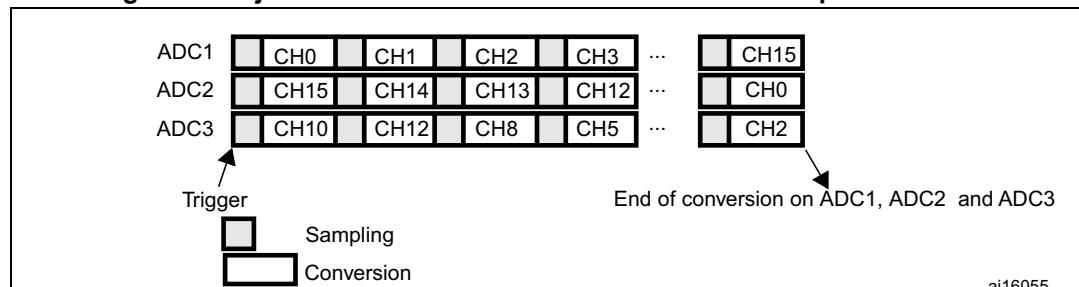


### Triple ADC mode

At the end of conversion event on ADC1, ADC2 or ADC3:

- The converted data are stored into the ADC\_JDRx registers of each ADC interface.
- A JEOC interrupt is generated (if enabled on one of the three ADC interfaces) when the ADC1/ADC2/ADC3's injected channels have all been converted.

**Figure 82. Injected simultaneous mode on 4 channels: triple ADC mode**



### 15.9.2 Regular simultaneous mode

This mode is performed on a regular group of channels. The external trigger source comes from the regular group multiplexer of ADC1 (selected by the EXTSEL[3:0] bits in the ADC1\_CR2 register). A simultaneous trigger is provided to ADC2 and ADC3.

*Note:* *Do not convert the same channel on the two/three ADCs (no overlapping sampling times for the two/three ADCs when converting the same channel).*

*In regular simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the long conversion time of the 2 sequences (Dual ADC mode) /3 sequences (Triple ADC mode). Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.*

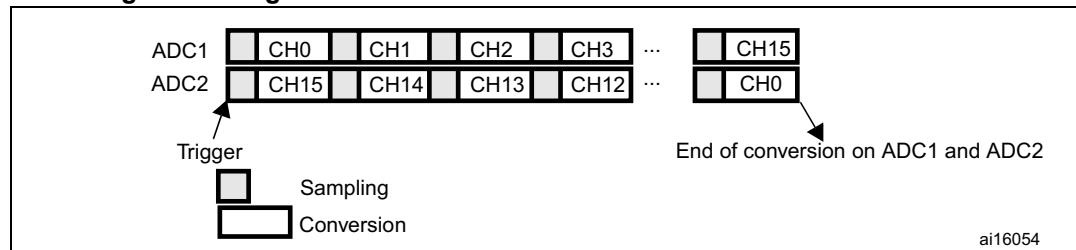
*Injected conversions must be disabled.*

### Dual ADC mode

At the end of conversion event on ADC1 or ADC2:

- A 32-bit DMA transfer request is generated (if DMA[1:0] bits in the ADC\_CCR register are equal to 0b10). This request transfers the ADC2 converted data stored in the upper half-word of the ADC\_CDR 32-bit register to the SRAM and then the ADC1 converted data stored in the lower half-word of ADC\_CCR to the SRAM.
- An EOC interrupt is generated (if enabled on one of the two ADC interfaces) when the ADC1/ADC2's regular channels have all been converted.

**Figure 83. Regular simultaneous mode on 16 channels: dual ADC mode**



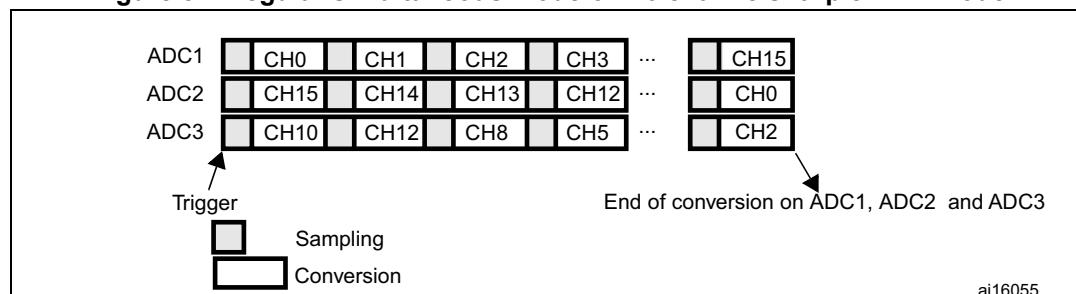
ai16054

### Triple ADC mode

At the end of conversion event on ADC1, ADC2 or ADC3:

- Three 32-bit DMA transfer requests are generated (if DMA[1:0] bits in the ADC\_CCR register are equal to 0b01). Three transfers then take place from the ADC\_CDR 32-bit register to SRAM: first the ADC1 converted data, then the ADC2 converted data and finally the ADC3 converted data. The process is repeated for each new three conversions.
- An EOC interrupt is generated (if enabled on one of the three ADC interfaces) when the ADC1/ADC2/ADC3's regular channels have all been converted.

**Figure 84. Regular simultaneous mode on 16 channels: triple ADC mode**



ai16055

### 15.9.3 Interleaved mode

This mode can be started only on a regular group (usually one channel). The external trigger source comes from the regular channel multiplexer of ADC1.

#### Dual ADC mode

After an external trigger occurs:

- ADC1 starts immediately
- ADC2 starts after a delay of several-ADC clock cycles

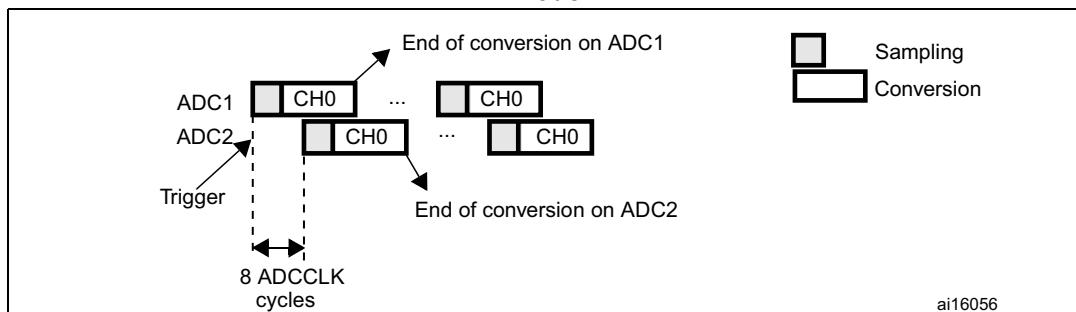
The minimum delay which separates 2 conversions in interleaved mode is configured in the DELAY bits in the ADC\_CCR register. However, an ADC cannot start a conversion if the complementary ADC is still sampling its input (only one ADC can sample the input signal at a given time). In this case, the delay becomes the sampling time + 2 ADC clock cycles. For instance, if  $\text{DELAY} = 5$  clock cycles and the sampling takes 15 clock cycles on both ADCs, then 17 clock cycles will separate conversions on ADC1 and ADC2).

If the CONT bit is set on both ADC1 and ADC2, the selected regular channels of both ADCs are continuously converted.

**Note:** *If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.*

After an EOC interrupt is generated by ADC2 (if enabled through the EOCIE bit) a 32-bit DMA transfer request is generated (if the DMA[1:0] bits in ADC\_CCR are equal to 0b10). This request first transfers the ADC2 converted data stored in the upper half-word of the ADC\_CDR 32-bit register into SRAM, then the ADC1 converted data stored in the register's lower half-word into SRAM.

**Figure 85. Interleaved mode on 1 channel in continuous conversion mode: dual ADC mode**



### Triple ADC mode

After an external trigger occurs:

- ADC1 starts immediately and
- ADC2 starts after a delay of several ADC clock cycles
- ADC3 starts after a delay of several ADC clock cycles referred to the ADC2 conversion

The minimum delay which separates 2 conversions in interleaved mode is configured in the DELAY bits in the ADC\_CCR register. However, an ADC cannot start a conversion if the complementary ADC is still sampling its input (only one ADC can sample the input signal at a given time). In this case, the delay becomes the sampling time + 2 ADC clock cycles. For instance, if  $\text{DELAY} = 5$  clock cycles and the sampling takes 15 clock cycles on the three ADCs, then 17 clock cycles will separate the conversions on ADC1, ADC2 and ADC3).

If the CONT bit is set on ADC1, ADC2 and ADC3, the selected regular channels of all ADCs are continuously converted.

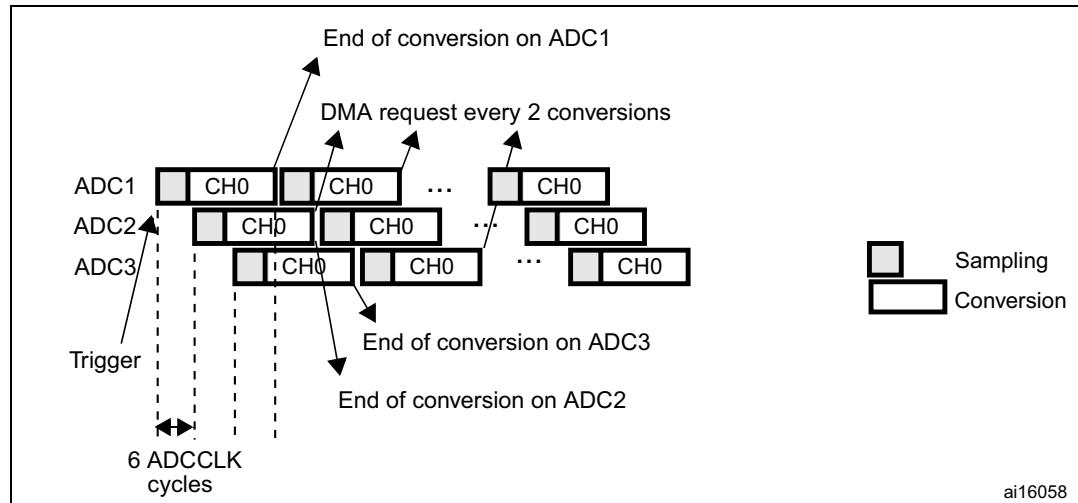
**Note:** *If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.*

In this mode a DMA request is generated each time 2 data items are available, (if the DMA[1:0] bits in the ADC\_CCR register are equal to 0b10). The request first transfers the

first converted data stored in the lower half-word of the ADC\_CDR 32-bit register to SRAM, then it transfers the second converted data stored in ADC\_CDR's upper half-word to SRAM. The sequence is the following:

- 1st request: ADC\_CDR[31:0] = ADC2\_DR[15:0] | ADC1\_DR[15:0]
- 2nd request: ADC\_CDR[31:0] = ADC1\_DR[15:0] | ADC3\_DR[15:0]
- 3rd request: ADC\_CDR[31:0] = ADC3\_DR[15:0] | ADC2\_DR[15:0]
- 4th request: ADC\_CDR[31:0] = ADC2\_DR[15:0] | ADC1\_DR[15:0], ...

**Figure 86. Interleaved mode on 1 channel in continuous conversion mode: triple ADC mode**



#### 15.9.4 Alternate trigger mode

This mode can be started only on an injected group. The source of external trigger comes from the injected group multiplexer of ADC1.

**Note:** *Regular conversions can be enabled on one or all ADCs. In this case the regular conversions are independent of each other. A regular conversion is interrupted when the ADC has to perform an injected conversion. It is resumed when the injected conversion is finished.*

*If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.*

*The time interval between 2 trigger events must be greater than or equal to 1 ADC clock period. The minimum time interval between 2 trigger events that start conversions on the same ADC is the same as in the single ADC mode.*

#### Dual ADC mode

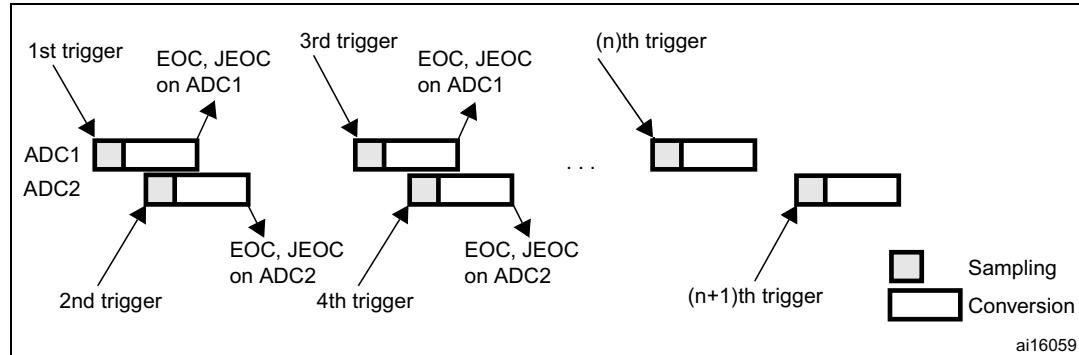
- When the 1st trigger occurs, all injected ADC1 channels in the group are converted
- When the 2nd trigger occurs, all injected ADC2 channels in the group are converted
- and so on

A JEOC interrupt, if enabled, is generated after all injected ADC1 channels in the group have been converted.

A JEOC interrupt, if enabled, is generated after all injected ADC2 channels in the group have been converted.

If another external trigger occurs after all injected channels in the group have been converted then the alternate trigger process restarts by converting the injected ADC1 channels in the group.

**Figure 87. Alternate trigger: injected group of each ADC**



If the injected discontinuous mode is enabled for both ADC1 and ADC2:

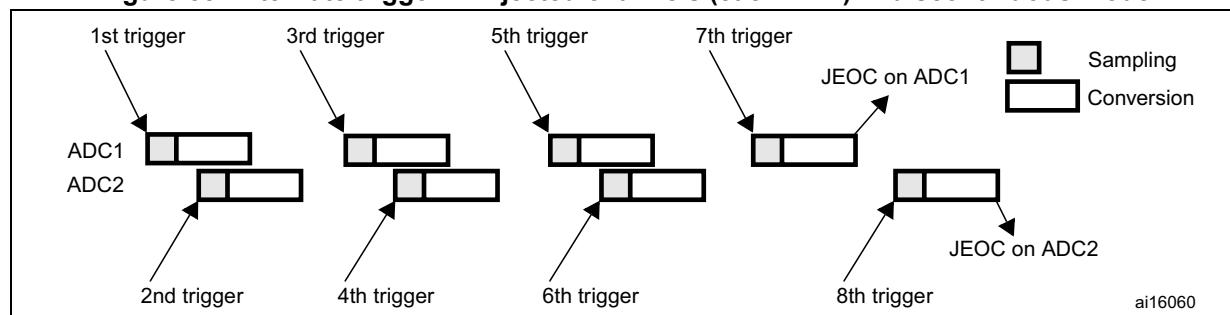
- When the 1st trigger occurs, the first injected ADC1 channel is converted.
- When the 2nd trigger occurs, the first injected ADC2 channel are converted
- and so on

A JEOC interrupt, if enabled, is generated after all injected ADC1 channels in the group have been converted.

A JEOC interrupt, if enabled, is generated after all injected ADC2 channels in the group have been converted.

If another external trigger occurs after all injected channels in the group have been converted then the alternate trigger process restarts.

**Figure 88. Alternate trigger: 4 injected channels (each ADC) in discontinuous mode**



### Triple ADC mode

- When the 1st trigger occurs, all injected ADC1 channels in the group are converted.
- When the 2nd trigger occurs, all injected ADC2 channels in the group are converted.
- When the 3rd trigger occurs, all injected ADC3 channels in the group are converted.
- and so on

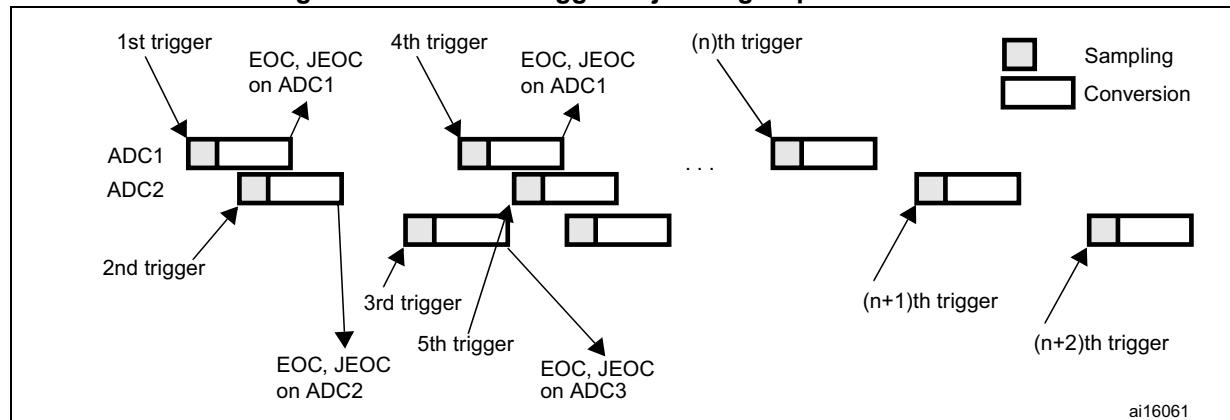
A JEOC interrupt, if enabled, is generated after all injected ADC1 channels in the group have been converted.

A JEOC interrupt, if enabled, is generated after all injected ADC2 channels in the group have been converted.

A JEOC interrupt, if enabled, is generated after all injected ADC3 channels in the group have been converted.

If another external trigger occurs after all injected channels in the group have been converted then the alternate trigger process restarts by converting the injected ADC1 channels in the group.

**Figure 89. Alternate trigger: injected group of each ADC**



### 15.9.5 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.*

### 15.9.6 Combined regular simultaneous + alternate trigger mode

It is possible to interrupt the simultaneous conversion of a regular group to start the alternate trigger conversion of an injected group. [Figure 90](#) shows the behavior of an alternate trigger interrupting a simultaneous regular conversion.

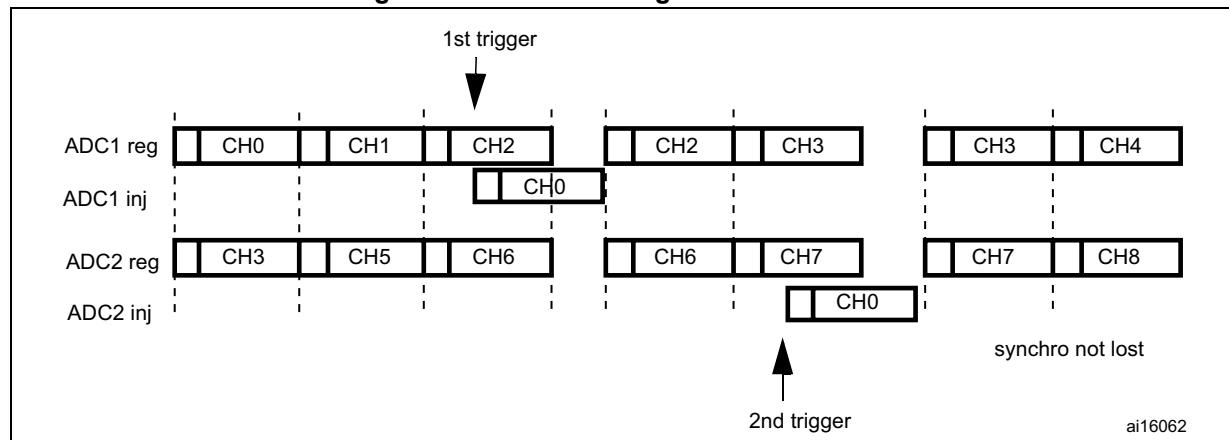
The injected alternate conversion is immediately started after the injected event. If regular conversion is already running, in order to ensure synchronization after the injected conversion, the regular conversion of all (master/slave) ADCs is stopped and resumed synchronously at the end of the injected conversion.

**Note:** *In combined regular simultaneous + alternate trigger mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the long conversion time of the 2 sequences (Dual ADC mode) /3 sequences (Triple ADC mode).*

*Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.*

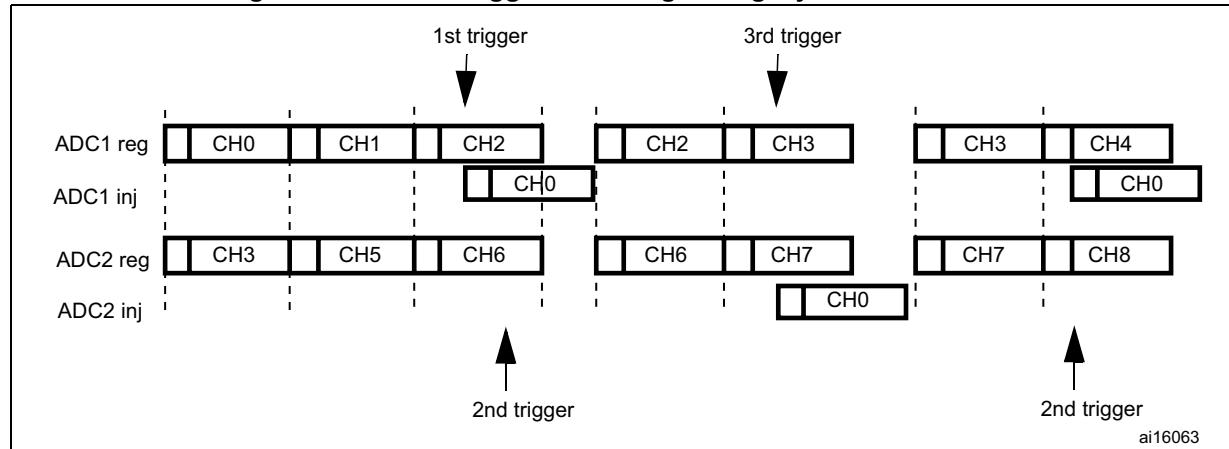
If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.

**Figure 90. Alternate + regular simultaneous**



If a trigger occurs during an injected conversion that has interrupted a regular conversion, it is ignored. [Figure 91](#) shows the behavior in this case (2nd trigger is ignored).

**Figure 91. Case of trigger occurring during injected conversion**



## 15.10 Temperature sensor

The temperature sensor can be used to measure the junction temperature ( $T_J$ ) of the device.

- On STM32F75xxx and STM32F74xxx 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 92 shows the block diagram of the temperature sensor.

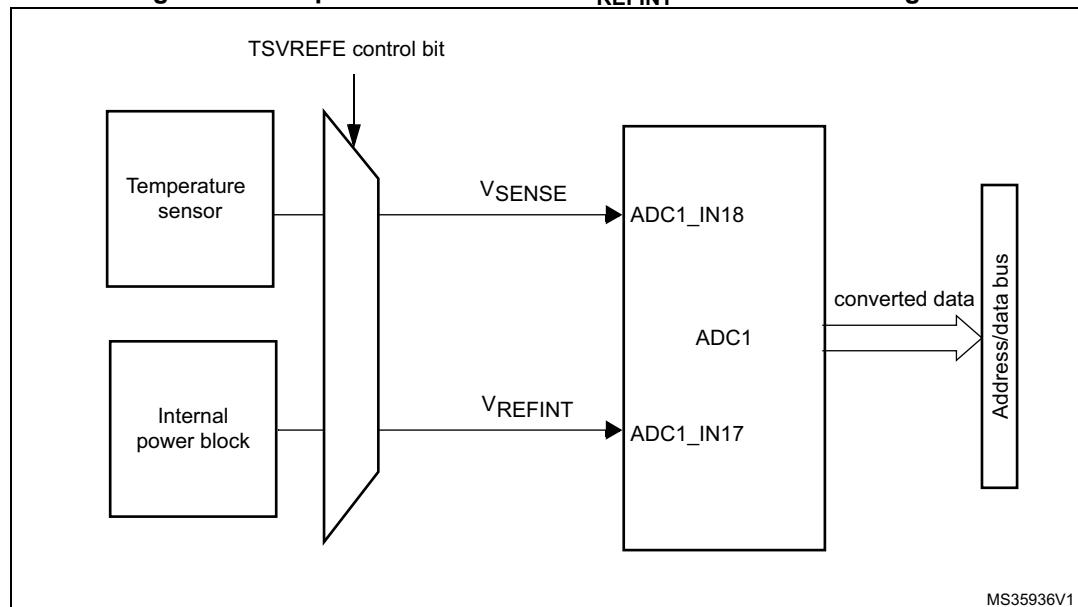
When not in use, the sensor can be put in power down mode.

Note: The TSVREFE bit must be set to enable the conversion of both internal channels: the ADC1\_IN18 (temperature sensor) and the ADC1\_IN17 (VREFINT).

### Main features

- Supported temperature range: -40 to 125 °C
- Precision: ±1.5 °C

Figure 92. Temperature sensor and V<sub>REFINT</sub> channel block diagram



1.  $V_{SENSE}$  is input to ADC1\_IN18.

### Reading the temperature

To use the sensor:

3. Select ADC1\_IN18 input channel.
4. Select a sampling time greater than the minimum sampling time specified in the datasheet.
5. Set the TSVREFE bit in the ADC\_CCR register to wake up the temperature sensor from power down mode
6. Start the ADC conversion by setting the SWSTART bit (or by external trigger)
7. Read the resulting  $V_{SENSE}$  data in the ADC data register
8. Calculate the temperature using the following formula:

$$\text{Temperature (in } ^\circ\text{C)} = \{(V_{SENSE} - V_{25}) / \text{Avg\_Slope}\} + 25$$

Where:

- $V_{25}$  =  $V_{SENSE}$  value for  $25^\circ\text{C}$
- Avg\_Slope = average slope of the temperature vs.  $V_{SENSE}$  curve (given in  $\text{mV}/^\circ\text{C}$  or  $\mu\text{V}/^\circ\text{C}$ )

Refer to the datasheet electrical characteristics section for the actual values of  $V_{25}$  and Avg\_Slope.

**Note:**

*The sensor has a startup time after waking from power down mode before it can output  $V_{SENSE}$  at the correct level. The ADC also has a startup time after power-on, so to minimize the delay, the ADON and TSVREFE bits should be set at the same time.*

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  $45^\circ\text{C}$  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.

## 15.11

### Battery charge monitoring

The VBATE 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 VBATE is set, the bridge is automatically enabled to connect:

- $V_{BAT}/4$  to the ADC1\_IN18 input channel

**Note:**

*The  $V_{BAT}$  and temperature sensor are connected to the same ADC internal channel (ADC1\_IN18). Only one conversion, either temperature sensor or  $V_{BAT}$ , must be selected at a time. When both conversion are enabled simultaneously, only the  $V_{BAT}$  conversion is performed.*

## 15.12 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:

- JSTRT (Start of conversion for channels of an injected group)
- STRT (Start of conversion for channels of a regular group)

**Table 98. 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

## 15.13 ADC registers

Refer to [Section 1.2 on page 63](#) 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).

### 15.13.1 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 = 1), or sequence of conversions not complete (EOCS = 0)
- 1: Conversion complete (EOCS = 1), or sequence of conversions complete (EOCS = 0)

#### 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

### 15.13.2 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[1:0]		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

### 15.13.3 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[1:0]		EXTSEL[3:0]				Res.	JSWSTART	JEXTEN[1:0]		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	ALIGN	EOCS	DDS	DMA	Res.	Res.	Res.	Res.	Res.	Res.	CONT	ADON
				rw	rw	rw	rw							rw	rw

Bit 31 Reserved, must be kept at reset value.

Bit 30 **SWSTART**: Start conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as the conversion starts.

0: Reset state

1: Starts conversion of regular channels

*Note: This bit can be set only when ADON = 1 otherwise no conversion is launched.*

Bits 29:28 **EXTEN**: External trigger enable for regular channels

These bits are set and cleared by software to select the external trigger polarity and enable the trigger of a regular group.

00: Trigger detection disabled

01: Trigger detection on the rising edge

10: Trigger detection on the falling edge

11: Trigger detection on both the rising and falling edges

Bits 27:24 **EXTSEL[3:0]**: External event select for regular group

These bits select the external event used to trigger the start of conversion of a regular group:  
Refer to [Section 15.6: Conversion on external trigger and trigger polarity](#).

Bit 23 Reserved, must be kept at reset value.

Bit 22 **JSWSTART**: Start conversion of injected channels

This bit is set by software and cleared by hardware as soon as the conversion starts.

0: Reset state

1: Starts conversion of injected channels

This bit can be set only when ADON = 1 otherwise no conversion is launched.

Bits 21:20 **JEXTEN**: External trigger enable for injected channels

These bits are set and cleared by software to select the external trigger polarity and enable the trigger of an injected group.

00: Trigger detection disabled

01: Trigger detection on the rising edge

10: Trigger detection on the falling edge

11: Trigger detection on both the rising and falling edges

Bits 19:16 **JEXTSEL[3:0]**: External event select for injected group

These bits select the external event used to trigger the start of conversion of an injected group.  
Refer to [Section 15.6: Conversion on external trigger and trigger polarity](#).

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

Bit 11 **ALIGN**: Data alignment

This bit is set and cleared by software. Refer to [Figure 77](#) and [Figure 78](#).

0: Right alignment

1: Left alignment

Bit 10 **EOCS**: End of conversion selection

This bit is set and cleared by software.

0: The EOC bit is set at the end of each sequence of regular conversions. Overrun detection is enabled only if DMA=1.

1: The EOC bit is set at the end of each regular conversion. Overrun detection is enabled.

Bit 9 **DDS**: DMA disable selection (for single ADC mode)

This bit is set and cleared by software.

0: No new DMA request is issued after the last transfer (as configured in the DMA controller)

1: DMA requests are issued as long as data are converted and DMA=1

Bit 8 **DMA**: Direct memory access mode (for single ADC mode)

This bit is set and cleared by software. Refer to the DMA controller chapter for more details.

0: DMA mode disabled

1: DMA mode enabled

Bits 7:2 Reserved, must be kept at reset value.

Bit 1 **CONT**: Continuous conversion

This bit is set and cleared by software. If it is set, conversion takes place continuously until it is cleared.

0: Single conversion mode

1: Continuous conversion mode

Bit 0 **ADON**: A/D Converter ON / OFF

This bit is set and cleared by software.

0: Disable ADC conversion and go to power down mode

1: Enable ADC

**15.13.4 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

### 15.13.5 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	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

### 15.13.6 ADC injected channel data offset register x (ADC\_JOFRx) (x=1..4)

Address offset: 0x14-0x20

Reset value: 0x0000 0000

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

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

Bits 11:0 **JOFFSETx[11:0]**: Data offset for injected channel x

These bits are written by software to define the offset to be subtracted from the raw converted data when converting injected channels. The conversion result can be read from in the ADC\_JDRx registers.

### 15.13.7 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.												
				rw											
HT[11:0]															

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.*

### 15.13.8 ADC watchdog lower threshold register (ADC\_LTR)

Address offset: 0x28

Reset value: 0x0000 0000

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

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.*

### 15.13.9 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]							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]						SQ14[4:0]						SQ13[4:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 23:20 **L[3:0]: Regular channel sequence length**

These bits are written by software to define the total number of conversions in the regular channel conversion sequence.

0000: 1 conversion

0001: 2 conversions

...

1111: 16 conversions

Bits 19:15 **SQ16[4:0]: 16th conversion in regular sequence**

These bits are written by software with the channel number (0..18) assigned as the 16th in the conversion sequence.

Bits 14:10 **SQ15[4:0]: 15th conversion in regular sequence**

Bits 9:5 **SQ14[4:0]: 14th conversion in regular sequence**

Bits 4:0 **SQ13[4:0]: 13th conversion in regular sequence**

### 15.13.10 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 29:25 **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

### 15.13.11 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]	
		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
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

### 15.13.12 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.

### 15.13.13 ADC injected data register x (ADC\_JDRx) (x = 1..4)

Address offset: 0x3C - 0x48

Reset value: 0x0000 0000

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

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

Bits 15:0 **JDATA[15:0]**: Injected data

These bits are read-only. They contain the conversion result from injected channel x. The data are left -or right-aligned as shown in [Figure 77](#) and [Figure 78](#).

#### 15.13.14 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 77](#) and [Figure 78](#).

#### 15.13.15 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.

### 15.13.16 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.*

### 15.13.17 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](#).

### 15.13.18 ADC register map

The following table summarizes the ADC registers.

Table 99. 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 100. ADC register map and reset values for each ADC

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00	ADC_SR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	OVR	5
	Reset value	—	—	—	—	—	—	—	—	—	—	—	—	—	—	STRTR	4
0x04	ADC_CR1	Res	Res	Res	Res	Res	OVRIE	RES[1:0]	AWDEN	JAWDEN	Res	Res	Res	Res	DISC NUM [2:0]	DISCEN	Res
	Reset value	—	—	—	—	—	0	0	0	0	—	—	—	—	0 0 0	0 0 0	Res
AWDCH[4:0]																	

**Table 100. ADC register map and reset values for each ADC (continued)**

Offset	Register	Register Descriptions																
0x08	<b>ADC_CR2</b>	Res.	<b>31</b>															
		Reset value	0	SWSTART	<b>30</b>													
0x0C	<b>ADC_SMPR1</b>	Sample time bits SMPX_X																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x10	<b>ADC_SMPR2</b>	Sample time bits SMPX_X																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x14	<b>ADC_JOFR1</b>	JOFFSET1[11:0]																
		Reset value													0	0	0	
0x18	<b>ADC_JOFR2</b>	JOFFSET2[11:0]																
		Reset value													0	0	0	
0x1C	<b>ADC_JOFR3</b>	JOFFSET3[11:0]																
		Reset value													0	0	0	
0x20	<b>ADC_JOFR4</b>	JOFFSET4[11:0]																
		Reset value													0	0	0	
0x24	<b>ADC_HTR</b>	HT[11:0]																
		Reset value													1	1	1	
0x28	<b>ADC_LTR</b>	LT[11:0]																
		Reset value													0	0	0	
0x2C	<b>ADC_SQR1</b>	Regular channel sequence SQx_x bits																
		Reset value		L[3:0]														
0x30	<b>ADC_SQR2</b>	Regular channel sequence SQx_x bits																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x34	<b>ADC_SQR3</b>	Regular channel sequence SQx_x bits																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x38	<b>ADC_JSQR</b>	Injected channel sequence JSQx_x bits																
		Reset value		JL[1:0]														
0x3C	<b>ADC_JDR1</b>	JDATA[15:0]																
		Reset value													0	0	0	
0x40	<b>ADC_JDR2</b>	JDATA[15:0]																
		Reset value													0	0	0	
0x44	<b>ADC_JDR3</b>	JDATA[15:0]																
		Reset value													0	0	0	
0x48	<b>ADC_JDR4</b>	JDATA[15:0]																
		Reset value													0	0	0	
0x4C	<b>ADC_DR</b>	Regular DATA[15:0]																
		Reset value													0	0	0	

**Table 101. 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.														
												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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x08	<b>ADC_CDR</b>	Regular DATA2[15:0]															Regular DATA1[15:0]															MULTI [4:0]	
0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 16 Digital-to-analog converter (DAC)

### 16.1 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.

### 16.2 DAC main features

- Two DAC converters: one output channel each
- Left or right data alignment in 12-bit mode
- Synchronized update capability
- Noise-wave generation
- Triangular-wave generation
- Dual DAC channel for independent or simultaneous conversions
- DMA capability for each channel
- DMA underrun error detection
- External triggers for conversion
- Input voltage reference,  $V_{REF+}$

*Figure 93* shows the block diagram of a DAC channel and *Table 102* gives the pin description.

Figure 93. DAC channel block diagram

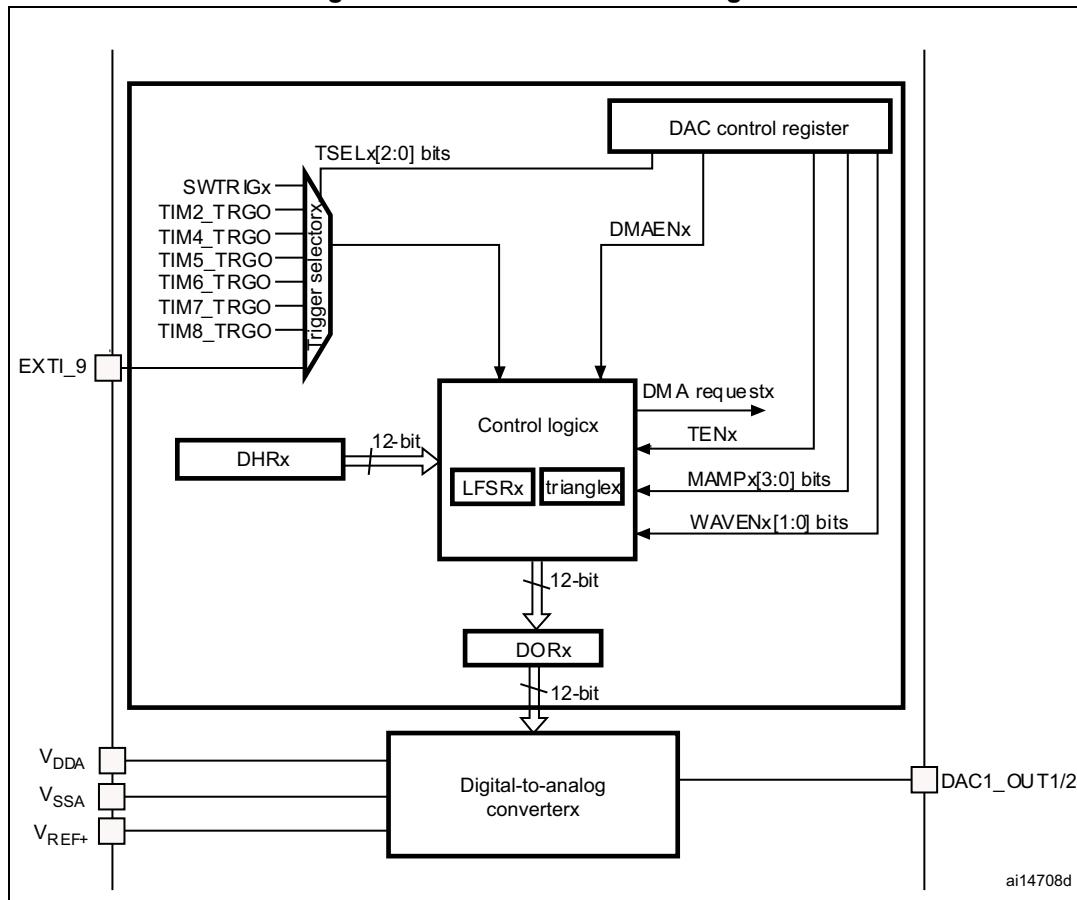


Table 102. DAC pins

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 channel <sub>x</sub> analog output

**Note:** Once the DAC channel<sub>x</sub> 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).

## 16.3 DAC functional description

### 16.3.1 DAC channel enable

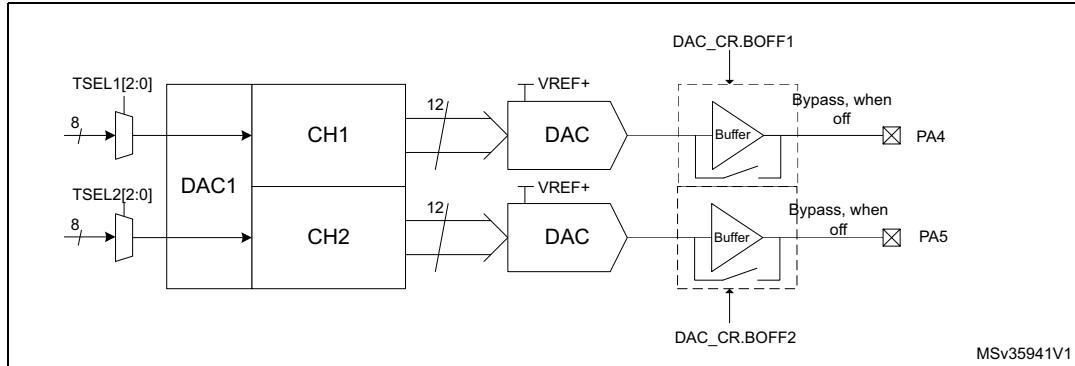
Each DAC channel can be powered on by setting its corresponding EN<sub>x</sub> bit in the DAC\_CR register. The DAC channel is then enabled after a startup time t<sub>WAKEUP</sub>.

**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.

### 16.3.2 DAC output buffer enable

The DAC integrates two output buffers that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. Each DAC channel output buffer can be enabled and disabled using the corresponding BOFFx bit in the DAC\_CR register.

**Figure 94. DAC output buffer connection**



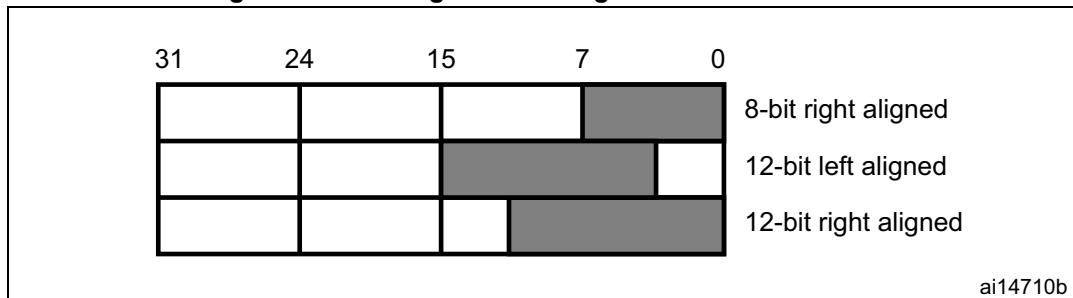
MSv35941V1

### 16.3.3 DAC data format

Depending on the selected configuration mode, the data have to be written into the specified register as described below:

- Single DAC channelx, there are three possibilities:
  - 8-bit right alignment: the software has to load data into the DAC\_DHR8Rx [7:0] bits (stored into the DHRx[11:4] bits)
  - 12-bit left alignment: the software has to load data into the DAC\_DHR12Lx [15:4] bits (stored into the DHRx[11:0] bits)
  - 12-bit right alignment: the software has to load data into the DAC\_DHR12Rx [11:0] bits (stored into the DHRx[11:0] bits)

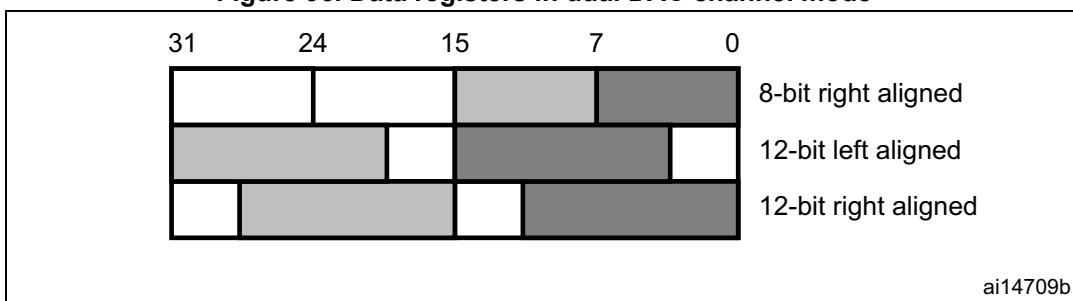
Depending on the loaded DAC\_DHRyyxx register, the data written by the user is shifted and stored into the corresponding DHRx (data holding registerx, which are internal non-memory-mapped registers). The DHRx register is then loaded into the DORx register either automatically, by software trigger or by an external event trigger.

**Figure 95. Data registers in single DAC channel mode**

ai14710b

- Dual DAC channels, there are three possibilities:
  - 8-bit right alignment: data for DAC channel1 to be loaded into the DAC\_DHR8RD [7:0] bits (stored into the DHR1[11:4] bits) and data for DAC channel2 to be loaded into the DAC\_DHR8RD [15:8] bits (stored into the DHR2[11:4] bits)
  - 12-bit left alignment: data for DAC channel1 to be loaded into the DAC\_DHR12LD [15:4] bits (stored into the DHR1[11:0] bits) and data for DAC channel2 to be loaded into the DAC\_DHR12LD [31:20] bits (stored into the DHR2[11:0] bits)
  - 12-bit right alignment: data for DAC channel1 to be loaded into the DAC\_DHR12RD [11:0] bits (stored into the DHR1[11:0] bits) and data for DAC channel2 to be loaded into the DAC\_DHR12LD [27:16] bits (stored into the DHR2[11:0] bits)

Depending on the loaded DAC\_DHRyyD register, the data written by the user is shifted and stored into DHR1 and DHR2 (data holding registers, which are internal non-memory-mapped registers). The DHR1 and DHR2 registers are then loaded into the DOR1 and DOR2 registers, respectively, either automatically, by software trigger or by an external event trigger.

**Figure 96. Data registers in dual DAC channel mode**

ai14709b

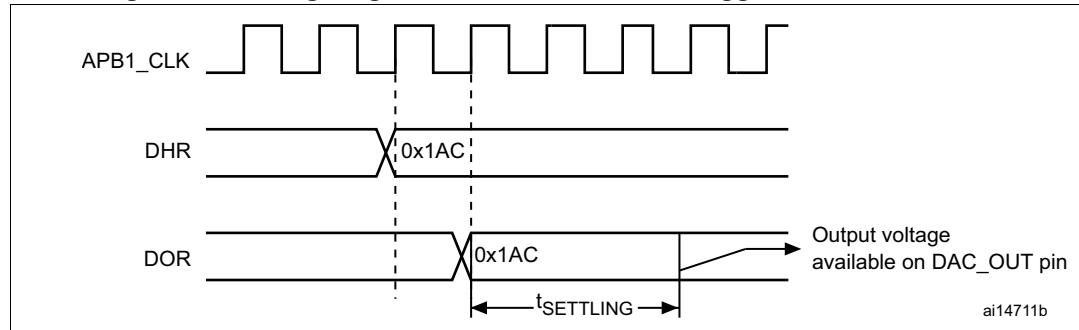
#### 16.3.4 DAC conversion

The DAC\_DORx cannot be written directly and any data transfer to the DAC channelx must be performed by loading the DAC\_DHRx register (write to DAC\_DHR8Rx, DAC\_DHR12Lx, DAC\_DHR12Rx, DAC\_DHR8RD, DAC\_DHR12LD or DAC\_DHR12RD).

Data stored in the DAC\_DHRx register are automatically transferred to the DAC\_DORx register after one APB1 clock cycle, if no hardware trigger is selected (TENx bit in DAC\_CR register is reset). However, when a hardware trigger is selected (TENx bit in DAC\_CR register is set) and a trigger occurs, the transfer is performed three APB1 clock cycles later.

When DAC\_DORx is loaded with the DAC\_DHRx contents, the analog output voltage becomes available after a time  $t_{SETTLING}$  that depends on the power supply voltage and the analog output load.

**Figure 97. Timing diagram for conversion with trigger disabled TEN = 0**



### 16.3.5 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:

$$\text{DACoutput} = V_{REF} \times \frac{\text{DOR}}{4096}$$

### 16.3.6 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 103](#).

**Table 103. 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.*

### 16.3.7 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.

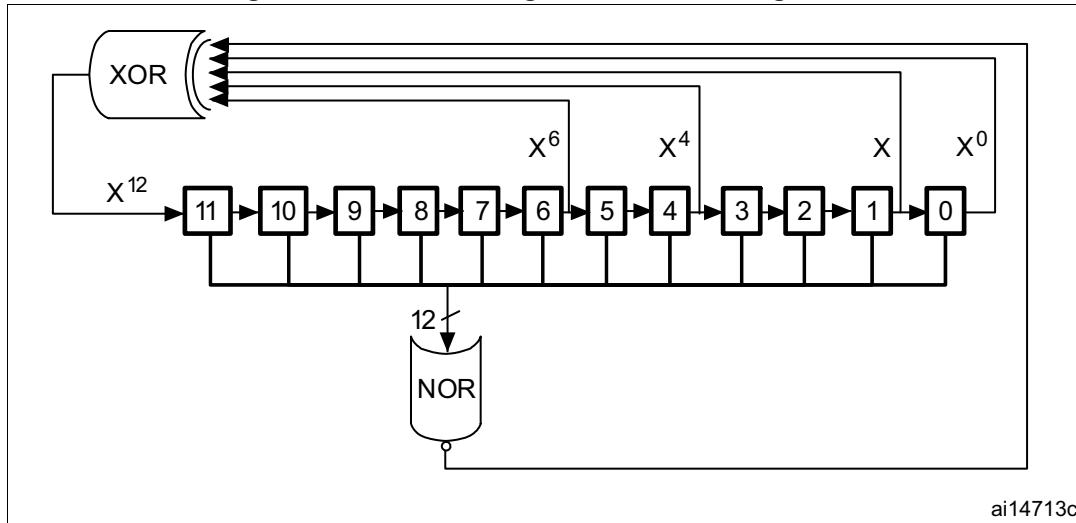
The software should clear the DMAUDRx flag by writing “1”, clear the DMAEN bit of the used DMA stream and re-initialize both DMA and DAC channelx to restart the transfer correctly. The software should modify the DAC trigger conversion frequency or lighten the DMA workload to avoid a new DMA underrun. 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.

### 16.3.8 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 WAVEx[1:0] to “01”. The preloaded value in LFSR is 0xAAA. This register is updated three APB1 clock cycles after each trigger event, following a specific calculation algorithm.

Figure 98. DAC LFSR register calculation algorithm

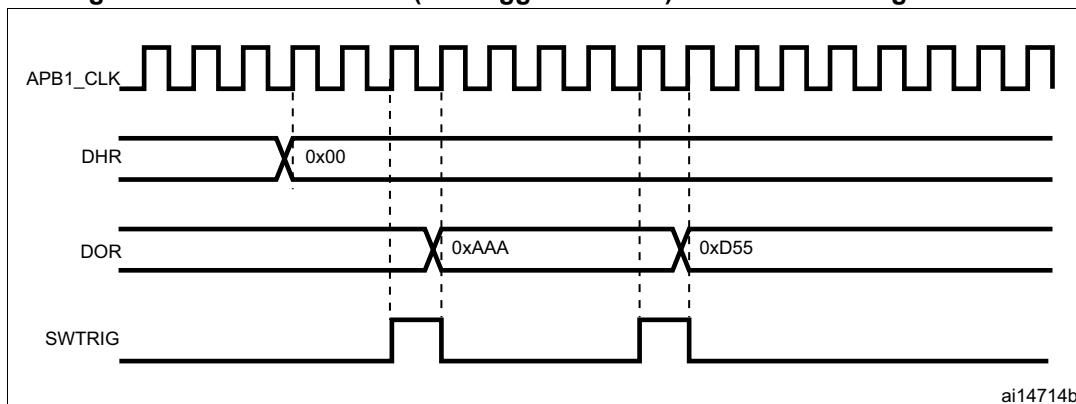


The LFSR value, that may be masked partially or totally by means of the MAMPx[3:0] bits in the DAC\_CR register, is added up to the DAC\_DHRx contents without overflow and this value is then stored into the DAC\_DORx register.

If LFSR is 0x0000, a ‘1’ is injected into it (antilock-up mechanism).

It is possible to reset LFSR wave generation by resetting the WAVEEx[1:0] bits.

Figure 99. DAC conversion (SW trigger enabled) with LFSR wave generation



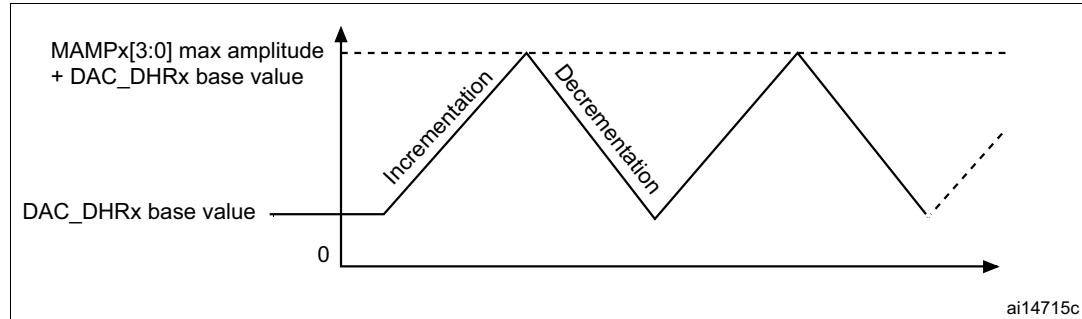
**Note:** The DAC trigger must be enabled for noise generation by setting the TENx bit in the DAC\_CR register.

### 16.3.9 Triangle-wave generation

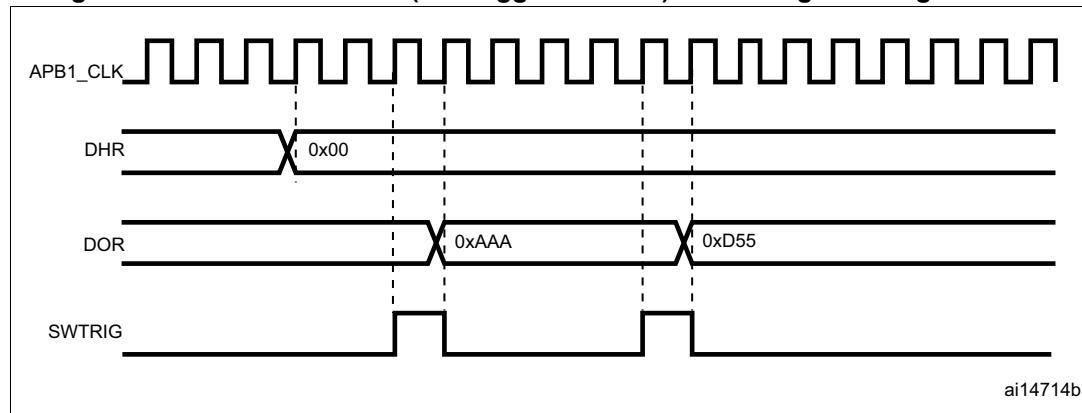
It is possible to add a small-amplitude triangular waveform on a DC or slowly varying signal. DAC triangle-wave generation is selected by setting WAVEEx[1:0] to “10”. The amplitude is configured through the MAMPx[3:0] bits in the DAC\_CR register. An internal triangle counter is incremented three APB1 clock cycles after each trigger event. The value of this counter is then added to the DAC\_DHRx register without overflow and the sum is stored into the DAC\_DORx register. The triangle counter is incremented as long as it is less than the maximum amplitude defined by the MAMPx[3:0] bits. Once the configured amplitude is reached, the counter is decremented down to 0, then incremented again and so on.

It is possible to reset triangle wave generation by resetting the WAVEx[1:0] bits.

**Figure 100. DAC triangle wave generation**



**Figure 101. DAC conversion (SW trigger enabled) with triangle wave generation**



#### Note:

The DAC trigger must be enabled for noise generation by setting the TENx bit in the DAC\_CR register.

The MAMPx[3:0] bits must be configured before enabling the DAC, otherwise they cannot be changed.

## 16.4 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.

#### 16.4.1 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).

#### 16.4.2 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.

#### 16.4.3 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.

#### 16.4.4 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.

#### 16.4.5 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.

#### 16.4.6 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.

#### 16.4.7 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).

#### 16.4.8 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.

#### 16.4.9 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.

### 16.4.10 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.

### 16.4.11 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.

## 16.5 DAC registers

Refer to [Section 1 on page 63](#) for a list of abbreviations used in register descriptions.

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

### 16.5.1 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

### 16.5.2 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.*

### 16.5.3 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.

#### 16.5.4 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.

#### 16.5.5 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.

### 16.5.6 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
Res.				DACC2DHR[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 **DACC2DHR[11:0]**: DAC channel2 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

### 16.5.7 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]												Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				

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

Bits 15:4 **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.

### 16.5.8 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.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.								DACC2DHR[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 7:0 **DACC2DHR[7:0]**: DAC channel2 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel2.

### 16.5.9 Dual DAC 12-bit right-aligned data holding register (DAC\_DHR12RD)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
Res.	Res.	Res.	Res.	DACC2DHR[11:0]														
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Res.	Res.	Res.	Res.	DACC1DHR[11:0]														
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

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

Bits 27:16 **DACC2DHR[11:0]**: DAC channel2 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

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

Bits 11:0 **DACC1DHR[11:0]**: DAC channel1 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

### 16.5.10 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.

### 16.5.11 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.

### 16.5.12 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.

### 16.5.13 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.

### 16.5.14 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.

### **16.5.15 DAC register map**

*Table 104* summarizes the DAC registers.

**Table 104. DAC register map**

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.



## 17 Digital camera interface (DCMI)

### 17.1 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).

### 17.2 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

### 17.3 DCMI functional description

The digital camera interface is a synchronous parallel interface that can receive high-speed data flows. It consists of up to 14 data lines (DCMI\_D[13:0]) 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.

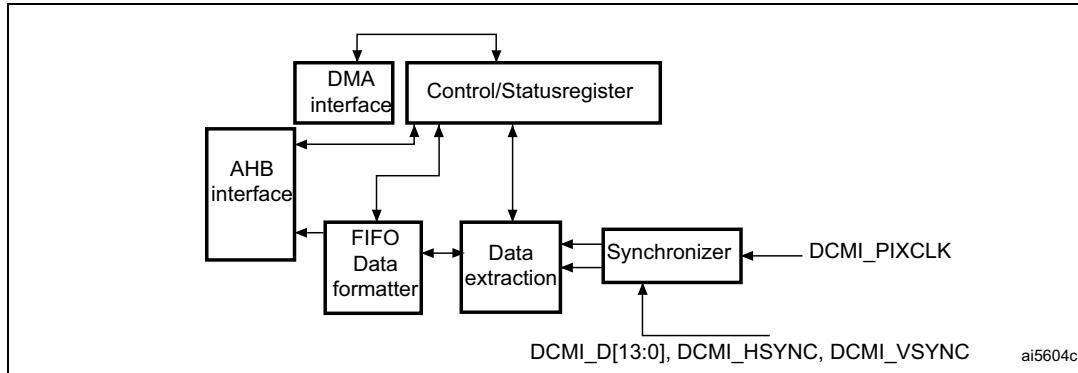
The data received from the camera can be organized in lines/frames (raw YUB/RGB/Bayer modes) or can be a sequence of JPEG images. To enable JPEG image reception, the JPEG bit (bit 3 of DCMI\_CR register) must be set.

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.

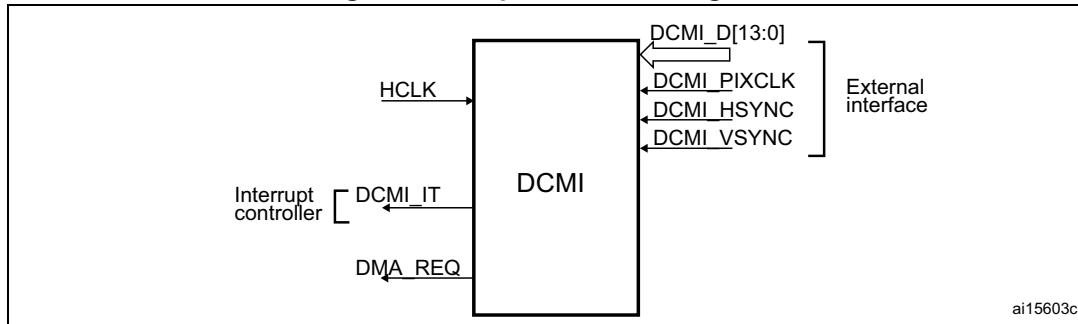
### 17.3.1 DCMI block diagram

*Figure 102* shows the DCMI block diagram.

**Figure 102. DCMI block diagram**



**Figure 103. Top-level block diagram**



### 17.3.2 DCMI pins

The following table shows DCMI pins.

**Table 105. DCMI input/output pins**

Mode	Pin name	Signal type	Description
8 bits	DCMI_D[7:0]		
10 bits	DCMI_D[9:0]		
12 bits	DCMI_D[11:0]	Inputs	DCMI data
14 bits	DCMI_D[13:0]		
	DCMI_PIXCLK	Input	Pixel clock
	DCMI_HSYNC	Input	Horizontal synchronization / Data valid
	DCMI_VSYNC	Input	Vertical synchronization

### 17.3.3 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.

#### 17.3.4 DCMI DMA interface

The DMA interface is active when the CAPTURE bit of 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.

#### 17.3.5 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 of the DCMI\_CR register. If less than 14 bits are used, the unused input pins must be connected to ground.

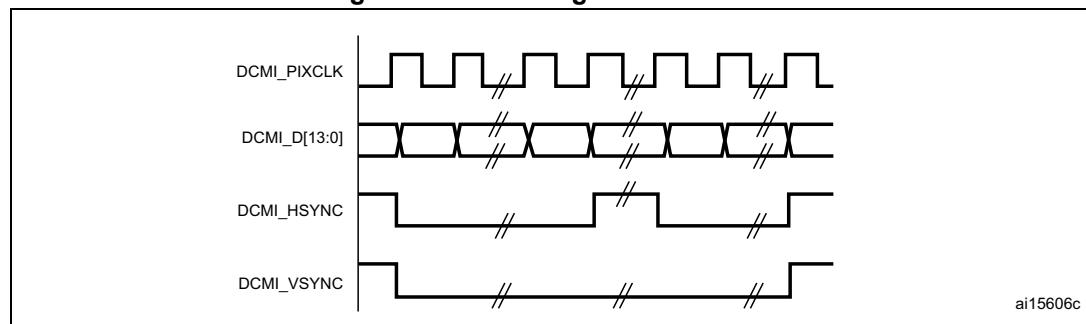
DCMI pins are shown in [Table 105](#).

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 104. 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] = 00 in DCMI\_CR the interface captures 8 LSBs at its input (DCMI\_D[7:0]) 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. The table below gives an example of the positioning of captured data bytes in two 32-bit words.

**Table 106. 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] = 01 in DCMI\_CR, the camera interface captures 10-bit data at its input DCMI\_D[9:0] and stores them as the 10 least significant bits of a 16-bit word. The remaining most significant bits of 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 the table below.

**Table 107. 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] = 10 in DCMI\_CR, the camera interface captures the 12-bit data at its input DCMI\_D[11:0] 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 the table below.

**Table 108. 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] = 11 in DCMI\_CR, the camera interface captures the 14-bit data at its input DCMI\_D[13:0] 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 the table below.

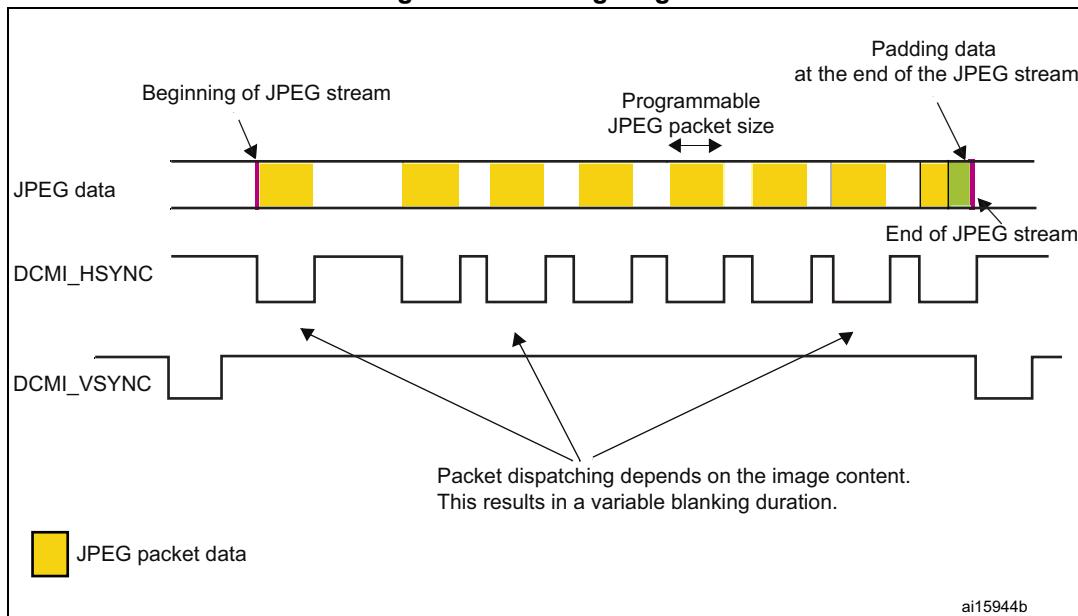
**Table 109. Positioning of captured data bytes in 32-bit words (14-bit width)**

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]

### 17.3.6 DCMI 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 must be cleared).

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 105](#) shows the corresponding timing diagram.

**Figure 105. Timing diagram**

#### 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 must be cleared). 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 17.5.7: 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  $\leq$  0xFF
- FE  $\leq$  0xFF
- LS  $\leq$  SAV (active)
- LE  $\leq$  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. A bit can therefore be selected 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.

## 17.3.7 DCMI capture modes

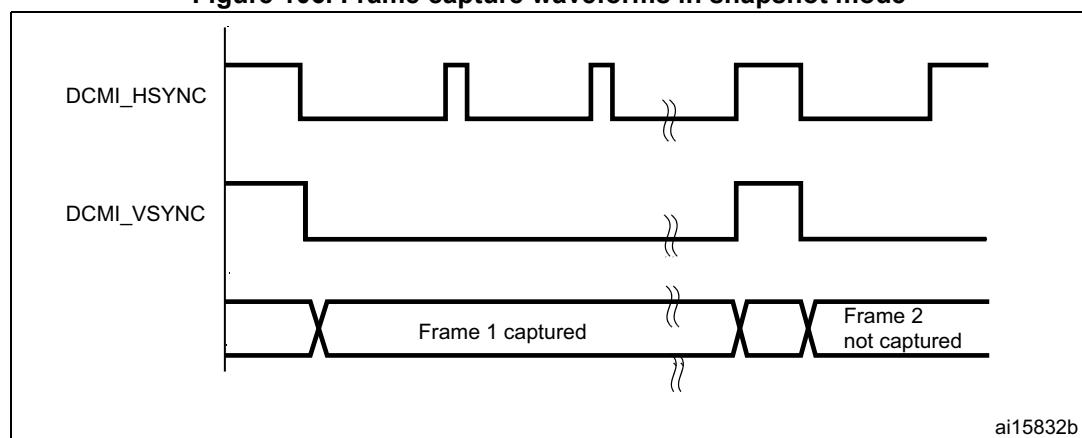
This interface supports two types of capture: snapshot (single frame) and continuous grab.

### Snapshot mode (single frame)

In this mode, a single frame is captured (CM = 1 of 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.

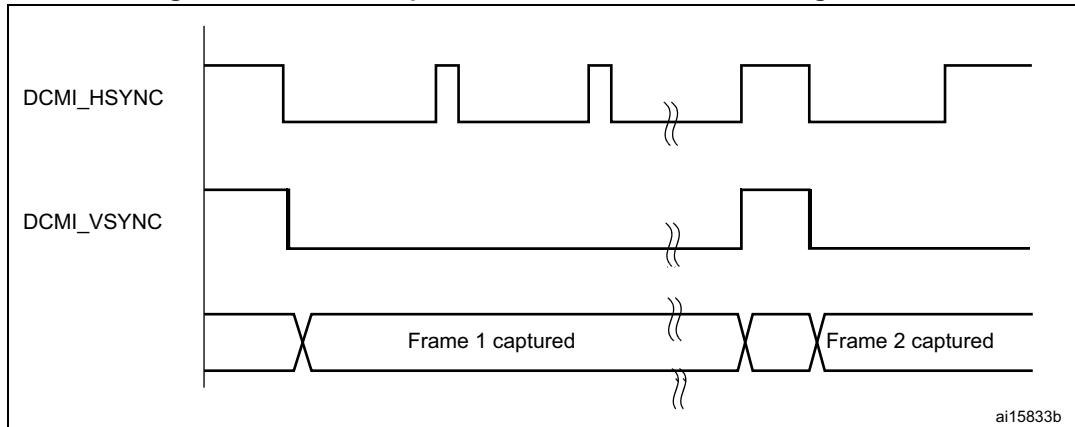
**Figure 106. Frame capture waveforms in snapshot mode**



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 107. 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.

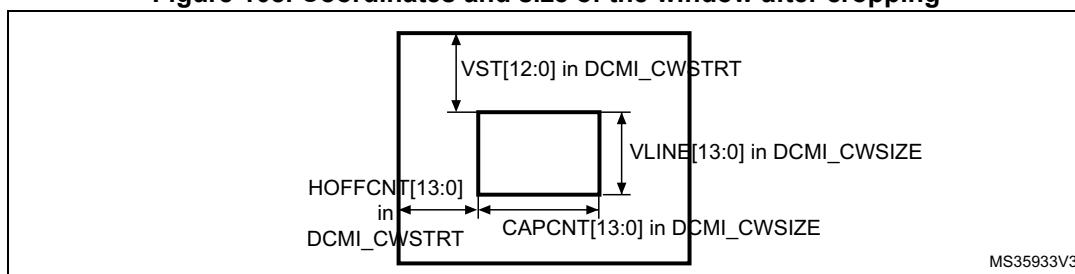
In continuous grab mode, the FCRC[1:0] bits in DCMI\_CR can be configured to grab all pictures, every second picture or one out of four pictures to decrease the frame capture rate.

**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.*

### 17.3.8 DCMI crop feature

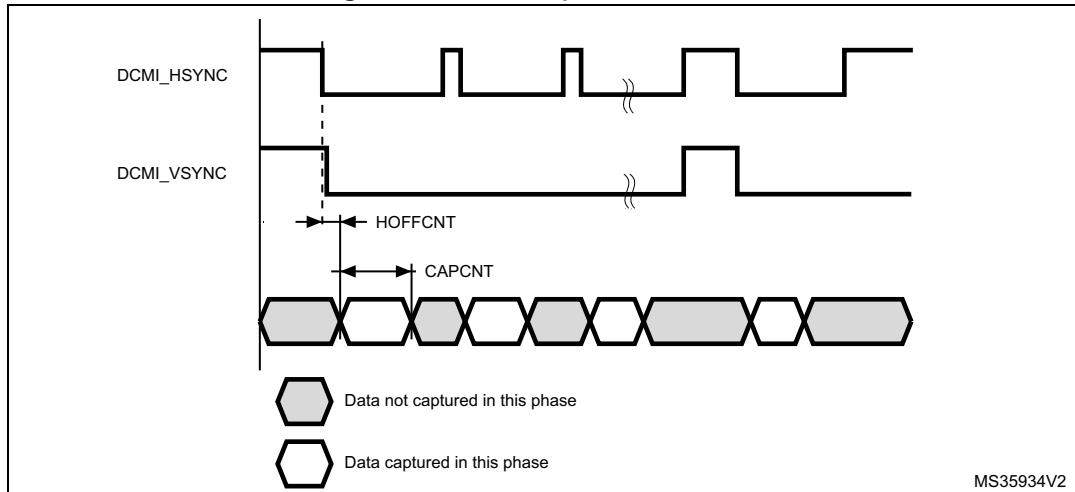
With the crop feature, the camera interface can select a rectangular window from the received image. The start (upper left corner) coordinates and size (horizontal dimension in number of pixel clocks and vertical dimension in number of lines) are specified using two 32-bit registers (DCMI\_CWSTRT and DCMI\_CWSIZE). The size of the window is specified in number of pixel clocks (horizontal dimension) and in number of lines (vertical dimension).

**Figure 108. 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 DCMI\_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 109. 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.

### 17.3.9 DCMI JPEG format

To allow JPEG image reception, it is necessary to set the JPEG bit of 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. This case must be carefully handled 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 zeros and a DMA request is generated.

The crop feature and embedded synchronization codes cannot be used in JPEG format.

### 17.3.10 DCMI FIFO

A 8-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.

### 17.3.11 DCMI data format description

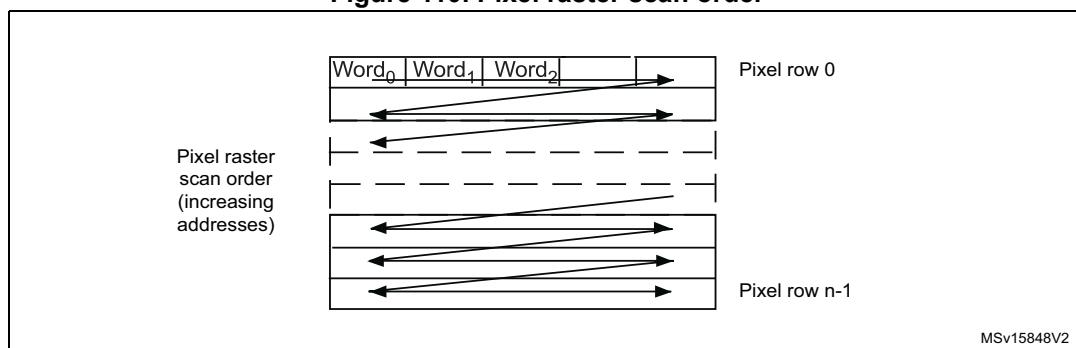
#### Data formats

The following types of data are supported:

- 8/10/12/14-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 monochrome, RGB and YCbCr, the frame buffer is stored in raster mode. 32-bit words are used. Only the little-endian format is supported.

**Figure 110. Pixel raster scan order**



#### Monochrome format

Characteristics:

- Raster format
- 8 bits per pixel

The table below shows how the data are stored.

**Table 110. 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

#### RGB format

Characteristics:

- Raster format
- RGB
- Interleaved: one buffer: R, G and B interleaved (such as BRGBRGRG)
- 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 gray-scale 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.

The table below shows how the data are stored.

**Table 111. 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

### YCbCr format

Characteristics:

- Raster format
- YCbCr 4:2:2
- Interleaved: one buffer: Y, Cb and Cr interleaved (such as CbYCrYCbYCr)

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 the table below.

**Table 112. 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

### 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 the luma component of each pixel, encoded in 8 bits, is stored as shown in [Table 113](#).

The result is a monochrome image having the same resolution as the original YCbCr data.

**Table 113. 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

### Half resolution image extraction

This is a modification of the previous reception modes, being applicable to monochrome, RGB or Y extraction modes.

This mode is used to only store a half resolution image. It is selected through OELS and LSM control bits.

## 17.4 DCMI interrupts

Five interrupts are generated. All interrupts are maskable by software. The global interrupt (DCMI\_IT) is the OR of all the individual interrupts. The table below gives the list of all interrupts.

**Table 114. DCMI interrupts**

Interrupt acronym	Interrupt event	Event flag	Enable control bit	Interrupt clear method	Exits Sleep mode	Exists Stop and Standby modes
DCMI_IT	End of line	LINE_RIS	LINE_IE	Set LINE_ISC	Yes	No
	End of frame capture	FRAME_RIS	FRAME_IE	Set FRAME_ISC	Yes	No
	Overrun of data reception	OVR_RIS	OVR_IE	Set OVR_ISC	Yes	No
	Synchronization frame	VSYNC_RIS	VSYNC_IE	Set VSYNC_ISC	Yes	No
	Detection of an error in the embedded synchronization frame detection	ERR_RIS	ERR_IE	Set ERR_ISC	Yes	No

## 17.5 DCMI registers

Refer to [Section 1.2 on page 63](#) for list of abbreviations used in register descriptions. All DCMI registers must be accessed as 32-bit words, otherwise a bus error occurs.

### 17.5.1 DCMI control register (DCMI\_CR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OEELS	LSM	OEBS	BSM[1:0]	
											rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	ENABL E	Res.	Res.	EDM[1:0]		FCRC[1:0]		VSPOL	HSPOL	PCKPO L	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 **OELS:** Odd/Even Line Select (Line Select Start)

This bit works in conjunction with the 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 works for EDM[1:0] = 00. For all other EDM values, this 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 must 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 out of four 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 must be a multiple of four.

1: Only the data inside the window specified by the crop register is 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 first frame received.

In continuous grab mode, if the software clears this bit while a capture is ongoing, the bit is effectively cleared after the frame end.

*Note: The DMA controller and all DCMI configuration registers must be programmed correctly before enabling this bit.*

### 17.5.2 DCMI status register (DCMI\_SR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.														
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**: Vertical synchronization

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**: Horizontal synchronization

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.

### 17.5.3 DCMI raw interrupt status register (DCMI\_RIS)

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.

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.											
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

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 setting the LINE\_ISC bit of the DCMI\_ICR register.

**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 setting the VSYNC\_ISC bit of the DCMI\_ICR register.

**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 setting the ERR\_ISC bit of the DCMI\_ICR register.

*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.

The bit is cleared by setting the OVR\_ISC bit of the DCMI\_ICR register.

**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 (for example window cropped outside the frame).

The bit is cleared by setting the FRAME\_ISC bit of the DCMI\_ICR register.

#### 17.5.4 DCMI interrupt enable register (DCMI\_IER)

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.

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.											
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

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).

### 17.5.5 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 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.	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.

## 17.5.6 DCMI interrupt clear register (DCMI\_ICR)

The DCMI\_ICR register is write-only. Setting a bit of this register clears the corresponding flag in the DCMI\_RIS and DCMI\_MIS registers. Writing 0 has no effect.

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.											
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										

Bits 31:5 Reserved, must be kept at reset value.

**Bit 4 LINE\_ISC:** line interrupt status clear

Setting this bit clears the LINE\_RIS flag in the DCMI\_RIS register.

**Bit 3 VSYNC\_ISC:** Vertical Synchronization interrupt status clear

Setting this bit clears the VSYNC\_RIS flag in the DCMI\_RIS register.

**Bit 2 ERR\_ISC:** Synchronization error interrupt status clear

Setting this bit clears the ERR\_RIS flag in the DCMI\_RIS register.

*Note: This bit is available only in embedded synchronization mode.*

Bit 1 **OVR\_ISC**: Overrun interrupt status clear

Setting this bit clears the OVR\_RIS flag in the DCMI\_RIS register.

Bit 0 **FRAME\_ISC**: Capture complete interrupt status clear

Setting this bit clears the FRAME\_RIS flag in the DCMI\_RIS register.

### 17.5.7 DCMI embedded synchronization code register (DCMI\_ESCR)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FEC[7:0]								LEC[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
LSC[7:0]								FSC[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **FEC[7:0]**: 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[7:0]**: 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[7:0]**: 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[7:0]**: 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 first occurrence of LSC after an FEC code is interpreted as a start of frame delimiter.

### 17.5.8 DCMI embedded synchronization unmask register (DCMI\_ESUR)

Address offset: 0x1C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FEU[7:0]								LEU[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
LSU[7:0]								FSU[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **FEU[7:0]**: 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[7:0]**: 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[7:0]**: 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[7:0]**: 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.

### 17.5.9 DCMI crop window start (DCMI\_CWSTRT)

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.	VST[12:0]												
			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.	HOFFCNT[13:0]													
		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.

### 17.5.10 DCMI crop window size (DCMI\_CWSIZE)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	VLINE[13:0]													
		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.	CAPCNT[13:0]													
		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 **VLINE[13:0]**: 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. It value must corresponds to word-aligned data for different widths of parallel interfaces.

0x0000: 1 pixel

0x0001: 2 pixels

0x0002: 3 pixels

....

### 17.5.11 DCMI data register (DCMI\_DR)

Address offset: 0x28

Reset value: 0x0000 0000

The digital camera Interface packages all the received data in 32-bit format before requesting a DMA transfer. A 8-word deep FIFO is available to leave enough time for DMA transfers and avoid DMA overrun conditions.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BYTE3[7:0]								BYTE2[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
BYTE1[7:0]								BYTE0[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **BYTE3[7:0]**: Data byte 3

Bits 23:16 **BYTE2[7:0]**: Data byte 2

Bits 15:8 **BYTE1[7:0]**: Data byte 1

Bits 7:0 **BYTE0[7:0]**: Data byte 0

### 17.5.12 DCMI register map

**Table 115. DCMI register map and reset values**

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5		
0x00	DCMI_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																														
0x04	DCMI_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.		
	Reset value																														
0x08	DCMI_RIS	Res.	Res.	Res.	Res.	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	DCMI_IER	Res.	Res.	Res.	Res.	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	DCMI_MIS	Res.	Res.	Res.	Res.	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	DCMI_ICR	Res.	Res.	Res.	Res.	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	DCMI_ESCR	FEC[7:0]				LEC[7:0]				LSC[7:0]				FSC[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		
0x1C	DCMI_ESUR	FEU[7:0]				LEU[7:0]				LSU[7:0]				FSU[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		
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.	VLINE[13:0]												CAPCNT[13:0]															
	Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x28	DCMI_DR	BYTE3[7:0]				BYTE2[7:0]				BYTE1[7:0]				BYTE0[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		

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

## 18 LCD-TFT display controller (LTDC)

This section applies to the whole STM32F756xx, STM32F750xx and STM32F746xx devices, unless otherwise specified.

### 18.1 Introduction

The LCD-TFT (liquid crystal display - thin film transistor) display controller provides a parallel digital RGB (red, green, blue) and signals for horizontal, vertical synchronization, pixel clock and data enable as output to interface directly to a variety of LCD and TFT panels.

### 18.2 LTDC main features

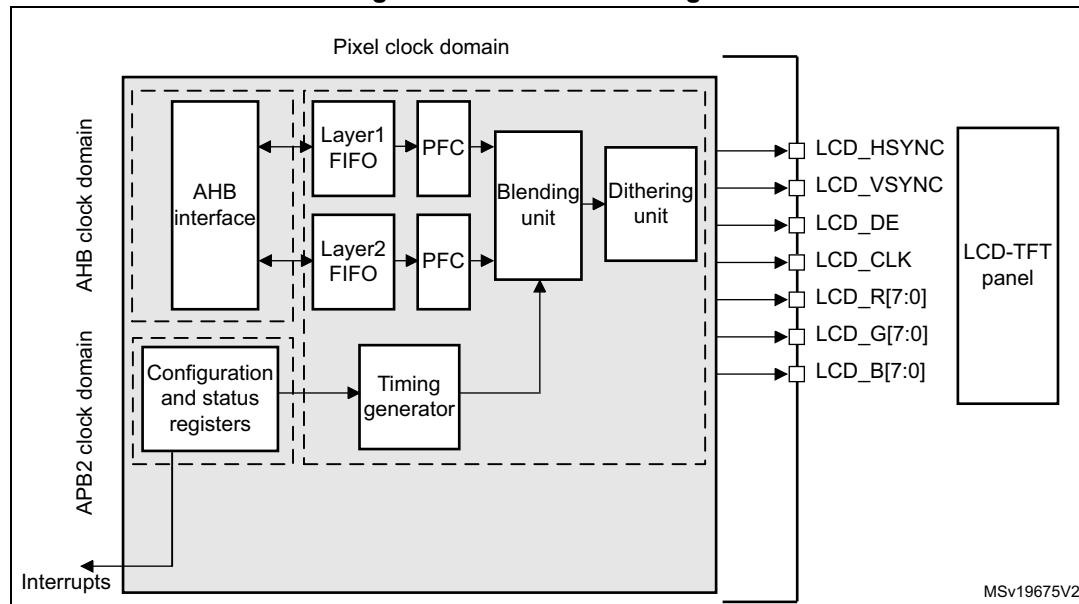
- 24-bit RGB parallel pixel output; 8 bits-per-pixel (RGB888)
- 2 display layers with dedicated FIFO (64x32-bit)
- Color look-up table (CLUT) up to 256 color (256x24-bit) per layer
- Programmable timings for different display panels
- Programmable background color
- Programmable polarity for HSYNC, VSYNC and data enable
- Up to 8 input color formats selectable per layer:
  - ARGB8888
  - RGB888
  - RGB565
  - ARGB1555
  - ARGB4444
  - L8 (8-bit luminance or CLUT)
  - AL44 (4-bit alpha + 4-bit luminance)
  - AL88 (8-bit alpha + 8-bit luminance)
- Pseudo-random dithering output for low bits per channel
  - Dither width 2 bits for red, green, blue
- Flexible blending between two layers using alpha value (per pixel or constant)
- Color keying (transparency color)
- Programmable window position and size
- Supports thin film transistor (TFT) color displays
- AHB master interface with burst of 16 words
- Up to 4 programmable interrupt events

## 18.3 LTDC functional description

### 18.3.1 LTDC block diagram

The block diagram of the LTDC is shown in the figure below.

**Figure 111. LTDC block diagram**



Layer FIFO: One FIFO 64x32-bit per layer.

PFC: pixel format converter, performing the pixel format conversion from the selected input pixel format of a layer to words.

AHB interface: for data transfer from memories to the FIFO.

Blending, dithering unit and timings generator: Refer to [Section 18.4.1](#) and [Section 18.4.2](#).

### 18.3.2 LTDC pins and external signal interface

The table below summarizes the LTDC signal interface.

**Table 116. LTDC pins and signal interface**

LCD-TFT signals	I/O	Description
LCD_CLK	O	Clock output
LCD_HSYNC	O	Horizontal synchronization
LCD_VSYNC	O	Vertical synchronization
LCD_DE	O	Not data enable
LCD_R[7:0]	O	Data: 8-bit red data
LCD_G[7:0]	O	Data: 8-bit green data
LCD_B[7:0]	O	Data: 8-bit blue data

The LTDC-TFT controller pins must be configured by the user application. The unused pins can be used for other purposes.

For LTDC outputs up to 24 bits (RGB888), if less than 8 bpp are used to output for example RGB565 or RGB666 to interface on 16- or 18-bit displays, the RGB display data lines must be connected to the MSB of the LCD-TFT controller RGB data lines. As an example, in the case of an LCD-TFT controller interfacing with a RGB565 16-bit display, the LCD display R[4:0], G[5:0] and B[4:0] data lines pins must be connected to LCD-TFT controller LCD\_R[7:3], LCD\_G[7:2] and LCD\_B[7:3].

### 18.3.3 LTDC reset and clocks

The LCD-TFT controller peripheral uses the following clock domains:

- AHB clock domain (HCLK)

This domain contains the LCD-TFT AHB master interface for data transfer from the memories to the Layer FIFO and the frame buffer configuration register

- APB2 clock domain (PCLK2):

This domain contains the global configuration registers and the interrupt register.

- Pixel clock domain (LCD\_CLK)

This domain contains the pixel data generation, the layer configuration register as well as the LCD-TFT interface signal generator. The LCD\_CLK output must be configured following the panel requirements. The LCD\_CLK is generated from a specific PLL output (refer to the reset and clock control section).

The table below summarizes the clock domain for each register.

**Table 117. Clock domain for each register**

LTDC register	Clock domain
LTDC_LxCR	HCLK
LTDC_LxCFBAR	
LTDC_LxCFBLR	
LTDC_LxCFBLNR	
LTDC_SRCR	PCLK2
LTDC_IER	
LTDC_ISR	
LTDC_ICR	

**Table 117. Clock domain for each register (continued)**

LTDC register	Clock domain
LTDC_SSCR	
LTDC_BPCR	
LTDC_AWCR	
LTDC_TWCR	
LTDC_GCR	
LTDC_BCCR	
LTDC_LIPCR	
LTDC_CPSR	
LTDC_CDSR	Pixel clock (LCD_CLK)
LTDC_LxWHPCR	
LTDC_LxWVPCR	
LTDC_LxCKCR	
LTDC_LxPFCR	
LTDC_LxCACR	
LTDC_LxDCCR	
LTDC_LxBFCR	
LTDC_LxCLUTWR	

Care must be taken while accessing the LTDC registers, the APB2 bus is stalled during:

- 6 PCKL2 periods + 5 LCD\_CLK periods (five HCLK periods for register on AHB clock domain) for register write access and update
- 7 PCKL2 periods + 5 LCD\_CLK periods (five HCLK periods for register on AHB clock domain) for register read access

For registers on PCLK2 clock domain, APB2 bus is stalled for six PCKL2 periods during the register write accesses, and for seven PCKL2 periods during read accesses.

The LCD controller can be reset by setting the corresponding bit in the RCC\_APB2RSTR register. It resets the three clock domains.

## 18.4 LTDC programmable parameters

The LCD-TFT controller provides flexible configurable parameters. It can be enabled or disabled through the LTDC\_GCR register.

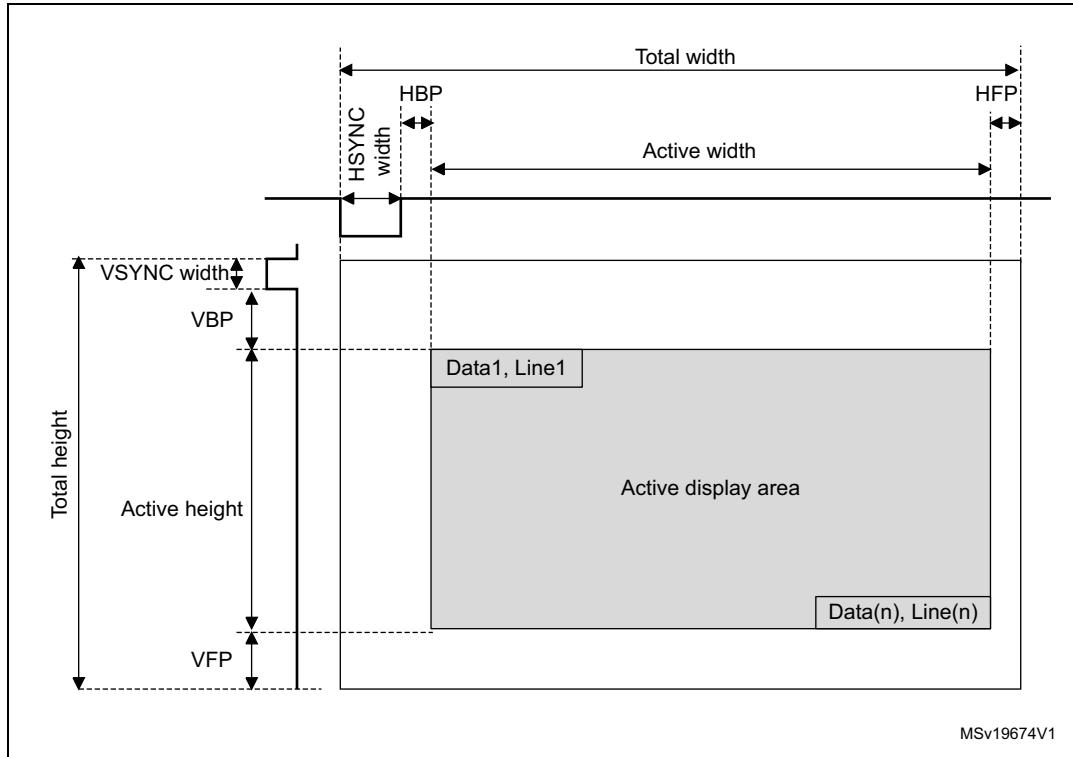
### 18.4.1 LTDC global configuration parameters

#### Synchronous timings

*Figure 112* presents the configurable timing parameters generated by the synchronous timings generator block presented in the block diagram *Figure 111*. It generates the

horizontal and vertical synchronization timings panel signals, the pixel clock and the data enable signals.

**Figure 112. LCD-TFT synchronous timings**



**Note:** The HBP and HFP are respectively the horizontal back porch and front porch period.

The VBP and the VFP are respectively the vertical back porch and front porch period.

The LCD-TFT programmable synchronous timings are the following:

- HSYNC and VSYNC width: horizontal and vertical synchronization width, configured by programming a value of HSYNC width - 1 and VSYNC width - 1 in the LTDC\_SSCR register
- HBP and VBP: horizontal and vertical synchronization back porch width, configured by programming the accumulated value HSYNC width + HBP - 1 and the accumulated value VSYNC width + VBP - 1 in the LTDC\_BPCR register.
- Active width and active height: the active width and active height are configured by programming the accumulated value HSYNC width + HBP + active width - 1 and the accumulated value VSYNC width + VBP + active height - 1 in the LTDC\_AWCR register.
- Total width: the total width is configured by programming the accumulated value HSYNC width + HBP + active width + HFP - 1 in the LTDC\_TWCR register. The HFP is the horizontal front porch period.
- Total height: the total height is configured by programming the accumulated value VSYNC height + VBP + active height + VFP - 1 in the LTDC\_TWCR register. The VFP is the vertical front porch period.

**Note:** When the LTDC is enabled, the timings generated start with X/Y=0/0 position as the first horizontal synchronization pixel in the vertical synchronization area and following the back

*porch, active data display area and the front porch.*

*When the LTDC is disabled, the timing generator block is reset to X = total width - 1, Y = total height - 1 and held the last pixel before the vertical synchronization phase and the FIFO are flushed. Therefore only blanking data is output continuously.*

### Example of synchronous timings configuration

LCD-TFT timings (must be extracted from panel datasheet):

- horizontal and vertical synchronization width: 0xA pixels and 0x2 lines
- horizontal and vertical back porch: 0x14 pixels and 0x2 lines
- active width and active height: 0x140 pixels, 0xF0 lines (320x240)
- horizontal front porch: 0xA pixels
- vertical front porch: 0x4 lines

The programmed values in the LTDC timings registers are:

- LTDC\_SSCR register to be programmed to 0x00090001  
(HSW[11:0] is 0x9 and VSH[10:0] is 0x1)
- LTDC\_BPCR register to be programmed to 0x001D0003  
(AHBP[11:0] is 0x1D (0xA+ 0x13) and AVBP[10:0]A is 0x3 (0x2 + 0x1))
- LTDC\_AWCR register to be programmed to 0x015D00F3  
(AAW[11:0] is 0x15D (0xA +0x14 +0x13F) and AAH[10:0] is 0xF3 (0x2 + 0x2 + 0xEF))
- LTDC\_TWCR register to be programmed to 0x000000167  
(TOTALW[11:0] is 0x167 (0xA +0x14 +0x140 + 0x9))
- LTDC\_THCR register to be programmed to 0x0000000F7  
(TOTALH[10:0]is 0xF7 (0x2 +0x2 + 0xF0 + 3))

### Programmable polarity

The horizontal and vertical synchronization, data enable and pixel clock output signals polarity can be programmed to active high or active low through the LTDC\_GCR register.

### Background color

A constant background color (RGB888) can programmed through the LTDC\_BCCR register. It is used for blending with the bottom layer.

### Dithering

The dithering pseudo-random technique using an LFSR is used to add a small random value (threshold) to each pixel color channel (R, G or B) value, thus rounding up the MSB in some cases when displaying a 24-bit data on 18-bit display. Thus the dithering technique is used to round data which is different from one frame to the other.

The dithering pseudo-random technique is the same as comparing LSBs against a threshold value and adding a 1 to the MSB part only, if the LSB part is  $\geq$  the threshold. The LSBs are typically dropped once dithering was applied.

The width of the added pseudo-random value is two bits for each color channel: two bits for red, two bits for green and two bits for blue.

Once the LCD-TFT controller is enabled, the LFSR starts running with the first active pixel and it is kept running even during blanking periods and when dithering is switched off. If the LTDC is disabled, the LFSR is reset.

The dithering can be switched on and off on the fly through the LTDC\_GCR register.

### Reload shadow registers

Some configuration registers are shadowed. The shadow registers values can be reloaded immediately to the active registers when writing to these registers or at the beginning of the vertical blanking period following the configuration in the LTDC\_SRCR register. If the immediate reload configuration is selected, the reload must be activated only when all new registers have been written.

The shadow registers must not be modified again before the reload is done. Reading from the shadow registers returns the actual active value. The new written value can only be read after the reload has taken place.

A register reload interrupt can be generated if enabled in the LTDC\_IER register.

The shadowed registers are all layer1 and layer2 registers except LTDC\_LxCLUTWR.

### Interrupt generation event

Refer to [Section 18.5: LTDC interrupts](#) for the interrupt configuration.

## 18.4.2 Layer programmable parameters

Up to two layers can be enabled, disabled and configured separately. The layer display order is fixed and it is bottom up. If two layers are enabled, the layer2 is the top displayed window.

### Windowing

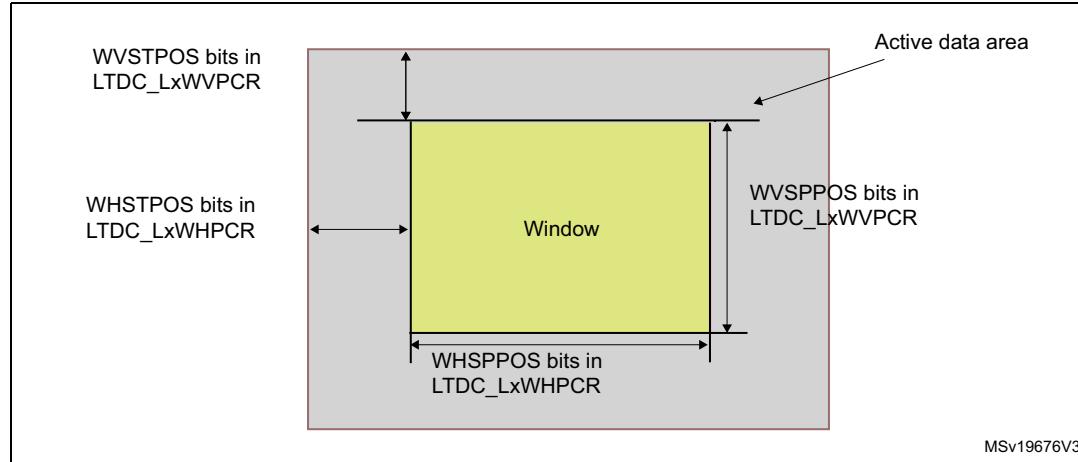
Every layer can be positioned and resized and it must be inside the active display area.

The window position and size are configured through the top-left and bottom-right X/Y positions and the internal timing generator that includes the synchronous, back porch size and the active data area. Refer to LTDC\_LxWHPCR and LTDC\_WVPCR registers.

The programmable layer position and size defines the first/last visible pixel of a line and the first/last visible line in the window. It allows to display either the full image frame or only a part of the image frame (see the figure below):

- The first and the last visible pixel in the layer are set by configuring the WHSTPOS[11:0] and WHSPPOS[11:0] in the LTDC\_LxWHPCR register.
- The first and the last visible lines in the layer are set by configuring the WVSTPOS[10:0] and WVSPPPOS[10:0] in the LTDC\_LxWVPCR register.

**Figure 113. Layer window programmable parameters**



### Pixel input format

The programmable pixel format is used for the data stored in the frame buffer of a layer.

Up to height input pixel formats can be configured for every layer through the LTDC\_LxPFCR register

The pixel data is read from the frame buffer and then transformed to the internal 8888 (ARGB) format as follows: components having a width of less than 8 bits get expanded to 8 bits by bit replication. The selected bit range is concatenated multiple times until it is longer than 8 bits. Of the resulting vector, the 8 MSB bits are chosen. Example: 5 bits of an RGB565 red channel become (bit positions) 43210432 (the three LSBs are filled with the three MSBs of the five bits)

The table below describes the pixel data mapping depending on the selected format.

**Table 118. Pixel data mapping versus color format**

ARGB8888			
@+3 Ax[7:0]	@+2 Rx[7:0]	@+1 Gx[7:0]	@ Bx[7:0]
@+7 Ax+1[7:0]	@+6 Rx+1[7:0]	@+5 Gx+1[7:0]	@+4 Bx+1[7:0]

**Table 118. Pixel data mapping versus color format (continued)**

<b>RGB888</b>			
@+3 B <sub>x+1</sub> [7:0]	@+2 R <sub>x</sub> [7:0]	@+1 G <sub>x</sub> [7:0]	@ B <sub>x</sub> [7:0]
@+7 G <sub>x+2</sub> [7:0]	@+6 B <sub>x+2</sub> [7:0]	@+5 R <sub>x+1</sub> [7:0]	@+4 G <sub>x+1</sub> [7:0]
<b>RGB565</b>			
@+3 R <sub>x+1</sub> [4:0] G <sub>x+1</sub> [5:3]	@+2 G <sub>x+1</sub> [2:0] B <sub>x+1</sub> [4:0]	@+1 R <sub>x</sub> [4:0] G <sub>x</sub> [5:3]	@ G <sub>x</sub> [2:0] B <sub>x</sub> [4:0]
@+7 R <sub>x+3</sub> [4:0] G <sub>x+3</sub> [5:3]	@+6 G <sub>x+3</sub> [2:0] B <sub>x+3</sub> [4:0]	@+5 R <sub>x+2</sub> [4:0] G <sub>x+2</sub> [5:3]	@+4 G <sub>x+2</sub> [2:0] B <sub>x+2</sub> [4:0]
<b>ARGB1555</b>			
@+3 A <sub>x+1</sub> [0] R <sub>x+1</sub> [4:0] G <sub>x+1</sub> [4:3]	@+2 G <sub>x+1</sub> [2:0] B <sub>x+1</sub> [4:0]	@+1 A <sub>x</sub> [0] R <sub>x</sub> [4:0] G <sub>x</sub> [4:3]	@ G <sub>x</sub> [2:0] B <sub>x</sub> [4:0]
@+7 A <sub>x+3</sub> [0] R <sub>x+3</sub> [4:0] G <sub>x+3</sub> [4:3]	@+6 G <sub>x+3</sub> [2:0] B <sub>x+3</sub> [4:0]	@+5 A <sub>x+2</sub> [0] R <sub>x+2</sub> [4:0] G <sub>x+2</sub> [4: 3]	@+4 G <sub>x+2</sub> [2:0] B <sub>x+2</sub> [4:0]
<b>ARGB4444</b>			
@+3 A <sub>x+1</sub> [3:0] R <sub>x+1</sub> [3:0]	@+2 G <sub>x+1</sub> [3:0] B <sub>x+1</sub> [3:0]	@+1 A <sub>x</sub> [3:0] R <sub>x</sub> [3:0]	@ G <sub>x</sub> [3:0] B <sub>x</sub> [3:0]
@+7 A <sub>x+3</sub> [3:0] R <sub>x+3</sub> [3:0]	@+6 G <sub>x+3</sub> [3:0] B <sub>x+3</sub> [3:0]	@+5 A <sub>x+2</sub> [3:0] R <sub>x+2</sub> [3:0]	@+4 G <sub>x+2</sub> [3:0] B <sub>x+2</sub> [3:0]
<b>L8</b>			
@+3 L <sub>x+3</sub> [7:0]	@+2 L <sub>x+2</sub> [7:0]	@+1 L <sub>x+1</sub> [7:0]	@ L <sub>x</sub> [7:0]
@+7 L <sub>x+7</sub> [7:0]	@+6 L <sub>x+6</sub> [7:0]	@+5 L <sub>x+5</sub> [7:0]	@+4 L <sub>x+4</sub> [7:0]
<b>AL44</b>			
@+3 A <sub>x+3</sub> [3:0] L <sub>x+3</sub> [3:0]	@+2 A <sub>x+2</sub> [3:0] L <sub>x+2</sub> [3:0]	@+1 A <sub>x+1</sub> [3:0] L <sub>x+1</sub> [3:0]	@ A <sub>x</sub> [3:0] L <sub>x</sub> [3:0]
@+7 A <sub>x+7</sub> [3:0] L <sub>x+7</sub> [3:0]	@+6 A <sub>x+6</sub> [3:0] L <sub>x+6</sub> [3:0]	@+5 A <sub>x+5</sub> [3:0] L <sub>x+5</sub> [3:0]	@+4 A <sub>x+4</sub> [3:0] L <sub>x+4</sub> [3:0]
<b>AL88</b>			
@+3 A <sub>x+1</sub> [7:0]	@+2 L <sub>x+1</sub> [7:0]	@+1 A <sub>x</sub> [7:0]	@ L <sub>x</sub> [7:0]
@+7 A <sub>x+3</sub> [7:0]	@+6 L <sub>x+3</sub> [7:0]	@+5 A <sub>x+2</sub> [7:0]	@+4 L <sub>x+2</sub> [7:0]

### Color look-up table (CLUT)

The CLUT can be enabled at run-time for every layer through the LTDC\_LxCR register and it is only useful in case of indexed color when using the L8, AL44 and AL88 input pixel format.

First, the CLUT must be loaded with the R, G and B values that replace the original R, G, B values of that pixel (indexed color). Each color (RGB value) has its own address that is the position within the CLUT.

The R, G and B values and their own respective address are programmed through the LTDC\_LxCLUTWR register:

- In case of L8 and AL88 input pixel format, the CLUT must be loaded by 256 colors. The address of each color is configured in the CLUTADD bits in the LTDC\_LxCLUTWR register.
- In case of AL44 input pixel format, the CLUT must be loaded by only 16 colors. The address of each color must be filled by replicating the 4-bit L channel to 8-bit as follows:
  - L0 (indexed color 0), at address 0x00
  - L1, at address 0x11
  - L2, at address 0x22
  - .....
  - L15, at address 0xFF

### Color frame buffer address

Every layer has a start address for the color frame buffer configured through the LTDC\_LxCFBAR register.

When a layer is enabled, the data is fetched from the color frame buffer.

### Color frame buffer length

Every layer has a total line length setting for the color frame buffer in bytes and a number of lines in the frame buffer configurable in the LTDC\_LxCFBLR and LTDC\_LxCFBLNR register respectively.

The line length and the number of lines settings are used to stop the prefetching of data to the layer FIFO at the end of the frame buffer:

- If it is set to less bytes than required, a FIFO underrun interrupt is generated if it has been previously enabled.
- If it is set to more bytes than actually required, the useless data read from the FIFO is discarded. The useless data is not displayed.

### Color frame buffer pitch

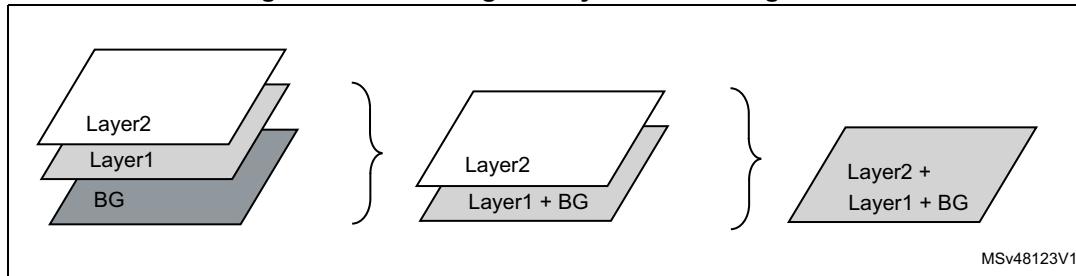
Every layer has a configurable pitch for the color frame buffer, that is the distance between the start of one line and the beginning of the next line in bytes. It is configured through the LTDC\_LxCFBLR register.

## Layer blending

The blending is always active and the two layers can be blended following the blending factors configured through the LTDC\_LxBFCR register.

The blending order is fixed and it is bottom up. If two layers are enabled, first the Layer1 is blended with the Background color, then the layer2 is blended with the result of blended color of layer1 and the background. Refer to the figure below.

**Figure 114. Blending two layers with background**



## Default color

Every layer can have a default color in the format ARGB which is used outside the defined layer window or when a layer is disabled.

The default color is configured through the LTDC\_LxDCCR register.

The blending is always performed between the two layers even when a layer is disabled. To avoid displaying the default color when a layer is disabled, keep the blending factors of this layer in the LTDC\_LxBFCR register to their reset value.

## Color keying

A color key (RGB) can be configured to be representative for a transparent pixel.

If the color keying is enabled, the current pixels (after format conversion and before CLUT respectively blending) are compared to the color key. If they match for the programmed RGB value, all channels (ARGB) of that pixel are set to 0.

The color key value can be configured and used at run-time to replace the pixel RGB value.

The color keying is enabled through the LTDC\_LxCKCR register.

The color keying is configured through the LTDC\_LxCKCR register. The programmed value depends on the pixel format as it is compared to current pixel after pixel format conversion to ARGB888.

Example: if the a mid-yellow color (50 % red + 50 % green) is used as the transparent color key:

- In RGB565, the mid-yellow color is 0x8400. Set the LTDC\_LxCKCR to 0x848200.
- In ARGB8888, the mid-yellow color is 0x808000. Set LTDC\_LxCKCR to 0x808000.
- In all CLUT-based color modes (L8, AL88, AL44), set one of the palette entry to the mid-yellow color 0x808000 and set the LTDC\_LxCKCR to 0x808000.

## 18.5 LTDC interrupts

The LTDC provides four maskable interrupts logically ORed to two interrupt vectors.

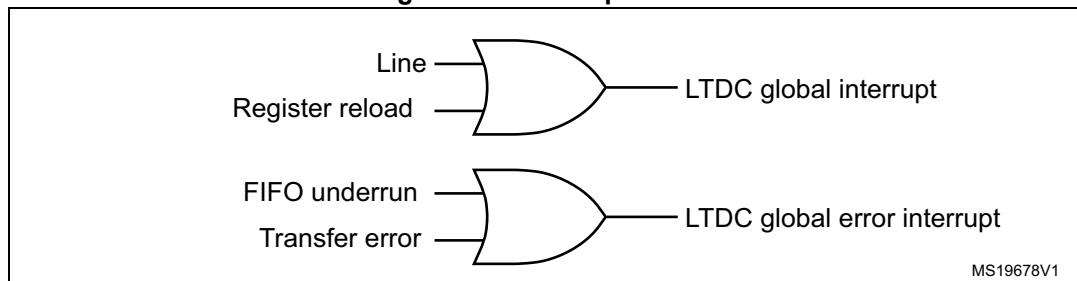
The interrupt sources can be enabled or disabled separately through the LTDC\_IER register. Setting the appropriate mask bit to 1 enables the corresponding interrupt.

The two interrupts are generated on the following events:

- Line interrupt: generated when a programmed line is reached. The line interrupt position is programmed in the LTDC\_LIPCR register
- Register reload interrupt: generated when the shadow registers reload is performed during the vertical blanking period
- FIFO underrun interrupt: generated when a pixel is requested from an empty layer FIFO
- Transfer error interrupt: generated when an AHB bus error occurs during data transfer

Those interrupts events are connected to the NVIC controller as described in the figure below.

**Figure 115. Interrupt events**



**Table 119. LTDC interrupt requests**

Interrupt event	Event flag	Enable control bit
Line	LIF	LIE
Register reload	RRIF	RRIEN
FIFO underrun	FUDERRIF	FUDERRIE
Transfer error	TERRIF	TERRIE

## 18.6 LTDC programming procedure

The steps listed below are needed to program the LTDC:

1. Enable the LTDC clock in the RCC register.
2. Configure the required pixel clock following the panel datasheet.
3. Configure the synchronous timings: VSYNC, HSYNC, vertical and horizontal back porch, active data area and the front porch timings following the panel datasheet as described in the [Section 18.4.1: LTDC global configuration parameters](#).
4. Configure the synchronous signals and clock polarity in the LTDC\_GCR register.
5. If needed, configure the background color in the LTDC\_BCCR register.
6. Configure the needed interrupts in the LTDC\_IER and LTDC\_LIPCR register.

7. Configure the layer1/2 parameters by:
  - programming the layer window horizontal and vertical position in the LTDC\_LxWHPCR and LTDC\_WVPCR registers. The layer window must be in the active data area.
  - programming the pixel input format in the LTDC\_LxPFCR register
  - programming the color frame buffer start address in the LTDC\_LxCFBAR register
  - programming the line length and pitch of the color frame buffer in the LTDC\_LxCFBLR register
  - programming the number of lines of the color frame buffer in the LTDC\_LxCFBLNR register
  - if needed, loading the CLUT with the RGB values and its address in the LTDC\_LxCLUTWR register
  - If needed, configuring the default color and the blending factors respectively in the LTDC\_LxDCCR and LTDC\_LxBFCR registers
8. Enable layer1/2 and if needed the CLUT in the LTDC\_LxCR register.
9. If needed, enable dithering and color keying respectively in the LTDC\_GCR and LTDC\_LxCKCR registers. They can be also enabled on the fly.
10. Reload the shadow registers to active register through the LTDC\_SRCR register.
11. Enable the LCD-TFT controller in the LTDC\_GCR register.
12. All layer parameters can be modified on the fly except the CLUT. The new configuration must be either reloaded immediately or during vertical blanking period by configuring the LTDC\_SRCR register.

**Note:** *All layer's registers are shadowed. Once a register is written, it must not be modified again before the reload has been done. Thus, a new write to the same register overrides the previous configuration if not yet reloaded.*

## 18.7 LTDC registers

### 18.7.1 LTDC synchronization size configuration register (LTDC\_SSCR)

Address offset: 0x08

Reset value: 0x0000 0000

This register defines the number of horizontal synchronization pixels minus 1 and the number of vertical synchronization lines minus 1. Refer to [Figure 112](#) and [Section 18.4](#) for an example of configuration.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
Res.	Res.	Res.	Res.	HSW[11:0]														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Res.	Res.	Res.	Res.	Res.	VSH[10:0]													
					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 **HSW[11:0]**: horizontal synchronization width (in units of pixel clock period)

These bits define the number of Horizontal Synchronization pixel minus 1.

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:0 **VSH[10:0]**: vertical synchronization height (in units of horizontal scan line)

These bits define the vertical Synchronization height minus 1. It represents the number of horizontal synchronization lines.

### 18.7.2 LTDC back porch configuration register (LTDC\_BPCR)

Address offset: 0x0C

Reset value: 0x0000 0000

This register defines the accumulated number of horizontal synchronization and back porch pixels minus 1 (HSYNC width + HBP - 1) and the accumulated number of vertical synchronization and back porch lines minus 1 (VSYNC height + VBP - 1).

Refer to [Figure 112](#) and [Section 18.4](#) for an example of configuration.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
Res.	Res.	Res.	Res.	AHBP[11:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Res.	Res.	Res.	Res.	Res.	AVBP[10:0]														
					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 27:16 **AHBP[11:0]**: accumulated horizontal back porch (in units of pixel clock period)

These bits define the accumulated horizontal back porch width that includes the horizontal synchronization and horizontal back porch pixels minus 1.

The horizontal back porch is the period between horizontal synchronization going inactive and the start of the active display part of the next scan line.

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:0 **AVBP[10:0]**: accumulated Vertical back porch (in units of horizontal scan line)

These bits define the accumulated vertical back porch width that includes the vertical synchronization and vertical back porch lines minus 1.

The vertical back porch is the number of horizontal scan lines at a start of frame to the start of the first active scan line of the next frame.

### 18.7.3 LTDC active width configuration register (LTDC\_AWCR)

Address offset: 0x10

Reset value: 0x0000 0000

This register defines the accumulated number of horizontal synchronization, back porch and active pixels minus 1 (HSYNC width + HBP + active width - 1) and the accumulated number of vertical synchronization, back porch lines and active lines minus 1 (VSYNC height + BVBP + active height - 1). Refer to [Figure 112](#) and [Section 18.4](#) for an example of configuration.

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

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

Bits 27:16 **AAW[11:0]**: accumulated active width (in units of pixel clock period)

These bits define the accumulated active width which includes the horizontal synchronization, horizontal back porch and active pixels minus 1.

The active width is the number of pixels in active display area of the panel scan line.

Refer to device datasheet for maximum active width supported following maximum pixel clock.

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:0 **AAH[10:0]**: accumulated active height (in units of horizontal scan line)

These bits define the accumulated height which includes the vertical synchronization, vertical back porch and the active height lines minus 1. The active height is the number of active lines in the panel.

Refer to device datasheet for maximum active height supported following maximum pixel clock.

### 18.7.4 LTDC total width configuration register (LTDC\_TWCR)

Address offset: 0x14

Reset value: 0x0000 0000

This register defines the accumulated number of horizontal synchronization, back porch, active and front porch pixels minus 1 (HSYNC width + HBP + active width + HFP - 1) and the accumulated number of vertical synchronization, back porch lines, active and front lines minus 1 (VSYNC height + BVBP + active height + VFP - 1). Refer to [Figure 112](#) and [Section 18.4](#) for an example of configuration.

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

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

Bits 27:16 **TOTALW[11:0]**: total width (in units of pixel clock period)

These bits defines the accumulated total width which includes the horizontal synchronization, horizontal back porch, active width and horizontal front porch pixels minus 1.

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:0 **TOTALH[10:0]**: total height (in units of horizontal scan line)

These bits defines the accumulated height which includes the vertical synchronization, vertical back porch, the active height and vertical front porch height lines minus 1.

### 18.7.5 LTDC global control register (LTDC\_GCR)

Address offset: 0x18

Reset value: 0x0000 2220

This register defines the global configuration of the LCD-TFT controller.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HSPOL	VSPOL	DEPOL	PCPOL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DEN
rw	rw	rw	rw												rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DRW[2:0]			Res.	DGW[2:0]			Res.	DBW[2:0]			Res.	Res.	Res.	LTDCE N
	r	r	r		r	r	r		r	r	r				rw

Bit 31 **HSPOL**: horizontal synchronization polarity

This bit is set and cleared by software.

0: horizontal synchronization polarity is active low.

1: horizontal synchronization polarity is active high.

Bit 30 **VSPOL**: vertical synchronization polarity

This bit is set and cleared by software.

0: vertical synchronization is active low.

1: vertical synchronization is active high.

Bit 29 **DEPOL**: not data enable polarity

This bit is set and cleared by software.

0: not data enable polarity is active low.

1: not data enable polarity is active high.

Bit 28 **PCPOL**: pixel clock polarity

This bit is set and cleared by software.

0: pixel clock polarity is active low.

1: pixel clock is active high.

Bits 27:17 Reserved, must be kept at reset value.

Bit 16 **DEN**: dither enable

This bit is set and cleared by software.

0: dither disable

1: dither enable

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **DRW[2:0]**: dither red width

These bits return the Dither Red Bits.

Bit 11 Reserved, must be kept at reset value.

Bits 10:8 **DGW[2:0]**: dither green width

These bits return the dither green bits.

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **DBW[2:0]**: dither blue width

These bits return the dither blue bits.

Bits 3:1 Reserved, must be kept at reset value.

Bit 0 **LTDCEN**: LCD-TFT controller enable

This bit is set and cleared by software.

0: LTDC disable

1: LTDC enable

### 18.7.6 LTDC shadow reload configuration register (LTDC\_SRCR)

Address offset: 0x24

Reset value: 0x0000 0000

This register allows to reload either immediately or during the vertical blanking period, the shadow registers values to the active registers. The shadow registers are all Layer1 and Layer2 registers except the LTDC\_L1CLUTWR and the LTDC\_L2CLUTWR.

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.	VBR	IMR													
														rw	rw

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

Bit 1 **VBR**: vertical blanking reload

This bit is set by software and cleared only by hardware after reload (it cannot be cleared through register write once it is set).

0: no effect

1: The shadow registers are reloaded during the vertical blanking period (at the beginning of the first line after the active display area).

Bit 0 **IMR**: immediate reload

This bit is set by software and cleared only by hardware after reload.

0: no effect

1: The shadow registers are reloaded immediately.

**Note:** The shadow registers read back the active values. Until the reload has been done, the 'old' value is read.

### 18.7.7 LTDC background color configuration register (LTDC\_BCCR)

Address offset: 0x2C

Reset value: 0x0000 0000

This register defines the background color (RGB888).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16						
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BCRED[7:0]													
								rw	rw	rw	rw	rw	rw	rw	rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
BCGREEN[7:0]								BCBLUE[7:0]													
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 23:16 **BCRED[7:0]**: background color red value

These bits configure the background red value.

Bits 15:8 **BCGREEN[7:0]**: background color green value

These bits configure the background green value.

Bits 7:0 **BCBLUE[7:0]**: background color blue value

These bits configure the background blue value.

### 18.7.8 LTDC interrupt enable register (LTDC\_IER)

This register determines which status flags generate an interrupt request by setting the corresponding bit to 1.

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.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RRIE	TERRIE	FUIE	LIE											
											rw	rw	rw	rw	

Bits 31:4 Reserved, must be kept at reset value.

Bit 3 **RRIE**: register reload interrupt enable

This bit is set and cleared by software.

0: register reload interrupt disable

1: register reload interrupt enable

Bit 2 **TERRIE**: transfer error interrupt enable

This bit is set and cleared by software.

0: transfer error interrupt disable

1: transfer error interrupt enable

Bit 1 **FUIE**: FIFO underrun interrupt enable

This bit is set and cleared by software.

0: FIFO underrun interrupt disable

1: FIFO underrun Interrupt enable

Bit 0 **LIE**: line interrupt enable

This bit is set and cleared by software.

0: line interrupt disable

1: line interrupt enable

### 18.7.9 LTDC interrupt status register (LTDC\_ISR)

Address offset: 0x38

Reset value: 0x0000 0000

This register returns the interrupt status flag.

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

Bits 31:4 Reserved, must be kept at reset value.

Bit 3 **RRIF**: register reload interrupt flag

0: no register reload interrupt generated

1: register reload interrupt generated when a vertical blanking reload occurs (and the first line after the active area is reached)

Bit 2 **TERRIF**: transfer error interrupt flag

0: no transfer error interrupt generated

1: transfer error interrupt generated when a bus error occurs

Bit 1 **FUIF**: FIFO underrun interrupt flag

0: no FIFO underrun interrupt generated.

1: FIFO underrun interrupt generated, if one of the layer FIFOs is empty and pixel data is read from the FIFO

Bit 0 **LIF**: line interrupt flag

0: no line interrupt generated

1: line interrupt generated when a programmed line is reached

### 18.7.10 LTDC Interrupt clear register (LTDC\_ICR)

Address offset: 0x3C

Reset value: 0x0000 0000

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

Bits 31:4 Reserved, must be kept at reset value.

Bit 3 **CRRIF**: clears register reload interrupt flag

0: no effect

1: clears the RRIF flag in the LTDC\_ISR register

Bit 2 **CTERRIF**: clears the transfer error interrupt flag

0: no effect

1: clears the TERRIF flag in the LTDC\_ISR register.

Bit 1 **CFUIF**: clears the FIFO underrun interrupt flag

0: no effect

1: clears the FUDERRIF flag in the LTDC\_ISR register.

Bit 0 **CLIF**: clears the line interrupt flag

0: no effect

1: clears the LIF flag in the LTDC\_ISR register.

### 18.7.11 LTDC line interrupt position configuration register (LTDC\_LIPCR)

Address offset: 0x40

Reset value: 0x0000 0000

This register defines the position of the line interrupt. The line value to be programmed depends on the timings parameters. Refer to [Figure 112](#).

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.	LIPOS[10:0]										
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:11 Reserved, must be kept at reset value.

Bits 10:0 **LIPOS[10:0]**: line interrupt position

These bits configure the line interrupt position.

### 18.7.12 LTDC current position status register (LTDC\_CPSR)

Address offset: 0x44

Reset value: 0x0000 0000

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

Bits 31:16 **CXPOS[15:0]**: current X position

These bits return the current X position.

Bits 15:0 **CYPOS[15:0]**: current Y position

These bits return the current Y position.

### 18.7.13 LTDC current display status register (LTDC\_CDSR)

Address offset: 0x48

Reset value: 0x0000 000F

This register returns the status of the current display phase which is controlled by the HSYNC, VSYNC, and horizontal/vertical DE signals.

Example: if the current display phase is the vertical synchronization, the VSYNCS bit is set (active high). If the current display phase is the horizontal synchronization, the HSYNCS bit is active high.

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

Bits 31:4 Reserved, must be kept at reset value.

Bit 3 **HSYNCS**: horizontal synchronization display status

0: active low

1: active high

Bit 2 **VSYNCS**: vertical synchronization display status

0: active low

1: active high

Bit 1 **HDES**: horizontal data enable display status

0: active low

1: active high

Bit 0 **VDES**: vertical data enable display status

- 0: active low
- 1: active high

**Note:** *The returned status does not depend on the configured polarity in the LTDC\_GCR register, instead it returns the current active display phase.*

### 18.7.14 LTDC layer x control register (LTDC\_LxCR)

Address offset:  $0x84 + 0x80 * (x - 1)$ , ( $x = 1$  to  $2$ )

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.	CLUTEN	Res.	Res.	COLKEN	LEN										
											rw			rw	rw

Bits 31:5 Reserved, must be kept at reset value.

Bit 4 **CLUTEN**: color look-up table enable

This bit is set and cleared by software.

- 0: color look-up table disable
- 1: color look-up table enable

The CLUT is only meaningful for L8, AL44 and AL88 pixel format. Refer to [Color look-up table \(CLUT\)](#)

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

Bit 1 **COLKEN**: color keying enable

This bit is set and cleared by software.

- 0: color keying disable
- 1: color keying enable

Bit 0 **LEN**: layer enable

This bit is set and cleared by software.

- 0: layer disable
- 1: layer enable

### 18.7.15 LTDC layer x window horizontal position configuration register (LTDC\_LxWHPCR)

Address offset:  $0x88 + 0x80 * (x - 1)$ , ( $x = 1$  to  $2$ )

Reset value: 0x0000 0000

This register defines the horizontal position (first and last pixel) of the layer 1 or 2 window.

The first visible pixel of a line is the programmed value of AHBP[11:0] bits + 1 in the LTDC\_BPCR register.

The last visible pixel of a line is the programmed value of AAW[11:0] bits in the LTDC\_AWCR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
Res.	Res.	Res.	Res.	WHSPPPOS[11:0]													
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Res.	Res.	Res.	Res.	WHSTPOS[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 **WHSPPPOS[11:0]**: window horizontal stop position

These bits configure the last visible pixel of a line of the layer window.

WHSPPPOS[11:0] must be  $\geq \text{AHBP}[11:0] \text{ bits} + 1$  (programmed in LTDC\_BPCR register).

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

Bits 11:0 **WHSTPOS[11:0]**: window horizontal start position

These bits configure the first visible pixel of a line of the layer window.

WHSTPOS[11:0] must be  $\leq \text{AAW}[11:0] \text{ bits}$  (programmed in LTDC\_AWCR register).

#### Example:

The LTDC\_BPCR register is configured to 0x000E0005 (AHBP[11:0] is 0xE) and the LTDC\_AWCR register is configured to 0x028E01E5 (AAW[11:0] is 0x28E). To configure the horizontal position of a window size of 630x460, with horizontal start offset of 5 pixels in the active data area:

1. layer window first pixel, WHSTPOS[11:0], must be programmed to 0x14 (0xE+1+0x5)
2. layer window last pixel, WHSPPPOS[11:0], must be programmed to 0x28A.

### 18.7.16 LTDC layer x window vertical position configuration register (LTDC\_LxWVPCR)

Address offset: 0x8C + 0x80 \* (x - 1), (x = 1 to 2)

Reset value: 0x0000 0000

This register defines the vertical position (first and last line) of the layer1 or 2 window.

The first visible line of a frame is the programmed value of AVBP[10:0] bits + 1 in the register LTDC\_BPCR register.

The last visible line of a frame is the programmed value of AAH[10:0] bits in the LTDC\_AWCR register.

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

Bits 31:27 Reserved, must be kept at reset value.

Bits 26:16 **WVSPPOS[10:0]**: window vertical stop position

These bits configure the last visible line of the layer window.

WVSPPOS[10:0] must be  $\geq \text{AVBP}[10:0] \text{ bits} + 1$  (programmed in LTDC\_BPCR register).

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:0 **WVSTPOS[10:0]**: window vertical start position

These bits configure the first visible line of the layer window.

WVSTPOS[10:0] must be  $\leq \text{AAH}[10:0] \text{ bits}$  (programmed in LTDC\_AWCR register).

#### Example:

The LTDC\_BPCR register is configured to 0x000E0005 (AVBP[10:0] is 0x5) and the LTDC\_AWCR register is configured to 0x028E01E5 (AAH[10:0] is 0x1E5).

To configure the vertical position of a window size of 630x460, with vertical start offset of 8 lines in the active data area:

1. layer window first line: WVSTPOS[10:0] must be programmed to 0xE (0x5 + 1 + 0x8).
2. layer window last line: WVSTPOS[10:0] must be programmed to 0x1DA.

### 18.7.17 LTDC layer x color keying configuration register (LTDC\_LxCKCR)

Address offset: 0x90 + 0x80 \* (x - 1), (x = 1 to 2)

Reset value: 0x0000 0000

This register defines the color key value (RGB), that is used by the color keying.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16						
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CKRED[7:0]													
								rw	rw	rw	rw	rw	rw	rw	rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
CKGREEN[7:0]								CKBLUE[7:0]													
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 23:16 **CKRED[7:0]**: color key red value

Bits 15:8 **CKGREEN[7:0]**: color key green value

Bits 7:0 **CKBLUE[7:0]**: color key blue value

### 18.7.18 LTDC layer x pixel format configuration register (LTDC\_LxPFCR)

Address offset: 0x94 + 0x80 \* (x - 1), (x = 1 to 2)

Reset value: 0x0000 0000

This register defines the pixel format that is used for the stored data in the frame buffer of a layer. The pixel data is read from the frame buffer and then transformed to the internal format 8888 (ARGB).

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.	PF[2:0]															
														rw	rw	rw

Bits 31:3 Reserved, must be kept at reset value.

Bits 2:0 **PF[2:0]**: pixel format

These bits configure the pixel format

000: ARGB8888

001: RGB888

010: RGB565

011: ARGB1555

100: ARGB4444

101: L8 (8-bit luminance)

110: AL44 (4-bit alpha, 4-bit luminance)

111: AL88 (8-bit alpha, 8-bit luminance)

### 18.7.19 LTDC layer x constant alpha configuration register (LTDC\_LxCACR)

Address offset:  $0x98 + 0x80 * (x - 1)$ , ( $x = 1$  to  $2$ )

Reset value:  $0x0000\ 00FF$

This register defines the constant alpha value (divided by 255 by hardware), that is used in the alpha blending. Refer to LTDC\_LxBFCR register.

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.	rw	rw	rw	rw	rw	rw	rw								
									CONSTA[7:0]						

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

Bits 7:0 **CONSTA[7:0]**: constant alpha

These bits configure the constant alpha used for blending. The constant alpha is divided by 255 by hardware.

Example: if the programmed constant alpha is  $0xFF$ , the constant alpha value is  $255 / 255 = 1$ .

### 18.7.20 LTDC layer x default color configuration register (LTDC\_LxDCCR)

Address offset:  $0x9C + 0x80 * (x - 1)$ , ( $x = 1$  to  $2$ )

Reset value:  $0x0000\ 0000$

This register defines the default color of a layer in the format ARGB. The default color is used outside the defined layer window or when a layer is disabled. The reset value of  $0x00000000$  defines a transparent black color.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DCALPHA[7:0]								DCRED[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
DCGREEN[7:0]								DCBLUE[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DCALPHA[7:0]**: default color alpha

These bits configure the default alpha value.

Bits 23:16 **DCRED[7:0]**: default color red

These bits configure the default red value.

Bits 15:8 **DCGREEN[7:0]**: default color green

These bits configure the default green value.

Bits 7:0 **DCBLUE[7:0]**: default color blue

These bits configure the default blue value.

### 18.7.21 LTDC layer x blending factors configuration register (LTDC\_LxBFCR)

Address offset: 0xA0 + 0x80 \* (x - 1), (x = 1 to 2)

Reset value: 0x0000 0607

This register defines the blending factors F1 and F2.

The general blending formula is: BC = BF1 x C + BF2 x Cs

- BC = blended color
- BF1 = blend factor 1
- C = current layer color
- BF2 = blend factor 2
- Cs = subjacent layers blended color

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.	BF1[2:0]			Res.	Res.	Res.	Res.	BF2[2:0]			
					rw	rw	rw					rw	rw	rw	

Bits 31:11 Reserved, must be kept at reset value.

Bits 10:8 **BF1[2:0]**: blending factor 1

These bits select the blending factor F1.

100: constant alpha

110: pixel alpha x constant alpha

Others: reserved

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

Bits 2:0 **BF2[2:0]**: blending factor 2

These bits select the blending factor F2

101: 1 - constant alpha

111: 1 - (pixel alpha x constant alpha)

Others: reserved

Note: *The constant alpha value, is the programmed value in the LxCACR register divided by 255 by hardware.*

Example: Only layer1 is enabled, BF1 configured to constant alpha. BF2 configured to 1 - constant alpha. The constant alpha programmed in the LxCACR register is 240 (0xF0). Thus, the constant alpha value is 240/255 = 0.94. C: current layer color is 128.

Cs: background color is 48. Layer1 is blended with the background color.

$BC = \text{constant alpha} \times C + (1 - \text{Constant Alpha}) \times Cs = 0.94 \times 128 + (1 - 0.94) \times 48 = 123$ .

### 18.7.22 LTDC layer x color frame buffer address register (LTDC\_LxCFBAR)

Address offset: 0xAC + 0x80 \* (x - 1), (x = 1 to 2)

Reset value: 0x0000 0000

This register defines the color frame buffer start address which has to point to the address where the pixel data of the top left pixel of a layer is stored in the frame buffer.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CFBADD[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
CFBADD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **CFBADD[31:0]**: color frame buffer start address

These bits define the color frame buffer start address.

### 18.7.23 LTDC layer x color frame buffer length register (LTDC\_LxCFBLR)

Address offset: 0xB0 + 0x80 \* (x - 1), (x = 1 to 2)

Reset value: 0x0000 0000

This register defines the color frame buffer line length and pitch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.		CFBP[12:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.		CFBLL[12:0]											
				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 **CFBP[12:0]**: color frame buffer pitch in bytes

These bits define the pitch that is the increment from the start of one line of pixels to the start of the next line in bytes.

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:0 **CFBLL[12:0]**: color frame buffer line length

These bits define the length of one line of pixels in bytes + 3.

The line length is computed as follows:

active high width \* number of bytes per pixel + 3.

#### Example:

- A frame buffer having the format RGB565 (2 bytes per pixel) and a width of 256 pixels (total number of bytes per line is  $256 * 2 = 512$ ), where pitch = line length requires a value of 0x02000203 to be written into this register.

- A frame buffer having the format RGB888 (3 bytes per pixel) and a width of 320 pixels (total number of bytes per line is  $320 * 3 = 960$ ), where pitch = line length requires a value of 0x03C003C3 to be written into this register.

### 18.7.24 LTDC layer x color frame buffer line number register (LTDC\_LxCFBLNR)

Address offset: 0xB4 + 0x80 \* (x - 1), (x = 1 to 2)

Reset value: 0x0000 0000

This register defines the number of lines in the color frame buffer.

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.	CFBLNBR[10:0]											
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:11 Reserved, must be kept at reset value.

Bits 10:0 **CFBLNBR[10:0]**: frame buffer line number

These bits define the number of lines in the frame buffer that corresponds to the active high width.

**Note:** The number of lines and line length settings define how much data is fetched per frame for every layer. If it is configured to less bytes than required, a FIFO underrun interrupt is generated if enabled.

The start address and pitch settings on the other hand define the correct start of every line in memory.

### 18.7.25 LTDC layer x CLUT write register (LTDC\_LxCLUTWR)

Address offset: 0xC4 + 0x80 \* (x - 1), (x = 1 to 2)

Reset value: 0x0000 0000

This register defines the CLUT address and the RGB value.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLUTADD[7:0]								RED[7:0]							
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
GREEN[7:0]								BLUE[7:0]							
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:24 **CLUTADD[7:0]**: CLUT address

These bits configure the CLUT address (color position within the CLUT) of each RGB value.

Bits 23:16 **RED[7:0]**: red value

These bits configure the red value.

Bits 15:8 **GREEN[7:0]**: green value

These bits configure the green value.

Bits 7:0 **BLUE[7:0]**: blue value

These bits configure the blue value.

**Note:** The CLUT write register must be configured only during blanking period or if the layer is disabled. The CLUT can be enabled or disabled in the LTDC\_LxCR register.

The CLUT is only meaningful for L8, AL44 and AL88 pixel format.

## 18.7.26 LTDC register map

**Table 120. LTDC register map and reset values**

Table 120. LTDC register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x04C-0x080	Reserved																																	
0x084	LTDC_L1CR	Res	CLUTEN	Res	LEN																													
	Reset value																											0	0	0	0			
0x088	LTDC_L1WHPCR	Res	WHSTPOS[11:0]	WHSPPOS[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
0x08C	LTDC_L1WVPCR	Res	WVSTPOS[10:0]	WVSPPOS[10:0]																														
	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
0x090	LTDC_L1CKCR	Res	CKGREEN[7:0]	CKRED[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
0x094	LTDC_L1PFCR	Res	PF[2:0]																															
	Reset value																											0	0	0	0			
0x098	LTDC_L1CACR	Res	CONSTA[7:0]																															
	Reset value																											1	1	1	1			
0x09C	LTDC_L1DCCR																														DCBLUE[7:0]	DCGREEN[7:0]		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0A0	LTDC_L1BFCR	Res	BF1[2:0]																															
	Reset value																														BF2[2:0]	1	1	
0x0A4-0x0A8	Reserved																																	
0x0AC	LTDC_L1CFBAR																														CFBADD[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	
0x0B0	LTDC_L1CFBLR	Res	CFBLL[12:0]	CFBP[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
0x0B4	LTDC_L1CFBLNR	Res	CFBLNBR[10:0]																															
	Reset value																														0	0	0	
0x0B8-0x0C0	Reserved																																	
0x0C4	LTDC_L1CLUTWR																														BLUE[7:0]	RED[7:0]	CLUTADD[7:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0C8-0x100	Reserved																																	
0x104	LTDC_L2CR	Res	CLUTEN	Res	LEN																													
	Reset value																												0	0	0	0		
0x108	LTDC_L2WHPCR	Res	WHSTPOS[11:0]	WHSPPOS[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
0x10C	LTDC_L2WVPCR	Res	WVSTPOS[10:0]	WVSPPOS[10:0]																														
	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
0x110	LTDC_L2CKCR	Res	CKGREEN[7:0]	CKRED[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
0x114	LTDC_L2PFCR	Res	PF[2:0]																															
	Reset value																											0	0	0	0			
0x118	LTDC_L2CACR	Res	CONSTA[7:0]																															
	Reset value																											1	1	1	1			

**Table 120. LTDC register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x11C	LTDC_L2DCCR	DCALPHA[7:0]					DCRED[7:0]					DCGREEN[7:0]					DCBLUE[7:0]																
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x120	LTDC_L2BFCR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	BF1[2:0]	Res	Res	Res	Res	Res	BF2[2:0]			
																								1	1	0				1	1	1	
0x124-0x128	Reserved	Reserved																															
0x12C	LTDC_L2CFBAR	CFBADD[31:0]																															
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x130	LTDC_L2CFBLR	Res	Res	Res	CFBP[12:0]					CFBLL[12:0]					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	0	0	0	0	0			
0x134	LTDC_L2CFBLNR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	CFBLNBR[10:0]								
																								0	0	0	0	0	0	0	0	0	
0x144	LTDC_L2CLUTWR	CLUTADD[7:0]					RED[7:0]					GREEN[7:0]					BLUE[7:0]																
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

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

## 19 True random number generator (RNG)

### 19.1 Introduction

The RNG is a true random number generator that continuously provides 32-bit entropy samples, based on an analog noise source. It can be used by the application as a live entropy source to build a NIST compliant deterministic random bit generator (DRBG).

The RNG true random number generator has been tested using NIST statistical test suite SP800-22 rev1a (April 2010).

### 19.2 RNG main features

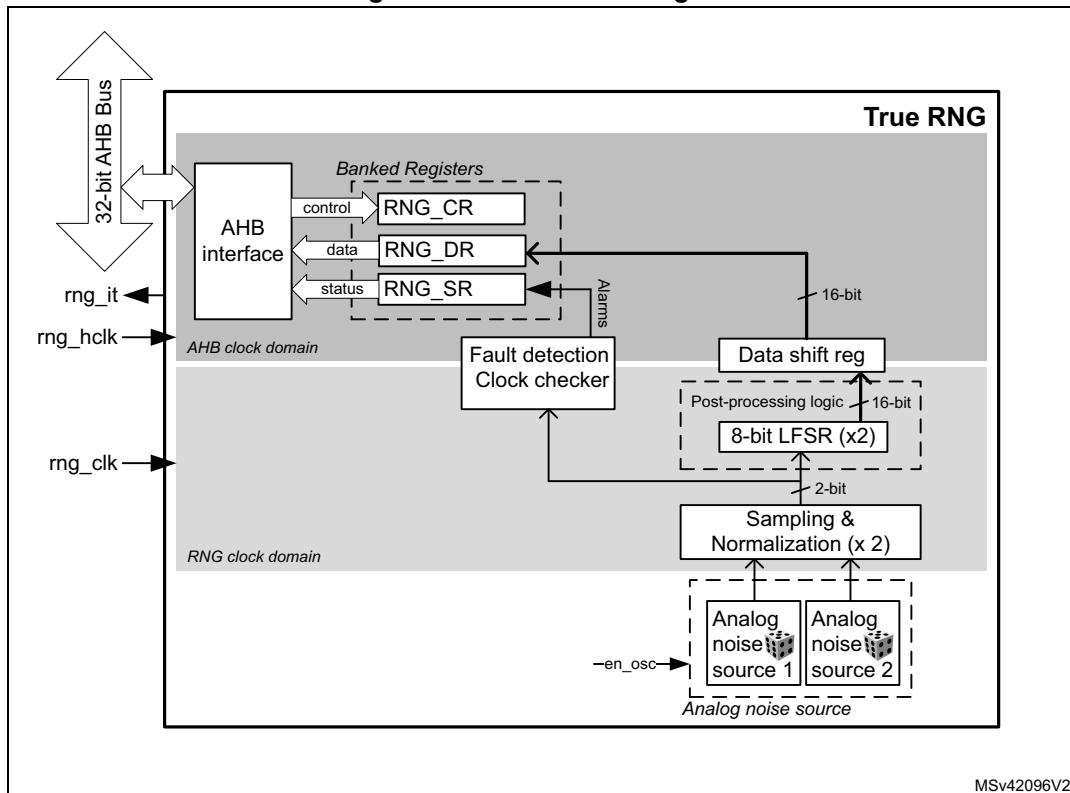
- The RNG delivers 32-bit true random numbers, produced by an analog entropy source post-processed with linear-feedback shift registers (LFSR).
- In the NIST configuration, it produces one 32-bit random samples every 42 RNG clock cycles(dedicated clock).
- It allows embedded continuous basic health tests with associated error management
  - Includes too low sampling clock detection and repetition count tests.
- It can be disabled to reduce power consumption.
- It has an AMBA AHB slave peripheral, accessible through 32-bit word single accesses only (else an AHB bus error is generated). Warning! any write not equal to 32 bits might corrupt the register content.

## 19.3 RNG functional description

### 19.3.1 RNG block diagram

*Figure 116* shows the RNG block diagram.

**Figure 116. RNG block diagram**



MSv42096V2

### 19.3.2 RNG internal signals

*Table 121* describes a list of useful-to-know internal signals available at the RNG level, not at the STM32 product level (on pads).

**Table 121. RNG internal input/output signals**

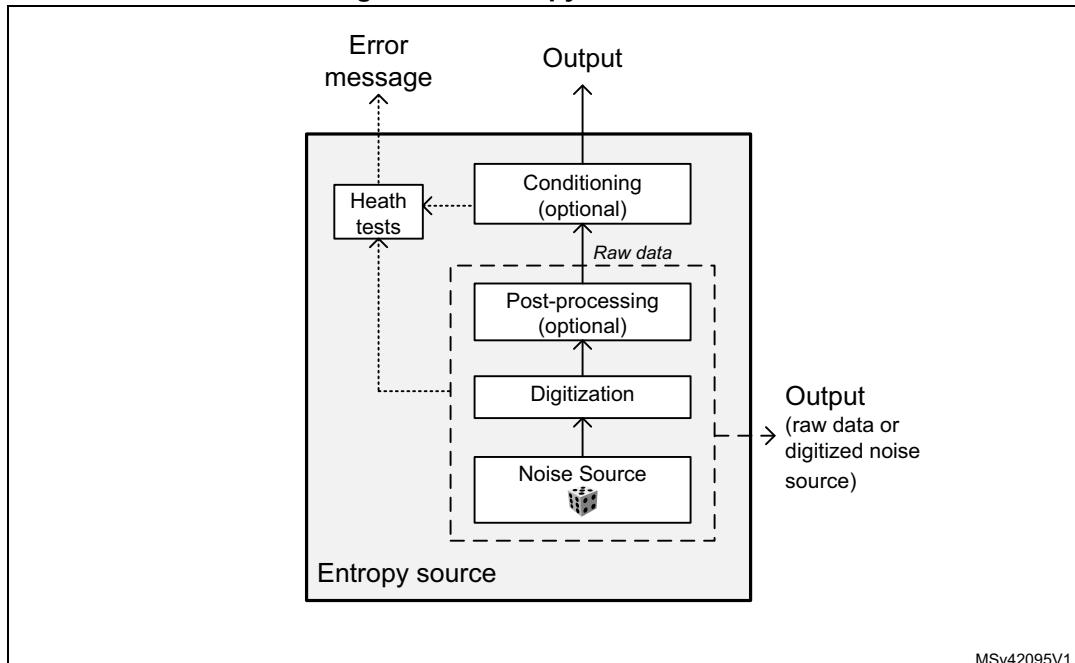
Signal name	Signal type	Description
rng_it	Digital output	RNG global interrupt request
rng_hclk	Digital input	AHB clock
rng_clk	Digital input	RNG dedicated clock, asynchronous to rng_hclk

### 19.3.3 Random number generation

The true random number generator (RNG) delivers truly random data through its AHB interface at deterministic intervals. Within its boundary the RNG implements the entropy source model pictured on [Figure 117](#), and provides three main functions to the application:

- Collects the bitstring output of the entropy source box
- Obtains samples of the noise source for validation purpose
- Collects error messages from continuous health tests

**Figure 117. Entropy source model**



MSv42095V1

The main components of the RNG are:

- A source of physical randomness (analog noise source)
- A digitization stage for this analog noise source
- A stage delivering post-processed noise source (raw data)
- An output buffer for the raw data. If further cryptographic conditioning is required by the application it needs to be performed by software.
- An optional output for the digitized noise source (unbuffered, on digital pads)
- Basic health tests on the digitized noise source

The components pictured above are detailed hereafter.

## Noise source

The noise source is the component that contains the non-deterministic, entropy-providing activity that is ultimately responsible for the uncertainty associated with the bitstring output by the entropy source. It is composed of:

- Two analog noise sources, each based on three XORed free-running ring oscillator outputs. It is possible to disable those analog oscillators to save power, as described in [Section 19.3.8](#).
- A sampling stage of these outputs clocked by a dedicated clock input (rng\_clk), delivering a 2-bit raw data output.

This noise source sampling is independent to the AHB interface clock frequency (rng\_hclk).

*Note:*

*In [Section 19.6](#) the recommended RNG clock frequencies are given.*

## Post processing

The sample values obtained from a true random noise source consist of 2-bit bitstrings. Because this noise source output is biased, the RNG implements a post-processing component that reduces that bias to a tolerable level.

The RNG post-processing consists of two stages, applied to each noise source bits:

- The RNG takes half of the bits from the sampled noise source, and half of the bits from inverted sampled noise source. Thus, if the source generates more '1' than '0' (or the opposite), it is filtered
- A linear feedback shift register (LFSR) performs a whitening process, producing 8-bit strings.

This component is clocked by the RNG clock.

The times required between two random number generations, and between the RNG initialization and availability of first sample are described in [Section 19.5](#).

## Output buffer

The RNG\_DR data output register can store up to two 16-bit words, which have been output from the post-processing component (LFSR). In order to read back 32-bit random samples, it is required to wait 42 RNG clock cycles.

Whenever a random number is available through the RNG\_DR register, the DRDY flag changes from 0 to 1. This flag remains high until the output buffer becomes empty after reading one word from the RNG\_DR register.

*Note:*

*When interrupts are enabled an interrupt is generated when this data ready flag transitions from 0 to 1. Interrupt is then cleared automatically by the RNG as explained above.*

## Health checks

This component ensures that the entire entropy source (with its noise source) starts then operates as expected, obtaining assurance that failures are caught quickly and with a high probability and reliability.

The RNG implements the following health check features.

1. Continuous health tests, running indefinitely on the output of the noise source
  - Repetition count test, flagging an error when:
    - a) One of the noise source has provided more than 64 consecutive bits at a constant value ("0" or "1")
    - b) One of the noise sources has delivered more than 32 consecutive occurrence of two bits patterns ("01" or "10")
  - 2. Vendor specific continuous test
    - Real-time "too slow" sampling clock detector, flagging an error when one RNG clock cycle is smaller than AHB clock cycle divided by 16.

The CECS and SECS status bits in the RNG\_SR register indicate when an error condition is detected, as detailed in [Section 19.3.7](#).

*Note:* An interrupt can be generated when an error is detected.

#### 19.3.4 RNG initialization

When a hardware reset occurs the following chain of events occurs:

1. The analog noise source is enabled, and the logic starts sampling the analog output after four RNG clock cycles, filling LFSR shift register and associated 16-bit post-processing shift register.
2. The output buffer is refilled automatically according to the RNG usage.

The associated initialization time can be found in [Section 19.5](#).

#### 19.3.5 RNG operation

##### Normal operation

To run the RNG using interrupts, the following steps are recommended:

1. Enable the interrupts by setting the IE bit in the RNG\_CR register. At the same time, enable the RNG by setting the bit RNGEN=1.
2. An interrupt is now generated when a random number is ready or when an error occurs. Therefore, at each interrupt, check that:
  - No error occurred. The SEIS and CEIS bits must be set to 0 in the RNG\_SR register.
  - A random number is ready. The DRDY bit must be set to 1 in the RNG\_SR register.
  - If the above two conditions are true the content of the RNG\_DR register can be read.

To run the RNG in polling mode following steps are recommended:

1. Enable the random number generation by setting the RNGEN bit to "1" in the RNG\_CR register.
2. Read the RNG\_SR register and check that:
  - No error occurred (the SEIS and CEIS bits must be set to 0)
  - A random number is ready (the DRDY bit must be set to 1)
3. If above conditions are true read the content of the RNG\_DR register.

**Note:** When data is not ready ( $DRDY = 0$ )  $RNG\_DR$  returns zero.  
It is recommended to always verify that  $RNG\_DR$  is different from zero. Because when it is the case a seed error occurred between  $RNG\_SR$  polling and  $RND\_DR$  output reading (rare event).

### Low-power operation

If the power consumption is a concern to the application, low-power strategies can be used, as described in [Section 19.3.8](#).

### Software post-processing

If a NIST approved DRBG with 128 bits of security strength is required an approved random generator software must be built around the RNG true random number generator.

## 19.3.6 RNG clocking

The RNG runs on two different clocks: the AHB bus clock and a dedicated RNG clock.

The AHB clock is used to clock the AHB banked registers and the post-processing component. The RNG clock is used for noise source sampling. Recommended clock configurations are detailed in [Section 19.6: RNG entropy source validation](#).

**Note:** When the CED bit in the  $RNG\_CR$  register is set to 0, the RNG clock frequency **must be higher than the AHB clock frequency divided by 16**, otherwise the clock checker always flags a clock error (CECS = 1 in the  $RNG\_SR$  register).

See [Section 19.3.1: RNG block diagram](#) for details (AHB and RNG clock domains).

## 19.3.7 Error management

In parallel to random number generation a health check block verifies the correct noise source behavior and the frequency of the RNG source clock as detailed in this section. Associated error state is also described.

### Clock error detection

When the clock error detection is enabled (CED = 0) and if the RNG clock frequency is too low, the RNG sets to 1 both the CEIS and CECS bits to indicate that a clock error occurred. In this case, the application must check that the RNG clock is configured correctly (see [Section 19.3.6: RNG clocking](#)) and then it must clear the CEIS bit interrupt flag. The CECS bit is automatically cleared when the clocking condition is normal.

**Note:** The clock error has no impact on generated random numbers that is the application can still read the  $RNG\_DR$  register.

CEIS is set only when CECS is set to 1 by RNG.

### Noise source error detection

When a noise source (or seed) error occurs, the RNG stops generating random numbers and sets to 1 both SEIS and SECS bits to indicate that a seed error occurred. If a value is available in the  $RNG\_DR$  register, it must not be used as it may not have enough entropy.

In order to fully recover from a seed error application must clear the SEIS bit by writing it to "0", then clear and set the RGEN bit to reinitialize and restart the RNG.

### 19.3.8 RNG low-power use

If power consumption is a concern, the RNG can be disabled as soon as the DRDY bit is set to “1” by setting the RNGEN bit to “0” in the RNG\_CR register. The 32-bit random value stored in the RNG\_DR register is still available. If a new random is needed the application needs to re-enable the RNG and wait for 42+4 RNG clock cycles.

When disabling the RNG the user deactivates all the analog seed generators, whose power consumption is given in the datasheet electrical characteristics section.

## 19.4 RNG interrupts

In the RNG an interrupt can be produced on the following events:

- Data ready flag
- Seed error, see [Section 19.3.7: Error management](#)
- Clock error, see [Section 19.3.7: Error management](#)

Dedicated interrupt enable control bits are available as shown in [Table 122](#).

**Table 122. RNG interrupt requests**

Interrupt acronym	Interrupt event	Event flag	Enable control bit	Interrupt clear method
RNG	Data ready flag	DRDY	IE	None (automatic)
	Seed error flag	SEIS	IE	Write 0 to SEIS.
	Clock error flag	CEIS	IE	Write 0 to CEIS

The user can enable or disable the above interrupt sources individually by changing the mask bits or the general interrupt control bit IE in the RNG\_CR register. The status of the individual interrupt sources can be read from the RNG\_SR register.

*Note:* *Interrupts are generated only when RNG is enabled.*

## 19.5 RNG processing time

The RNG can produce one 32-bit random numbers every 42 RNG clock cycles.

After enabling or re-enabling the RNG using the RNGEN bit, it takes 46 RNG clock cycles before random data are available.

## 19.6 RNG entropy source validation

### 19.6.1 Introduction

In order to assess the amount of entropy available from the RNG, STMicroelectronics has tested the peripheral using NIST SP800-22 rev1a statistical tests. The results can be provided on demand or the customer can reproduce the tests.

For more information on running this NIST statistical test suite, refer to *STM32 microcontrollers random number generation validation using NIST statistical test suite* application note (AN4230), available on STMicroelectronics website.

### 19.6.2 Validation conditions

STMicroelectronics has tested the RNG true random number generator in the following conditions:

- RNG clock `rng_clk`= 48 MHz (CED bit = '0' in `RNG_CR` register) and `rng_clk` = 400 kHz (CED bit = '1' in `RNG_CR` register).

**Table 123. RNG configurations**

Configuration	RNG_CR bits						Loop number (N)	RNG_HTCR register	RNG_NSCR register
	NISTC bit	RNG_CONFIG1 [5:0]	CLKDIV [3:0]	RNG_CONFIG2 [2:0] <sup>(1)</sup>	RNG_CONFIG3 [3:0] <sup>(2)</sup>	CED bit			
A	Refer to <i>NIST compliant RNG configuration</i> table in AN4230 available from <a href="http://www.st.com">www.st.com</a> . This application note also indicates if this configuration is part of an existing NIST SP800-90B Entropy Certificate listed on <a href="https://csrc.nist.gov/projects/cryptographic-module-validation-program">https://csrc.nist.gov/projects/cryptographic-module-validation-program</a> .								
B	1	0x18	0x0 <sup>(3)</sup>	0x0	0x0	0	1	0x0000 NA <sup>(4)</sup>	default
C	0	0x0F		0x0	0xD	0	2		default

1. 0x1 value is recommended instead of 0x0 for `RNG_CONFIG2[2:0]`, when RNG power consumption is critical. See the end of [Section 19.3.8](#) for details.
2. `RNG_CONFIG3[1:0]` defines the loop number N: 0x0 corresponds to N=1, 0x1 to N=2, 0x2 to N=3, 0x3 to N=4
3. The noise source sampling must be NA or less. Hence, if the RNG clock is different from NA, this value of CLKDIV must be adapted. See the CLKDIV bitfield description in [Section 19.7.1](#) for details.
4. This value can be fixed in the RNG driver (it does not depend upon the STM32 product).

### 19.6.3 Data collection

In order to run statistical tests, it is required to collect samples from the entropy source at the raw data level as well as at the output of the entropy source. For details on data collection and the running of statistical test suites refer to AN4230 “STM32 microcontrollers random number generation validation using NIST statistical test suite” available from [www.st.com](http://www.st.com).

*In bypass mode the bits [31:30] of the fourth word are always stuck at 0. Hence the continuous capture of samples is started from the fifth word.*

## 19.7 RNG registers

The RNG is associated with a control register, a data register and a status register.

### 19.7.1 RNG control register (RNG\_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.															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CED	Res.	IE	RNGEN	Res.	Res.									
										rw		rw	rw		

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

Bit 5 **CED**: Clock error detection

0: Clock error detection enabled

1: Clock error detection is disabled

The clock error detection cannot be enabled nor disabled on-the-fly when the RNG is enabled, that is to enable or disable CED, the RNG must be disabled.

Bit 4 Reserved, must be kept at reset value.

Bit 3 **IE**: Interrupt enable

0: RNG interrupt is disabled

1: RNG interrupt is enabled. An interrupt is pending as soon as the DRDY, SEIS, or CEIS is set in the RNG\_SR register.

Bit 2 **RNGEN**: True random number generator enable

0: True random number generator is disabled. Analog noise sources are powered off and logic clocked by the RNG clock is gated.

1: True random number generator is enabled.

Bits 1:0 Reserved, must be kept at reset value.

## 19.7.2 RNG status register (RNG\_SR)

Address offset: 0x004

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SEIS	CEIS	Res.	Res.	SECS	CECS	DRDY								
									rc_w0	rc_w0			r	r	r

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

Bit 6 **SEIS**: Seed error interrupt status

This bit is set at the same time as SECS. It is cleared by writing 0. Writing 1 has no effect.

0: No faulty sequence detected

1: At least one faulty sequence is detected. See SECS bit description for details.

An interrupt is pending if IE = 1 in the RNG\_CR register.

Bit 5 **CEIS**: Clock error interrupt status

This bit is set at the same time as CECS. It is cleared by writing 0. Writing 1 has no effect.

0: The RNG clock is correct ( $f_{RNGCLK} > f_{HCLK}/16$ )

1: The RNG is detected too slow ( $f_{RNGCLK} < f_{HCLK}/16$ )

An interrupt is pending if IE = 1 in the RNG\_CR register.

Bits 4:3 Reserved, must be kept at reset value.

**Bit 2 SECS:** Seed error current status

0: No faulty sequence has currently been detected. If the SEIS bit is set, this means that a faulty sequence was detected and the situation has been recovered.

1: One of the noise sources has provided more than 64 consecutive bits at a constant value ("0" or "1"), or more than 32 consecutive occurrence of two bits patterns ("01" or "10")

**Bit 1 CECS:** Clock error current status

0: The RNG clock is correct ( $f_{RNGCLK} > f_{HCLK}/16$ ). If the CEIS bit is set, this means that a slow clock was detected and the situation has been recovered.

1: The RNG clock is too slow ( $f_{RNGCLK} < f_{HCLK}/16$ ).

*Note: CECS bit is valid only if the CED bit in the RNG\_CR register is set to 0.*

**Bit 0 DRDY:** Data ready

0: The RNG\_DR register is not yet valid, no random data is available.

1: The RNG\_DR register contains valid random data.

Once the RNG\_DR register has been read, this bit returns to 0 until a new random value is generated.

If IE=1 in the RNG\_CR register, an interrupt is generated when DRDY = 1.

### 19.7.3 RNG data register (RNG\_DR)

Address offset: 0x008

Reset value: 0x0000 0000

The RNG\_DR register is a read-only register that delivers a 32-bit random value when read. After being read, this register delivers a new random value after 42 periods of RNG clock if the output FIFO is empty.

The content of this register is valid when the DRDY = 1 and the value is not 0x0, even if RNGEN = 0.

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

**Bits 31:0 RNDATA[31:0]:** Random data

32-bit random data, which are valid when DRDY = 1. When DRDY = 0, the RNDATA value is zero.

When DRDY is set, it is recommended to always verify that RNG\_DR is different from zero. The zero value means that a seed error occurred between RNG\_SR polling and RND\_DR output reading (a rare event).

### 19.7.4 RNG register map

Table 124. RNG register map and reset map

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	RNG_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CEIS	CED	IE	SECS	RNGEN	0		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x004	RNG_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CECS	Res.	DRDY	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		
0x008	RNG_DR	RNDDATA[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		

Refer to [Section 2.2: Memory organization](#) for the register boundary addresses.

## 20 Cryptographic processor (CRYP)

This section applies to the whole STM32F756xx and STM32F750xx devices, unless otherwise specified.

### 20.1 Introduction

The cryptographic processor (CRYP) can be used both to encrypt and decrypt data using the DES, Triple-DES or AES algorithms. It is a fully compliant implementation of the following standards:

- The data encryption standard (DES) and Triple-DES (TDES) as defined by Federal Information Processing Standards Publication (FIPS PUB 46-3, Oct 1999), and the American National Standards Institute (ANSI X9.52)
- The advanced encryption standard (AES) as defined by Federal Information Processing Standards Publication (FIPS PUB 197, Nov 2001)

Multiple key sizes and chaining modes are supported:

- DES/TDES chaining modes ECB and CBC, supporting standard 56-bit keys with 8-bit parity per key
- AES chaining modes ECB, CBC, CTR, GCM, GMAC, CCM for key sizes of 128, 192 or 256 bits

The CRYP processor is a 32-bit AHB peripheral. It supports DMA transfers for incoming and outgoing data (two DMA channels are required). The peripheral also includes input and output FIFOs (each 8 words deep) for better performance.

### 20.2 CRYP main features

- Compliant implementation of the following standards:
  - NIST *FIPS publication 46-3, Data Encryption Standard (DES)*
  - ANSI X9.52, *Triple Data Encryption Algorithm Modes of Operation*
  - NIST *FIPS publication 197, Advanced Encryption Standard (AES)*
- AES symmetric block cipher implementation
  - 128-bit data block processing
  - Support for 128-, 192- and 256-bit cipher key lengths
  - Encryption and decryption with multiple chaining modes: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Counter mode (CTR), Galois Counter Mode (GCM), Galois Message Authentication Code mode (GMAC) and Counter with CBC-MAC (CCM)
  - 14 (respectively 18) clock cycles for processing one 128-bit block of data with a 128-bit (respectively 256-bit) key in AES-ECB mode
  - Integrated key scheduler with its key derivation stage (ECB or CBC decryption only)
- DES/TDES encryption/decryption implementation
  - 64-bit data block processing
  - Support for 64-, 128- and 192-bit cipher key lengths (including parity)

- Encryption and decryption with support of ECB and CBC chaining modes
- Direct implementation of simple DES algorithms (a single key K1 is used)
- 16 (respectively 48) clock cycles for processing one 64-bit block of data in DES (respectively TDES) ECB mode
- Software implementation of ciphertext stealing
- Features common to DES/TDES and AES
  - AMBA AHB target peripheral, accessible through 32-bit word single accesses only (otherwise an AHB bus error is generated, and write accesses are ignored)
  - 256-bit register for storing the cryptographic key (8x 32-bit registers)
  - 128-bit registers for storing initialization vectors (4x 32-bit)
  - 1x32-bit INPUT buffer associated with an internal IN FIFO of eight 32-bit words, corresponding to four incoming DES blocks or two AES blocks
  - 1x32-bit OUTPUT buffer associated with an internal OUT FIFO of eight 32-bit words, corresponding to four processed DES blocks or two AES blocks
  - Automatic data flow control supporting direct memory access (DMA) using two channels (one for incoming data, one for processed data). The OUT FIFO supports both single and burst transfers, while the IN FIFO supports only burst transfers.
  - Data swapping logic to support 1-, 8-, 16- or 32-bit data
  - Possibility for software to suspend a message if the cryptographic processor needs to process another message with higher priority (suspend/resume operation)

## 20.3 CRYP implementation

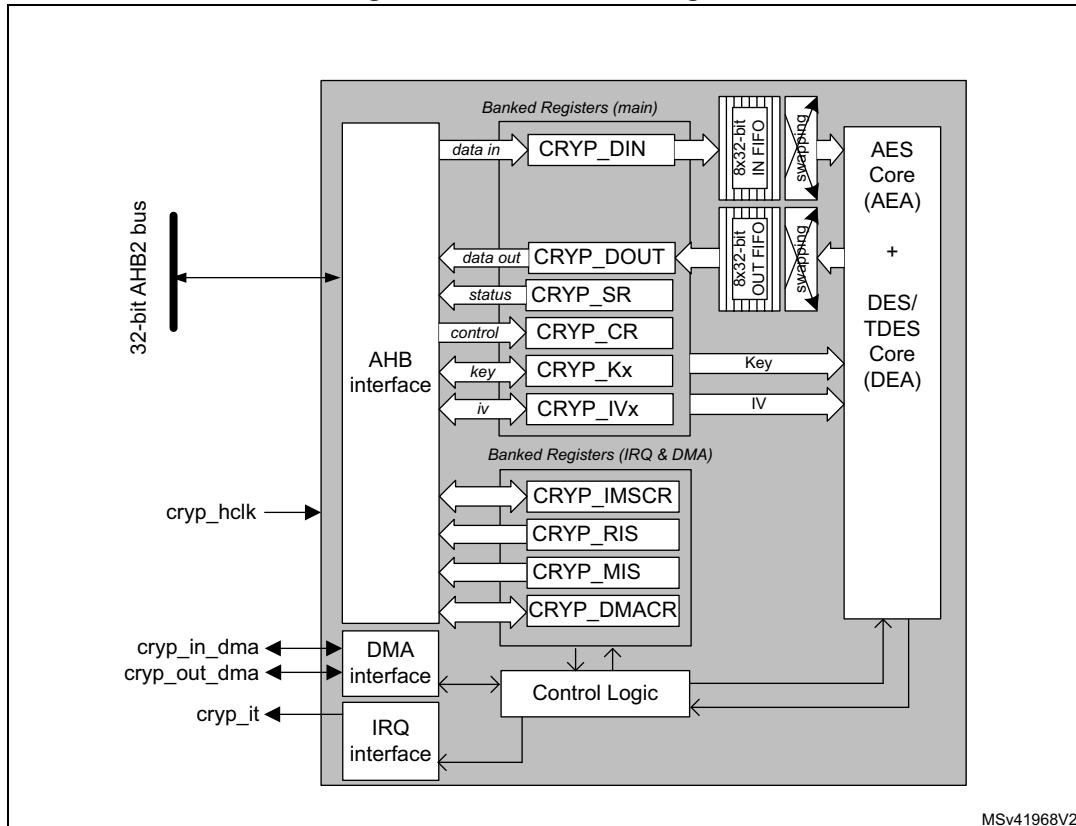
This device has one CRYP instance.

## 20.4 CRYP functional description

### 20.4.1 CRYP block diagram

The figure below shows the block diagram of the cryptographic processor.

Figure 118. CRYP block diagram



MSv41968V2

### 20.4.2 CRYP internal signals

[Table 125](#) provides a list of useful-to-know internal signals available at cryptographic processor level and not at STM32 product level (on pads).

Table 125. CRYP internal input/output signals

Signal name	Signal type	Description
cryp_hclk	Digital input	AHB bus clock
cryp_it	Digital output	Cryptographic processor global interrupt request
cryp_in_dma	Digital input/output	IN FIFO DMA burst request/ acknowledge
cryp_out_dma	Digital input/output	OUT FIFO DMA burst request/ acknowledge (with single request for DES/TDES)

## 20.4.3 CRYP DES/TDES cryptographic core

### Overview

The DES/Triple-DES cryptographic core consists of three components:

- The DES Algorithm (DEA core)
- Multiple keys (one for the DES algorithm, one to three for the TDES algorithm)
- The initialization vector, which is used only in CBC mode

The DES/Triple-DES cryptographic core provides two operating modes:

- **ALGODIR = 0:** Plaintext encryption using the key stored in the CRYP\_Kx registers.
- **ALGODIR = 1:** Ciphertext decryption using the key stored in the CRYP\_Kx registers.

The operating mode is selected by programming the ALGODIR bit in the CRYP\_CR register.

### Typical data processing

Typical usage of the cryptographic processor in DES modes can be found in [Section 20.4.10: CRYP DES/TDES basic chaining modes \(ECB, CBC\)](#).

*Note:* The outputs of the intermediate DEA stages are never revealed outside the cryptographic boundary, with the exclusion of the IV registers in CBC mode.

### DES/TDES keying and chaining modes

The DES/TDES allows three different keying options:

- *Three independent keys*

The first option specifies that all the keys are independent, that is, K1, K2 and K3 are independent. FIPS PUB 46-3 – 1999 (and ANSI X9.52 – 1998) refers to this option as the Keying Option 1 and, to the TDES as 3-key TDES.

- *Two independent keys*

The second option specifies that K1 and K2 are independent and K3 is equal to K1, that is, K1 and K2 are independent, K3 = K1. FIPS PUB 46-3 – 1999 (and ANSI X9.52 – 1998) refers to this second option as the Keying Option 2 and, to the TDES as 2-key TDES.

- *Three equal keys*

The third option specifies that K1, K2 and K3 are equal, that is:

$$K1 = K2 = K3$$

FIPS PUB 46-3 – 1999 (and ANSI X9.52 – 1998) refers to the third option as the Keying Option 3. This “1-key” TDES is equivalent to single DES.

The following chaining algorithms are supported by the DES hardware and can be selected through the ALGOMODE bits in the CRYP\_CR register:

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)

These modes are described in details in [Section 20.4.10: CRYP DES/TDES basic chaining modes \(ECB, CBC\)](#).

## 20.4.4 CRYP AES cryptographic core

### Overview

The AES cryptographic core consists of the following components:

- The AES algorithm (AEA core)
- The Multiplier over a binary Galois field (GF2mul)
- The key information
- The initialization vector (IV) or Nonce information
- Chaining algorithms logic (XOR, feedback/counter, mask)

The AES core works on 128-bit data blocks of (four words) with 128-, 192- or 256-bit key lengths. Depending on the chaining mode, the peripheral requires zero or one 128-bit initialization vector (IV).

The cryptographic peripheral features two operating modes:

- **ALGODIR = 0:** Plaintext encryption using the key stored in the CRYP\_Kx registers.
- **ALGODIR = 1:** Ciphertext decryption using the key stored in the CRYP\_Kx registers. When ECB and CBC chaining modes are selected, an initial key derivation process is automatically performed by the cryptographic peripheral.

The operating mode is selected by programming the ALGODIR bit in the CRYP\_CR register.

### Typical data processing

A description of cryptographic processor typical usage in AES mode can be found in [Section 20.4.11: CRYP AES basic chaining modes \(ECB, CBC\)](#).

*Note:* *The outputs of the intermediate AEA stages is never revealed outside the cryptographic boundary, with the exclusion of the IV registers.*

### AES chaining modes

The following chaining algorithms are supported by the cryptographic processor and can be selected through the ALGOMODE bits in the CRYP\_CR register:

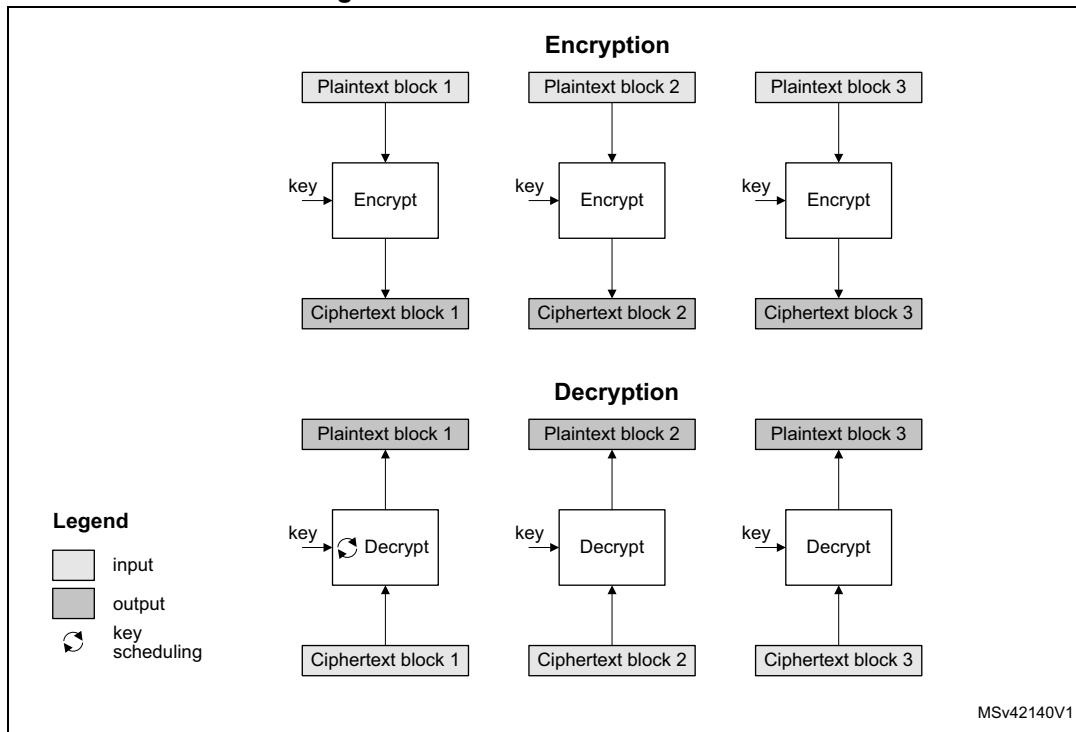
- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Counter Mode (CTR)
- Galois/Counter Mode (GCM)
- Galois Message Authentication Code mode (GMAC)
- Counter with CBC-MAC (CCM)

A quick introduction on these chaining modes can be found in the following subsections.

For detailed instructions, refer to [Section 20.4.11: CRYP AES basic chaining modes \(ECB, CBC\)](#) and onward.

## AES Electronic CodeBook (ECB)

Figure 119. AES-ECB mode overview

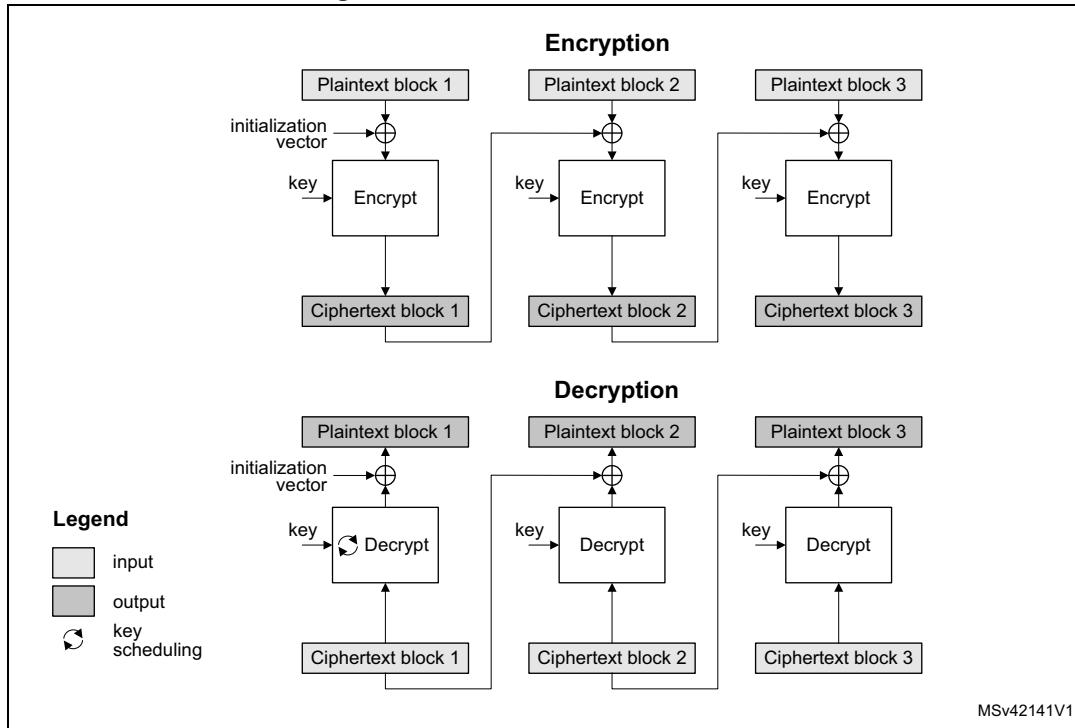


ECB is the simplest operating mode. There are no chaining operations, and no special initialization stage. The message is divided into blocks and each block is encrypted or decrypted separately.

*Note:* For decryption, a special key scheduling is required before processing the first block.

## AES Cipher block chaining (CBC)

Figure 120. AES-CBC mode overview

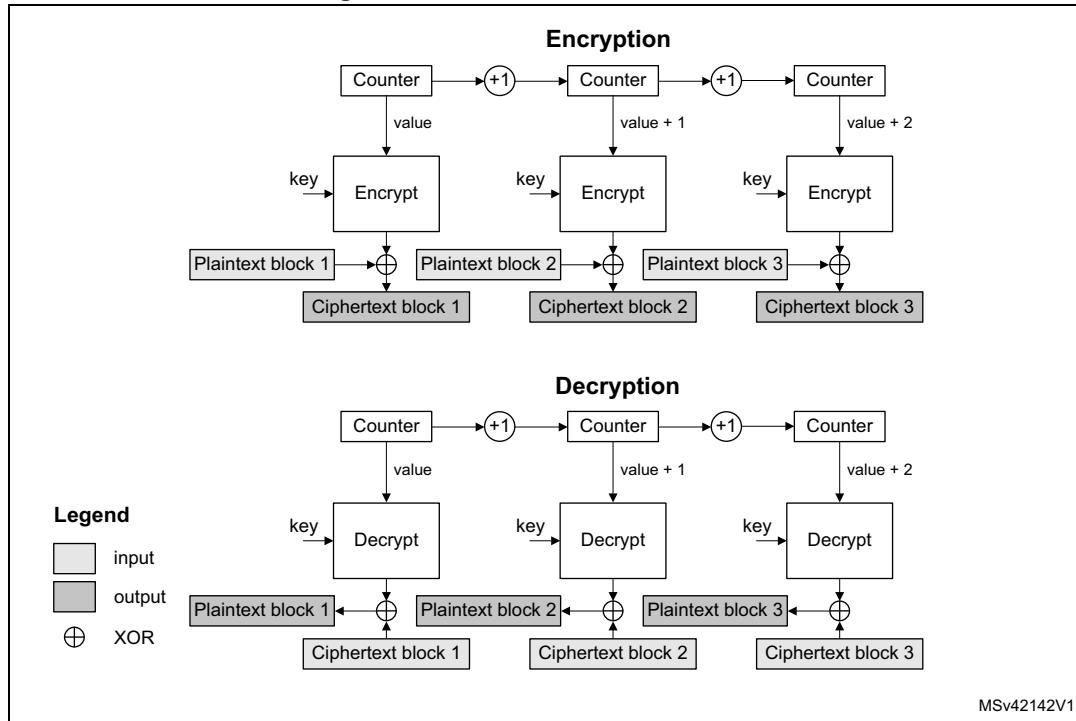


CBC operating mode chains the output of each block with the input of the following block. To make each message unique, an initialization vector is used during the first block processing.

**Note:** *For decryption, a special key scheduling is required before processing the first block.*

## AES Counter mode (CTR)

Figure 121. AES-CTR mode overview

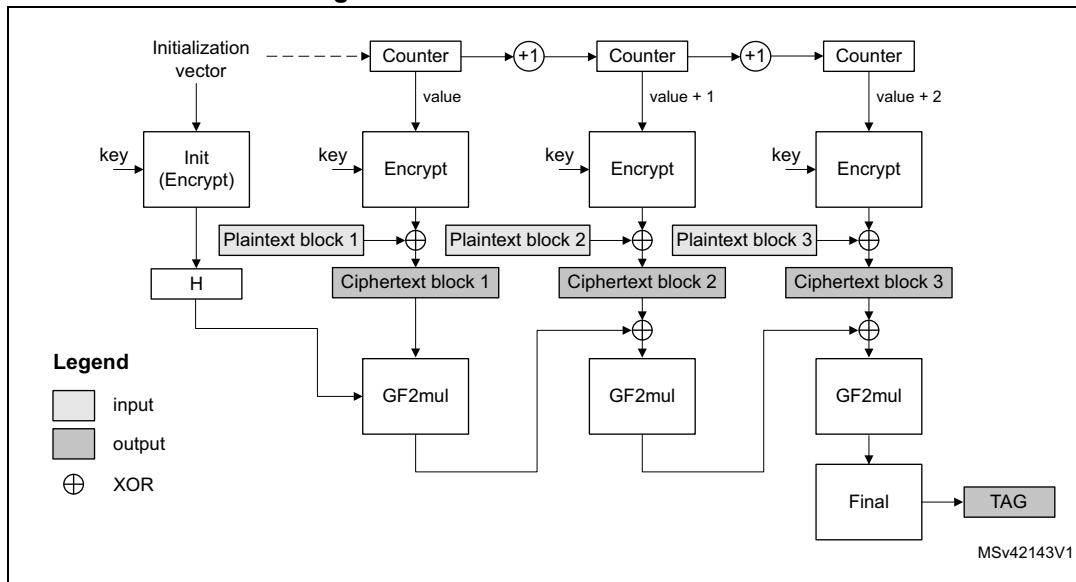


The CTR mode uses the AES core to generate a key stream; these keys are then XORed with the plaintext to obtain the ciphertext as specified in NIST Special Publication 800-38A, *Recommendation for Block Cipher Modes of Operation*.

**Note:** Unlike ECB and CBC modes, no key scheduling is required for the CTR decryption, since in this chaining scheme the AES core is always used in encryption mode for producing the counter blocks.

### AES Galois/Counter mode (GCM)

Figure 122. AES-GCM mode overview

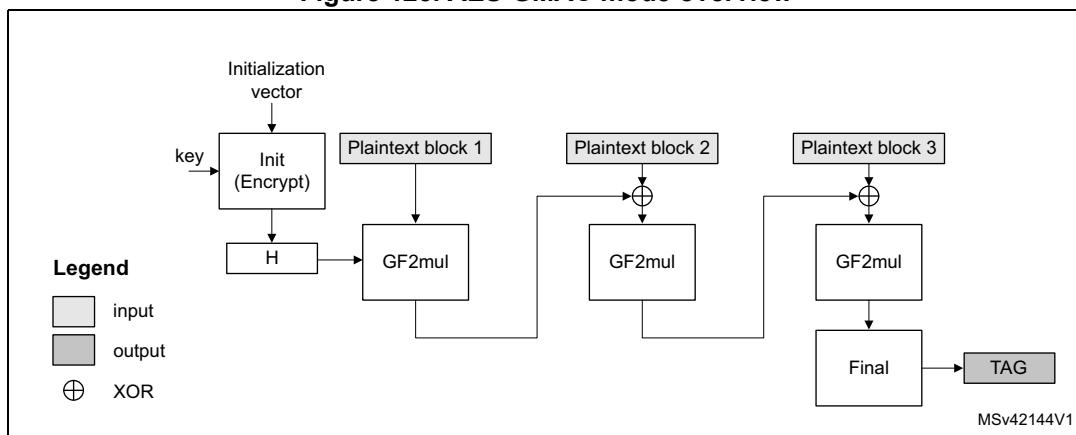


In Galois/Counter mode (GCM), the plaintext message is encrypted, while a message authentication code (MAC) is computed in parallel, thus generating the corresponding ciphertext and its MAC (also known as authentication tag). It is defined in NIST Special Publication 800-38D, *Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) and GMAC*.

GCM mode is based on AES in counter mode for confidentiality. It uses a multiplier over a fixed finite field for computing the message authentication code. It requires an initial value and a particular 128-bit block at the end of the message.

### AES Galois message authentication code (GMAC)

Figure 123. AES-GMAC mode overview

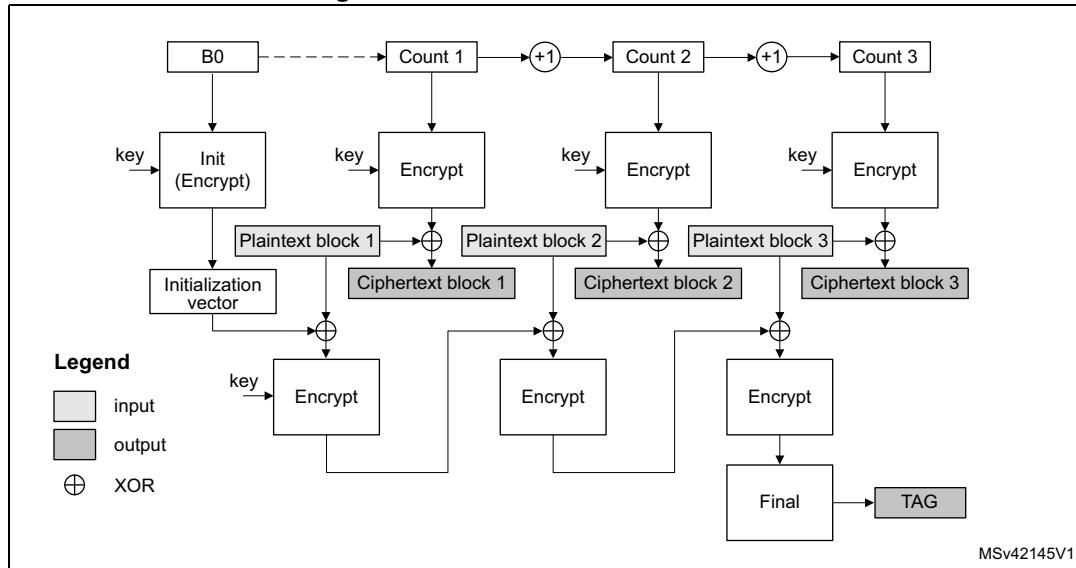


Galois Message Authentication Code (GMAC) allows authenticating a message and generating the corresponding message authentication code (MAC). It is defined in NIST *Special Publication 800-38D, Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) and GMAC*.

GMAC is similar to Galois/Counter mode (GCM), except that it is applied on a message composed only by clear-text authenticated data (i.e. only header, no payload).

### AES Counter with CBC-MAC (CCM)

Figure 124. AES-CCM mode overview



In Counter with Cipher Block Chaining-Message Authentication Code (CCM), the plaintext message is encrypted while a message authentication code (MAC) is computed in parallel, thus generating the corresponding ciphertext and the corresponding MAC (also known as tag). It is described by NIST in *Special Publication 800-38C, Recommendation for Block Cipher Modes of Operation - The CCM Mode for Authentication and Confidentiality*.

CCM mode is based on AES in counter mode for confidentiality and it uses CBC for computing the message authentication code. It requires an initial value.

Like GCM CCM chaining mode, AES-CCM mode can be applied on a message composed only by cleartext authenticated data (i.e. only header, no payload). Note that this way of using CCM is not called CMAC (it is not similar to GCM/GMAC), and its usage is not recommended by NIST.

## 20.4.5 CRYP procedure to perform a cipher operation

### Introduction

To understand how the cryptographic peripheral operates, a typical cipher operation is described below. For the detailed peripheral usage according to the cipher mode, refer to the specific section, for example [Section 20.4.11: CRYP AES basic chaining modes \(ECB, CBC\)](#).

### CRYP initialization

To initialize the cryptographic processor, first disable it by clearing the CRYPTEN bit in the CRYP\_CR register. Then execute the following steps:

1. Configure the algorithm and the chaining mode through the ALGOMODE and ALGODIR bits in the CRYP\_CR register. Configure also the key size with the KEYSIZE bits.
  - When ALGODIR is set to 1 (decryption) and the selected algorithm/chaining mode is AES-ECB or AES-CBC, an initial key derivation of the encryption key must be performed as described in [Section 20.4.7: Preparing the CRYP AES key for decryption](#).
2. When the previous step is complete, and when applicable, write the symmetric key into the CRYP\_KxL/R registers. The way to write key registers is defined in [Section 20.4.17: CRYP key registers](#)
3. Configure the data type (1, 8, 16 or 32 bits) through the DATATYPE bits of the CRYP\_CR register.
4. When it is required (for example for CBC or CTR chaining modes), write the initialization vectors into the CRYP\_IVx(L/R)R register.
5. Flush the IN and OUT FIFOs by writing the FFLUSH bit to 1 in the CRYP\_CR register.

### Preliminary warning for all cases

If the ECB or CBC mode is selected and data are not a multiple of 64 bits (for DES) or 128 bits (for AES), the second and the last block management is more complex than the sequences below. Refer to [Section 20.4.8: CRYP stealing and data padding](#) for more details.

### Appending data using the CPU in Polling mode

1. Enable the cryptographic processor by setting to 1 the CRYPEN bit in the CRYP\_CR register.
2. Write data in the IN FIFO (one block or until the FIFO is full).
3. Repeat the following sequence until the second last block of data has been processed:
  - a) Wait until the not-empty-flag OFNE is set to 1, then read the OUT FIFO (one block or until the FIFO is empty).
  - b) Wait until the not-full-flag IFNF is set to 1, then write the IN FIFO (one block or until the FIFO is full) except if it is the last block.
4. The BUSY bit is set automatically by the cryptographic processor. At the end of the processing, the BUSY bit returns to 0 and both FIFOs are empty (IN FIFO empty flag IFEM = 1 and OUT FIFO not empty flag OFNE = 0).
5. If the next processing block is the last block, the CPU must pad (when applicable) the data with zeroes to obtain a complete block
6. When the operation is complete, the cryptographic processor can be disabled by clearing the CRYPEN bit in CRYP\_CR register.

### Appending data using the CPU in Interrupt mode

1. Enable the interrupts by setting the INIM and OUTIM bits in the CRYP\_IMSCR register.
2. Enable the cryptographic processor by setting to 1 the CRYPEN bit in the CRYP\_CR register.
3. In the interrupt service routine that manages the input data:
  - a) If the last block is being loaded, the CPU must pad (when applicable) the data with zeroes to have a complete block. Then load the block into the IN FIFO.
  - b) If it is not the last block, load the data into the IN FIFO. You can load only one block (2 words for DES, 4 words for AES), or load data until the FIFO is full.
  - c) In all cases, after the last word of data has been written, disable the interrupt by clearing the INIM interrupt mask.
4. In the interrupt service routine that manages the output data:
  - a) Read the output data from the OUT FIFO. You can read only one block (2 words for DES, 4 words for AES), or read data until the FIFO is empty.
  - b) When the last word has been read, INIM and BUSY bits are set to 0 and both FIFOs are empty (IFEM = 1 and OFNE = 0). You can disable the interrupt by clearing the OUTIM bit, and disable the peripheral by clearing the CRYPEN bit.
  - c) If you read the last block of cleartext data (i.e. decryption), optionally discard the data that is not part of message/payload.

### Appending data using the DMA

1. Prepare the last block of data by optionally padding it with zeroes to have a complete block.
2. Configure the DMA controller to transfer the input data from the memory and transfer the output data from the peripheral to the memory, as described in [Section 20.4.19](#):

- CRYP DMA interface.* The DMA should be configured to set an interrupt on transfer completion to indicate that the processing is complete.
3. Enable the cryptographic processor by setting to 1 the CRYPEN bit in CRYP\_CR register, then enable the DMA IN and OUT requests by setting to 1 the DIEN and DOEN bits in the CRYP\_DMCR register.
  4. All the transfers and processing are managed by the DMA and the cryptographic processor. The DMA interrupt indicates that the processing is complete. Both FIFOs are normally empty and BUSY flag is set 0.

**Caution:** It is important that DMA controller empties the cryptographic processor output FIFO before filling up the cryptographic processor input FIFO. To achieve this, the DMA controller should be configured so that the transfer from the cryptographic peripheral to the memory has a higher priority than the transfer from the memory to the cryptographic peripheral.

#### 20.4.6 CRYP busy state

The cryptographic processor is busy and processing data (BUSY set to 1 in CRYP\_SR register) when all the conditions below are met:

- CRYPEN = 1 in CRYP\_CR register.
- There are enough data in the input FIFO (at least two words for the DES or TDES algorithm mode, four words for the AES algorithm mode).
- There is enough free-space in the output FIFO (at least two word locations for DES, four for AES).

Write operations to the CRYP\_Kx(L/R)R key registers, to the CRYP\_IVx(L/R)R initialization registers, or to bits [9:2] of the CRYP\_CR register, are ignored when cryptographic processor is busy (i.e. the registers are not modified). It is thus not possible to modify the configuration of the cryptographic processor while it is processing a data block.

It is possible to clear the CRYPEN bit while BUSY bit is set to 1. In this case the ongoing DES/TDES or AES processing first completes (i.e. the word results are written to the output FIFO) before the BUSY bit is cleared by hardware.

**Note:** *If the application needs to suspend a message to process another one with a higher priority, refer to [Section 20.4.9: CRYP suspend/resume operations](#)*

When a block is being processed in DES or TDES mode, if the output FIFO becomes full and the input FIFO contains at least one new block, then the new block is popped off the input FIFO and the BUSY bit remains high until there is enough space to store this new block into the output FIFO.

### 20.4.7 Preparing the CRYP AES key for decryption

When performing an AES **ECB or CBC decryption**, the AES key has to be prepared. Indeed, in AES encryption the round 0 key is the one stored in the key registers, and AES decryption must start using the last round key. Hence, as the encryption key is stored in memory, a special key scheduling must be performed to obtain the decryption key. This preparation is not required in any other AES modes than ECB or CBC decryption.

When the cryptographic processor is disabled (CRYPPEN = 0), the CRYP key preparation process is performed as follows:

1. Program ALGOMODE bits to 0x7 and ALGODIR to 0x0 in CRYP\_CR. In addition, configure the key length with the KEYSIZE bits.
2. Write the symmetric key to the CRYP\_KxL/R registers, as described in [Section 20.4.17: CRYP key registers](#).
3. Enable the cryptographic processor by setting the CRYPPEN bit in the CRYP\_CR register. It immediately starts an AES round for key preparation (BUSY = 1).
4. Wait until the BUSY bit is cleared in the CRYP\_SR register. Then update ALGOMODE bits in the CRYP\_CR register to select the correct chaining mode, that is 0x4 for ECB or 0x5 for CBC.
5. The AES key is available in the CRYP key registers, ready to use for decryption.

**Note:** As the CRYPPEN bitfield is reset by hardware at the end of the key preparation, the application software must set it again for the next operation.

The latency of the key preparation operation is 14, 16 or 18 clock cycles depending on the key size (128, 192 or 256 bits).

### 20.4.8 CRYP stealing and data padding

When using DES or AES algorithm in **ECB or CBC** modes to manage messages that are not multiple of the block size (64 bits for DES, 128 bits for AES), use ciphertext stealing techniques such as those described in *NIST Special Publication 800-38A, Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode*. Since the cryptographic processor does not implement such techniques, **the last two blocks** must be handled in a special way by the application.

**Note:** *Ciphertext stealing techniques are not documented in this reference manual.*

Similarly, when the AES algorithm is used in other modes than ECB or CBC, incomplete input data blocks (i.e. block shorter than 128 bits) have to be padded with zeroes by the application prior to encryption (i.e. extra bits should be appended to the trailing end of the data string). After decryption, the extra bits have to be discarded. The cryptographic processor does not implement automatic data padding operation to **the last block**, so the application should follow the recommendation given in [Section 20.4.5: CRYP procedure to perform a cipher operation](#) to manage messages that are not multiple of 128 bits.

**Note:** *Padding data are swapped in a similar way as normal data, according to the DATATYPE field in CRYP\_CR register (see [Section 20.4.16: CRYP data registers and data swapping](#) for details).*

With this version of cryptographic processor, a special workaround is required in order to properly compute authentication tags while doing a **GCM encryption** or a **CCM decryption** with the last block of payload size **inferior to 128 bits**. This workaround is described below:

- During GCM encryption payload phase and before inserting a last plaintext block smaller than 128 bits, the application has to follow the below sequence:
  - a) Disable the peripheral by setting the CRYPEN bit to 0 in CRYP\_CR.
  - b) Load CRYP\_IV1RR register content in a temporary variable. Decrement the value by 1 and reinsert the result in CRYP\_IV1RR register.
  - c) Change the AES mode to CTR mode by writing the ALGOMODE bitfield to 0x6 in the CRYP\_CR register.
  - a) Set the CRYPEN bit to 1 in CRYP\_CR to enable again the peripheral.
  - b) Pad the last block (smaller than 128 bits) with zeros to have a complete block of 128 bits, then write it into CRYP\_DIN register.
  - c) Upon encryption completion, read the 128-bit generated ciphertext from the CRYP\_DOUT register and store it as intermediate data.
  - d) Change again the AES mode to GCM mode by writing the ALGOMODE bitfield to 0x8 in the CRYP\_CR register.
  - e) Select Final phase by writing the GCM\_CCMPH bitfield to 0b11 in the CRYP\_CR register.
  - f) In the intermediate data, set to 0 the bits corresponding to the padded bits of the last payload block then insert the resulting data to CRYP\_DIN register.
  - g) When the operation is complete, read data from CRYP\_DOUT. These data have to be discarded.
  - h) Apply the normal Final phase as described in [Section 20.4.13: CRYP AES Galois/counter mode \(GCM\)](#).
- During CCM decryption payload phase and before inserting a last ciphertext block smaller than 128 bits, the application has to follow the below sequence:
  - a) To disable the peripheral, set the CRYPEN bit to 0 in CRYP\_CR.
  - b) Load CRYP\_IV1RR in a temporary variable (named here `IV1temp`).
  - c) Load CRYP\_CSGCMCCM0R, CRYP\_CSGCMCCM1R, CRYP\_CSGCMCCM2R, and CRYP\_CSGCMCCM3R register contents from LSB to MSB in 128-bit temporary variable (named here `temp1`).
  - d) Load in CRYP\_IV1RR the content previously stored in `IV1temp`.
  - e) Change the AES mode to CTR mode by writing the ALGOMODE bitfield to 0x6 in the CRYP\_CR register.
  - a) Set the CRYPEN bit to 1 in CRYP\_CR to enable again the peripheral.
  - b) Pad the last block (smaller than 128 bits) with zeros to have a complete block of 128 bits, then write it to CRYP\_DIN register.
  - c) Upon decryption completion, read the 128-bit generated data from DOUT register, and store them as intermediate data (here named `intdata_o`).
  - d) Save again CRYP\_CSGCMCCM0R, CRYP\_CSGCMCCM1R, CRYP\_CSGCMCCM2R, and CRYP\_CSGCMCCM3R register contents, from LSB to MSB, in a new 128-bit temporary variable (named here `temp2`).
  - e) Change again the AES mode to CCM mode by writing the ALGOMODE bitfield to 0x9 in the CRYP\_CR register.

- f) Select the header phase by writing the GCM\_CCMPH bitfield to 0b01 in the CRYP\_CR register.
- g) In the intermediate data (`intdata_o` which was generated with CTR), set to 0 the bits corresponding to the padded bits of the last payload block, XOR with `temp1`, XOR with `temp2`, and insert the resulting data into CRYP\_DIN register. In other words:  

$$\text{CRYP\_DIN} = (\text{intdata}_o \text{ AND mask}) \text{ XOR temp1 XOR temp2}.$$
- h) Wait for operation completion.
- i) Apply the normal Final phase as described in [Section 20.4.15: CRYP AES Counter with CBC-MAC \(CCM\)](#).

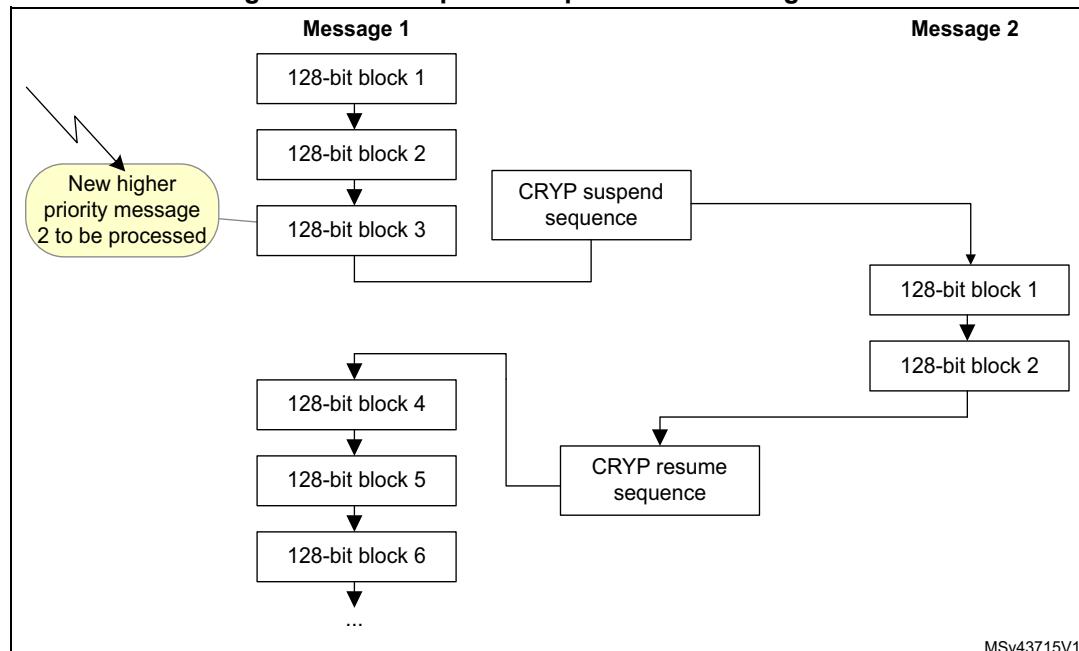
#### 20.4.9 CRYP suspend/resume operations

A message can be suspended if another message with a higher priority has to be processed. When this highest priority message has been sent, the suspended message can be resumed in both encryption or decryption mode.

Suspend/resume operations do not break the chaining operation and the message processing can be resumed as soon as cryptographic processor is enabled again to receive the next data block.

[Figure 125](#) gives an example of suspend.resume operation: message 1 is suspended in order to send a higher priority message (message 2), which is shorter than message 1 (AES algorithm).

**Figure 125. Example of suspend mode management**



A detailed description of suspend/resume operations can be found in each AES mode section.

### 20.4.10 CRYP DES/TDES basic chaining modes (ECB, CBC)

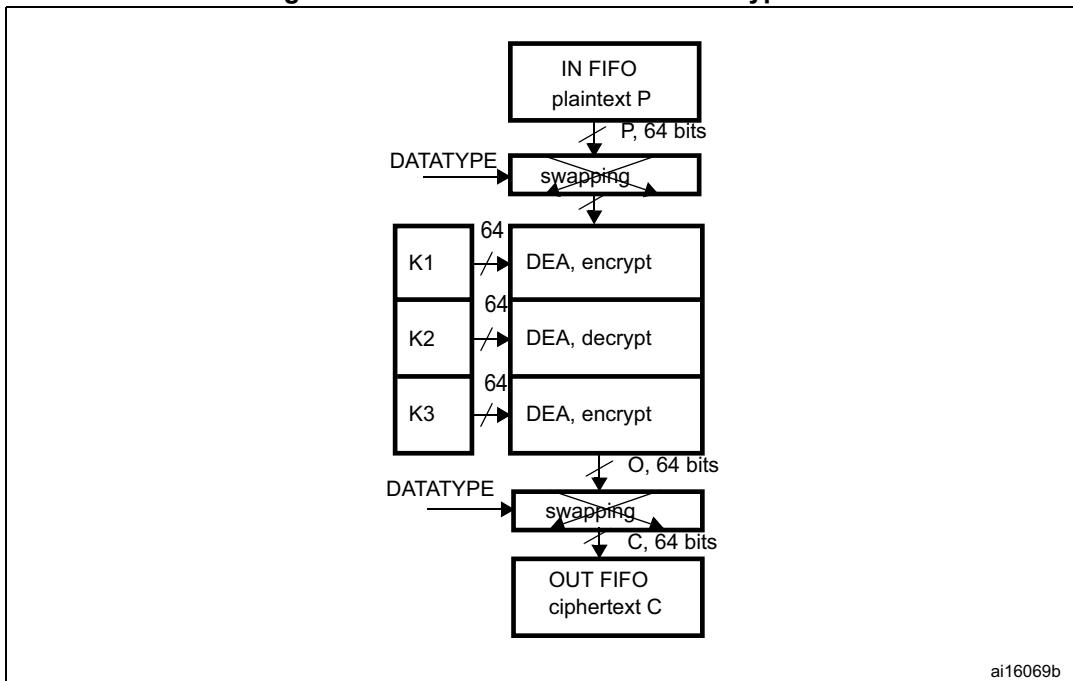
#### Overview

FIPS PUB 46-3 – 1999 (and ANSI X9.52-1998) provides a thorough explanation of the processing involved in the four operation modes supplied by the DES computing core: TDES-ECB encryption, TDES-ECB decryption, TDES-CBC encryption and TDES-CBC decryption. This section only gives a brief explanation of each mode.

#### DES/TDES-ECB encryption

*Figure 126* illustrates the encryption in DES and TDES Electronic CodeBook (DES/TDES-ECB) mode. This mode is selected by programming ALGOMODE 0x0 and ALGODIR to 0 in CRYP\_CR.

**Figure 126. DES/TDES-ECB mode encryption**



ai16069b

1. K: key; C: cipher text; I: input block; O: output block; P: plain text.

A 64-bit plaintext data block (P) is used after bit/byte/half-word as the input block (I). The input block is processed through the DEA in the encrypt state using K1. The output of this process is fed back directly to the input of the DEA where the DES is performed in the decrypt state using K2. The output of this process is fed back directly to the input of the DEA where the DES is performed in the encrypt state using K3. The resultant 64-bit output block (O) is used, after bit/byte/half-word swapping, as ciphertext (C) and it is pushed into the OUT FIFO.

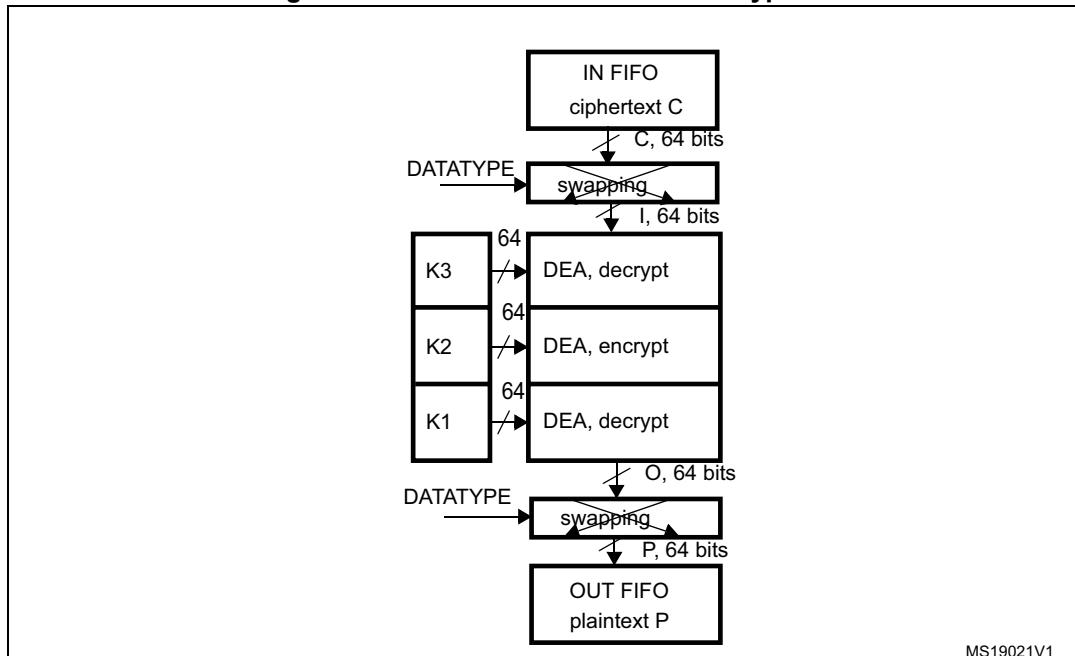
**Note:** *For more information on data swapping, refer to Section 20.4.16: CRYP data registers and data swapping.*

*Detailed DES/TDES encryption sequence can be found in Section 20.4.5: CRYP procedure to perform a cipher operation.*

### DES/TDES-ECB mode decryption

[Figure 127](#) illustrates the decryption in DES and TDES Electronic CodeBook (DES/TDES-ECB) mode. This mode is selected by programming ALGOMODE to 0x0 and ALGODIR to 1 in CRYP\_CR.

**Figure 127. DES/TDES-ECB mode decryption**



1. K: key; C: cipher text; I: input block; O: output block; P: plain text.

A 64-bit ciphertext block (C) is used, after bit/byte/half-word swapping, as the input block (I). The keying sequence is reversed compared to that used in the encryption process. The input block is processed through the DEA in the decrypt state using K3. The output of this process is fed back directly to the input of the DEA where the DES is performed in the encrypt state using K2. The new result is directly fed to the input of the DEA where the DES is performed in the decrypt state using K1. The resultant 64-bit output block (O), after bit/byte/half-word swapping, produces the plaintext (P).

Note:

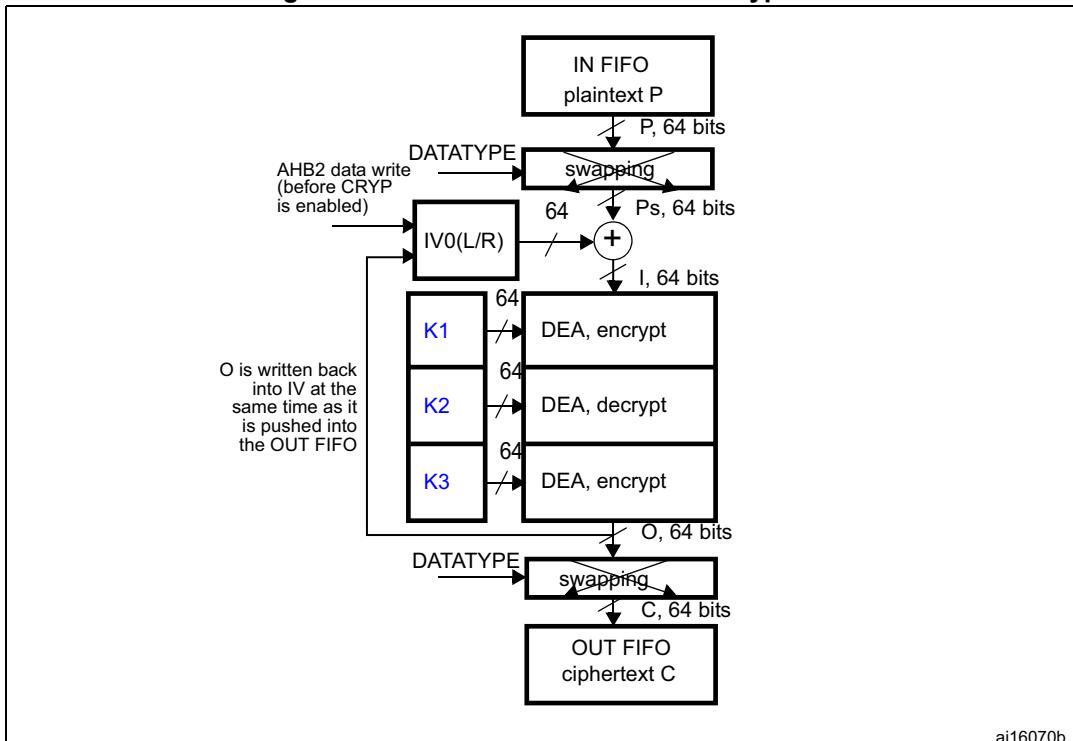
*For more information on data swapping refer to [Section 20.4.16: CRYP data registers and data swapping](#).*

*Detailed DES/TDES encryption sequence can be found in [Section 20.4.5: CRYP procedure to perform a cipher operation](#).*

### DES/TDES-CBC encryption

*Figure 128* illustrates the encryption in DES and TDES Cipher Block Chaining (DES/TDES-ECB) mode. This mode is selected by programming ALGOMODE to 0x1 and ALGODIR to 0 in CRYP\_CR.

**Figure 128. DES/TDES-CBC mode encryption**



ai16070b

K: key; C: cipher text; I: input block; O: output block; Ps: plain text before swapping (when decoding) or after swapping (when encoding); P: plain text; IV: initialization vectors.

This mode begins by dividing a plaintext message into 64-bit data blocks. In TCBC encryption, the first input block ( $I_1$ ), obtained after bit/byte/half-word swapping, is formed by exclusive-ORing the first plaintext data block ( $P_1$ ) with a 64-bit initialization vector IV ( $I_1 = IV \oplus P_1$ ). The input block is processed through the DEA in the encrypt state using  $K1$ . The output of this process is fed back directly to the input of the DEA, which performs the DES in the decrypt state using  $K2$ . The output of this process is fed directly to the input of the DEA, which performs the DES in the encrypt state using  $K3$ . The resultant 64-bit output block ( $O_1$ ) is used directly as the ciphertext ( $C_1$ ), that is,  $C_1 = O_1$ .

This first ciphertext block is then exclusive-ORed with the second plaintext data block to produce the second input block, ( $I_2$ ) = ( $C_1 \oplus P_2$ ). Note that  $I_2$  and  $P_2$  now refer to the second block. The second input block is processed through the TDEA to produce the second ciphertext block.

This encryption process continues to “chain” successive cipher and plaintext blocks together until the last plaintext block in the message is encrypted.

If the message does not consist of an integral number of data blocks, then the final partial data block should be encrypted in a manner specified for the application.

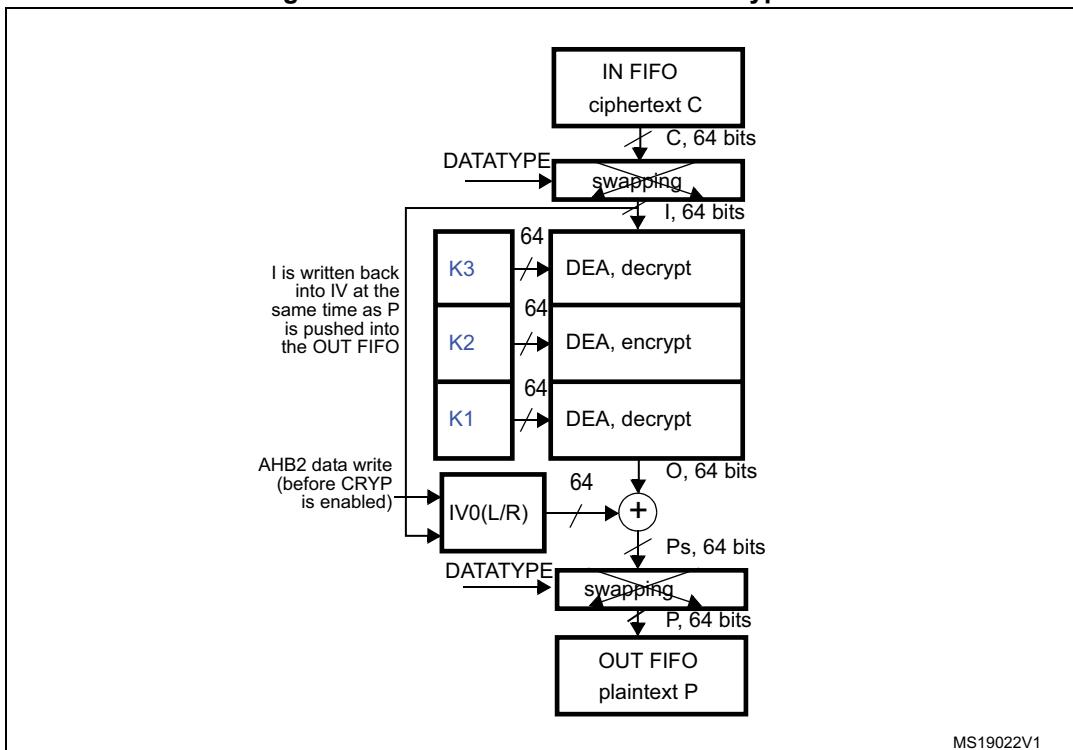
**Note:** For more information on data swapping refer to [Section 20.4.16: CRYP data registers and data swapping](#).

Detailed DES/TDES encryption sequence can be found in [Section 20.4.5: CRYP procedure to perform a cipher operation](#).

### DES/TDES-CBC decryption

[Figure 128](#) illustrates the decryption in DES and TDES Cipher Block Chaining (DES/TDES-ECB) mode. This mode is selected by writing ALGOMODE to 0x1 and ALGODIR to 1 in CRYP\_CR.

**Figure 129. DES/TDES-CBC mode decryption**



1. K: key; C: cipher text; I: input block; O: output block; Ps: plain text before swapping (when decoding) or after swapping (when encoding); P: plain text; IV: initialization vectors.

In this mode the first ciphertext block ( $C_1$ ) is used directly as the input block ( $I_1$ ). The keying sequence is reversed compared to that used for the encrypt process. The input block is processed through the DEA in the decrypt state using K3. The output of this process is fed directly to the input of the DEA where the DES is processed in the encrypt state using K2. This resulting value is directly fed to the input of the DEA where the DES is processed in the decrypt state using K1. The resulting output block is exclusive-ORed with the IV (which must be the same as that used during encryption) to produce the first plaintext block ( $P_1 = O_1 \oplus IV$ ).

The second ciphertext block is then used as the next input block and is processed through the TDEA. The resulting output block is exclusive-ORed with the first ciphertext block to produce the second plaintext data block ( $P_2 = O_2 \oplus C_1$ ). Note that  $P_2$  and  $O_2$  refer to the second block of data.

The DES/TDES-CBC decryption process continues in this manner until the last complete ciphertext block has been decrypted.

Ciphertext representing a partial data block must be decrypted in a manner specified for the application.

**Note:** *For more information on data swapping refer to [Section 20.4.16: CRYP data registers and data swapping](#).*

*Detailed DES/TDES encryption sequence can be found in [Section 20.4.5: CRYP procedure to perform a cipher operation](#).*

### DES/TDES suspend/resume operations in ECB/CBC modes

Before interrupting the current message, the user application must respect the following steps:

1. If DMA is used, stop the DMA transfers to the IN FIFO by clearing to 0 the DIEN bit in the CRYP\_DMACR register.
2. Wait until both the IN and the OUT FIFOs are empty (IFEM = 1 and OFNE = 0 in the CRYP\_SR) and the BUSY bit is cleared. Alternatively, as the input FIFO can contain up to four unprocessed DES blocks, the application could decide for real-time reason to interrupt the cryptographic processing without waiting for the IN FIFO to be empty. In this case, the alternative is:
  - a) Wait until OUT FIFO is empty (OFNE = 0).
  - b) Read back the data loaded in the IN FIFO that have not been processed and save them in the memory until the IN FIFO is empty.
3. If DMA is used stop the DMA transfers from the OUT FIFO by clearing to 0 the DOEN bit in the CRYP\_DMACR register.
4. Disable the cryptographic processor by setting the CRYPPEN bit to 0 in CRYP\_CR, then save the current configuration (bits [9:2] in the CRYP\_CR register). If CBC mode is selected, save the initialization vector registers, since CRYP\_IVx(L/R)R registers have changed from initial values during the data processing.

**Note:** *Key registers do not need to be saved as the original key value is known by the application.*

5. If DMA is used, save the DMA controller status (such as the pointers to IN and OUT data transfers, number of remaining bytes).

To resume message processing, the user application must respect the following sequence:

1. If DMA is used, reconfigure the DMA controller to complete the rest of the FIFO IN and FIFO OUT transfers.
2. Make sure the cryptographic processor is disabled by reading the CRYPPEN bit in CRYP\_CR (it must be 0).
3. Configure again the cryptographic processor with the initial setting in CRYP\_CR, as well as the key registers using the saved configuration.
4. If the CBC mode is selected, restore CRYP\_IVx(L/R)R registers using the saved configuration.
5. Optionally, write the data that were saved during context saving into the IN FIFO.
6. Enable the cryptographic processor by setting the CRYPPEN bit to 1.
7. If DMA is used, enable again DMA requests for the cryptographic processor, by setting to 1 the DIEN and DOEN bits in the CRYP\_DMACR register.

### 20.4.11 CRYP AES basic chaining modes (ECB, CBC)

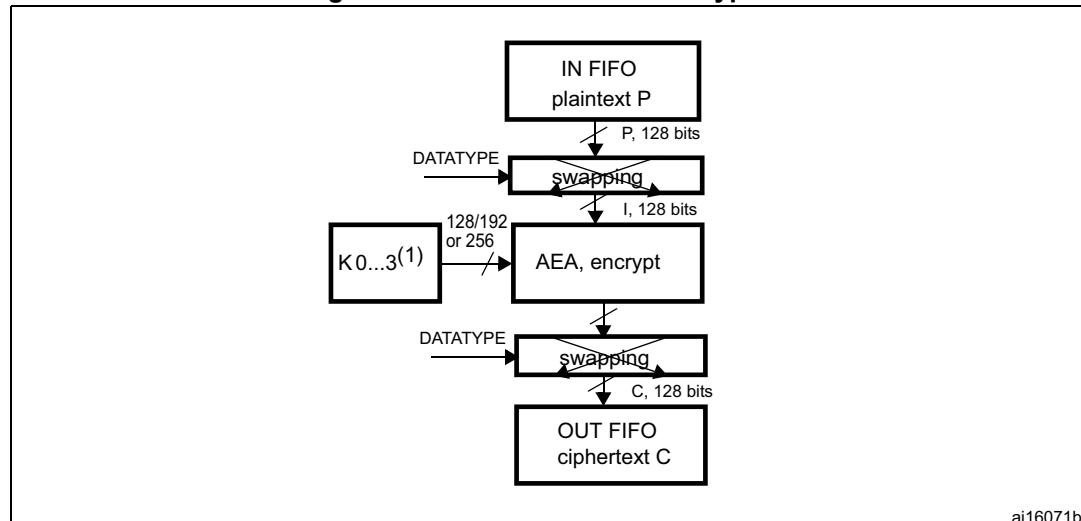
#### Overview

FIPS PUB 197 (November 26, 2001) provides a thorough explanation of the processing involved in the four basic operation modes supplied by the AES computing core: AES-ECB encryption, AES-ECB decryption, AES-CBC encryption and AES-CBC decryption. This section only gives a brief explanation of each mode.

#### AES ECB encryption

*Figure 130* illustrates the AES Electronic codebook (AES-ECB) mode encryption. This mode is selected by writing ALGOMODE to 0x4 and ALGODIR to 0 in CRYP\_CR.

**Figure 130. AES-ECB mode encryption**



ai16071b

1. K: key; C: cipher text; I: input block; O: output block; P: plain text.
2. If Key size = 128: Key = [K3 K2].  
If Key size = 192: Key = [K3 K2 K1].  
If Key size = 256: Key = [K3 K2 K1 K0].

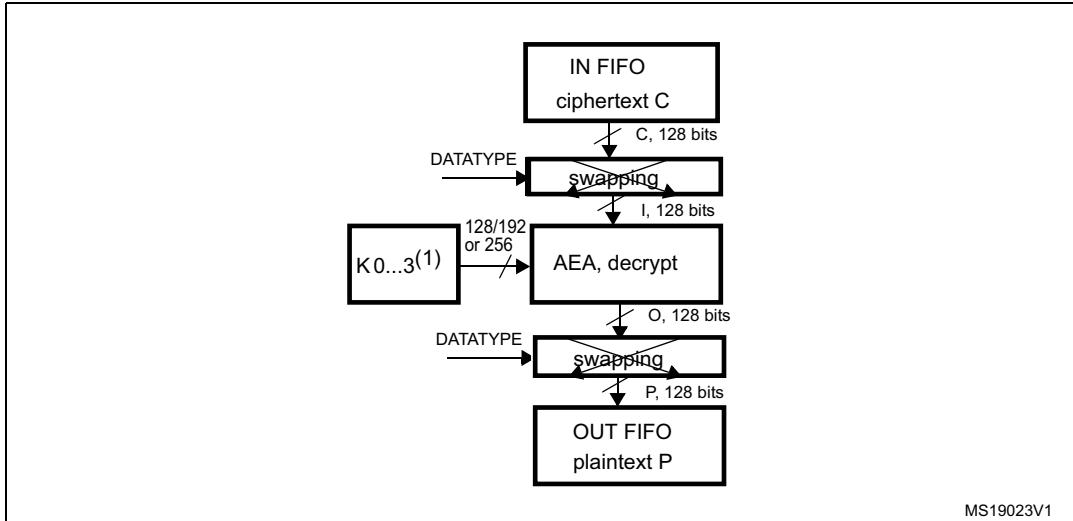
In this mode a 128-bit plaintext data block (P) is used after bit/byte/half-word swapping as the input block (I). The input block is processed through the AEA in the encrypt state using the 128, 192 or 256-bit key. The resultant 128-bit output block (O) is used after bit/byte/half-word swapping as ciphertext (C). It is then pushed into the OUT FIFO.

For more information on data swapping refer to [Section 20.4.16: CRYP data registers and data swapping](#).

### AES ECB decryption

*Figure 131* illustrates the AES Electronic codebook (AES-ECB) mode decryption. This mode is selected by programming ALGOMODE to 0x4 and ALGODIR to 1 in CRYP\_CR.

**Figure 131. AES-ECB mode decryption**



MS19023V1

1. K: key; C: cipher text; I: input block; O: output block; P: plain text.
2. If Key size = 128 => Key = [K3 K2]  
 If Key size = 192 => Key = [K3 K2 K1]  
 If Key size = 256 => Key = [K3 K2 K1 K0].

To perform an AES decryption in ECB mode, the secret key has to be prepared (it is necessary to execute the complete key schedule for encryption) by collecting the last round key, and using it as the first round key for the decryption of the ciphertext. This preparation phase is computed by the AES core. Refer to [Section 20.4.7: Preparing the CRYP AES key for decryption](#) for more details on how to prepare the key.

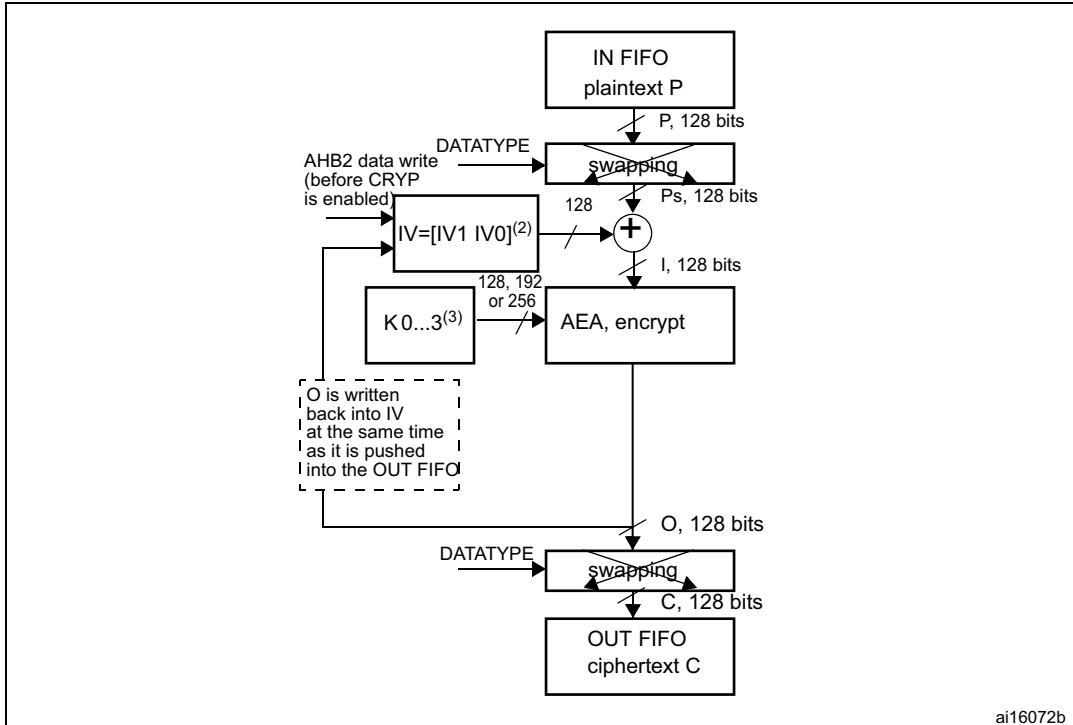
When the key preparation is complete, the decryption proceed as follows: a 128-bit ciphertext block (C) is used after bit/byte/half-word swapping as the input block (I). The keying sequence is reversed compared to that of the encryption process. The resultant 128-bit output block (O), after bit/byte or half-word swapping, produces the plaintext (P). The AES-CBC decryption process continues in this manner until the last complete ciphertext block has been decrypted.

For more information on data swapping refer to [Section 20.4.16: CRYP data registers and data swapping](#).

## AES CBC encryption

[Figure 132](#) illustrates the AES Cipher block chaining (AES-CBC) mode encryption. This mode is selected by writing ALGOMODE to 0x5 and ALGODIR to 0 in CRYP\_CR.

**Figure 132. AES-CBC mode encryption**



ai16072b

1. K: key; C: cipher text; I: input block; O: output block; Ps: plain text before swapping (when decoding) or after swapping (when encoding); P: plain text; IV: Initialization vectors.
2. IVx = [IVxR IVxL], R = right, L = left.
3. If Key size = 128 => Key = [K3 K2].  
If Key size = 192 => Key = [K3 K2 K1].  
If Key size = 256 => Key = [K3 K2 K1 K0].

In this mode the first input block ( $I_1$ ) obtained after bit/byte/half-word swapping is formed by exclusive-ORing the first plaintext data block ( $P_1$ ) with a 128-bit initialization vector IV ( $I_1 = IV \oplus P_1$ ). The input block is processed through the AEA in the encrypt state using the 128-, 192- or 256-bit key ( $K_0 \dots K_3$ ). The resultant 128-bit output block ( $O_1$ ) is used directly as ciphertext ( $C_1$ ), that is,  $C_1 = O_1$ . This first ciphertext block is then exclusive-ORED with the second plaintext data block to produce the second input block, ( $I_2$ ) = ( $C_1 \oplus P_2$ ). Note that  $I_2$  and  $P_2$  now refer to the second block. The second input block is processed through the AEA to produce the second ciphertext block. This encryption process continues to “chain” successive cipher and plaintext blocks together until the last plaintext block in the message is encrypted.

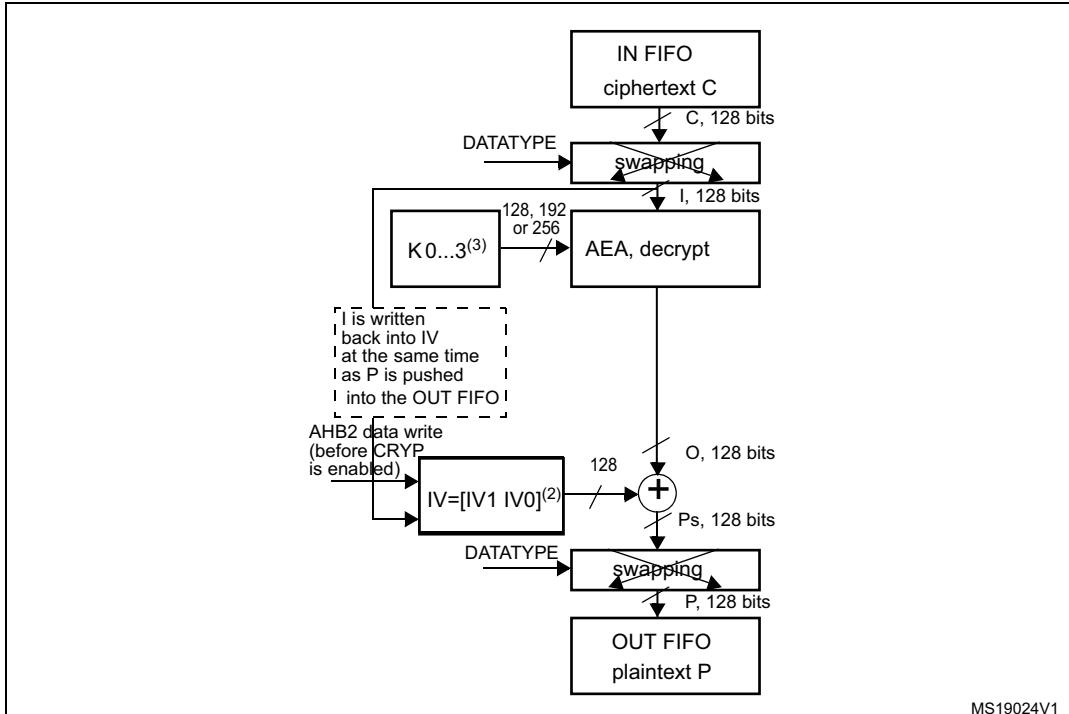
If the message does not consist of an integral number of data blocks, then the final partial data block should be encrypted in a manner specified for the application, as explained in [Section 20.4.8: CRYP stealing and data padding](#).

For more information on data swapping, refer to [Section 20.4.16: CRYP data registers and data swapping](#).

### AES CBC decryption

*Figure 133* illustrates the AES Cipher block chaining (AES-CBC) mode decryption. This mode is selected by writing ALGOMODE to 0x5 and ALGODIR to 1 in CRYP\_CR.

**Figure 133. AES-CBC mode decryption**



MS19024V1

1. K: key; C: cipher text; I: input block; O: output block; Ps: plain text before swapping (when decoding) or after swapping (when encoding); P: plain text; IV: Initialization vectors.
2. IVx = [IVxR IVxL], R = right, L = left.
3. If Key size = 128 => Key = [K3 K2].  
If Key size = 192 => Key = [K3 K2 K1].  
If Key size = 256 => Key = [K3 K2 K1 K0].

In CBC mode, like in ECB mode, the secret key must be prepared to perform an AES decryption. Refer to [Section 20.4.7: Preparing the CRYP AES key for decryption](#) for more details on how to prepare the key.

When the key preparation process is complete, the decryption proceeds as follows: the first 128-bit ciphertext block ( $C_1$ ) is used directly as the input block ( $I_1$ ). The input block is processed through the AEA in the decrypt state using the 128-, 192- or 256-bit key. The resulting output block is exclusive-ORED with the 128-bit initialization vector IV (which must be the same as that used during encryption) to produce the first plaintext block ( $P_1 = O_1 \oplus IV$ ).

The second ciphertext block is then used as the next input block and is processed through the AEA. The resulting output block is exclusive-ORED with the first ciphertext block to produce the second plaintext data block ( $P_2 = O_2 \oplus C_1$ ). Note that  $P_2$  and  $O_2$  refer to the second block of data. The AES-CBC decryption process continues in this manner until the last complete ciphertext block has been decrypted.

Ciphertext representing a partial data block must be decrypted in a manner specified for the application, as explained in [Section 20.4.8: CRYP stealing and data padding](#).

For more information on data swapping, refer to [Section 20.4.16: CRYP data registers and data swapping](#).

### AES suspend/resume operations in ECB/CBC modes

Before interrupting the current message, the user application must respect the following sequence:

1. If DMA is used, stop the DMA transfers to the IN FIFO by clearing to 0 the DIEN bit in the CRYP\_DMACR register.
2. Wait until both the IN and the OUT FIFOs are empty (IFEM = 1 and OFNE = 0 in the CRYP\_SR) and the BUSY bit is cleared.
3. If DMA is used, stop the DMA transfers from the OUT FIFO by clearing to 0 the DOEN bit in the CRYP\_DMACR register.
4. Disable the CRYP by setting the CRYPEN bit to 0 in CRYP\_CR, then save the current configuration (bits [9:2] in the CRYP\_CR register). If ECB mode is not selected, save the initialization vector registers, because CRYP\_IVx(L/R)R registers have changed from initial values during the data processing.

Note:

*Key registers do not need to be saved as the original key value is known by the application.*

5. If DMA is used, save the DMA controller status (such as pointers to IN and OUT data transfers, number of remaining bytes).

To resume message processing, the user application must respect the following sequence:

1. If DMA is used, reconfigure the DMA controller to complete the rest of the FIFO IN and FIFO OUT transfers.
2. Make sure the cryptographic processor is disabled by reading the CRYPEN bit in CRYP\_CR (it must be set to 0).
3. Configure the cryptographic processor again with the initial setting in CRYP\_CR, as well as the key registers using the saved configuration.
4. For AES-ECB or AES-CBC decryption, the key must be prepared again, as described in [Section 20.4.7: Preparing the CRYP AES key for decryption](#).
5. If ECB mode is not selected, restore CRYP\_IVx(L/R)R registers using the saved configuration.
6. Enable the cryptographic processor by setting the CRYPEN bit to 1.
7. If DMA is used, enable again the DMA requests from the cryptographic processor, by setting DIEN and DOEN bits to 1 in the CRYP\_DMACR register.

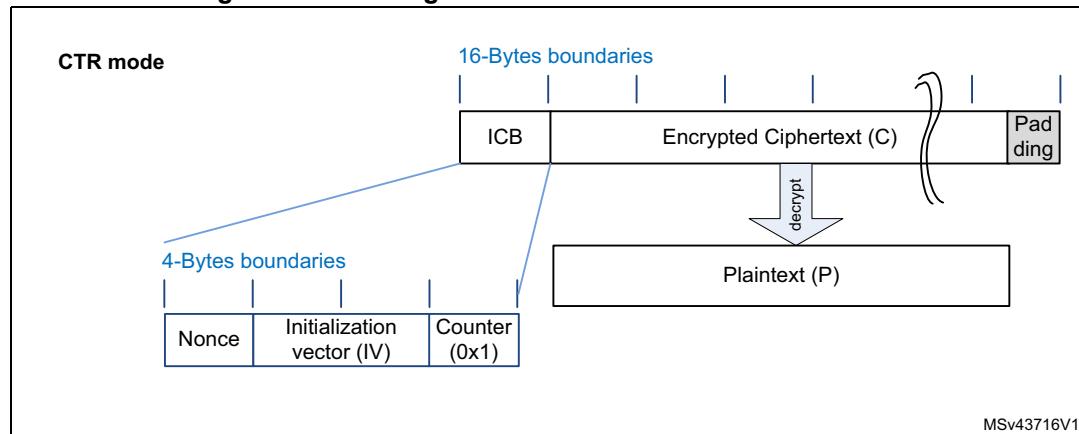
## 20.4.12 CRYP AES counter mode (AES-CTR)

### Overview

The AES counter mode (CTR) uses the AES block as a key stream generator. The generated keys are then XORed with the plaintext to obtain the ciphertext.

CTR chaining is defined in NIST Special Publication 800-38A, *Recommendation for Block Cipher Modes of Operation*. A typical message construction in CTR mode is given in [Figure 134](#).

**Figure 134. Message construction for the Counter mode**



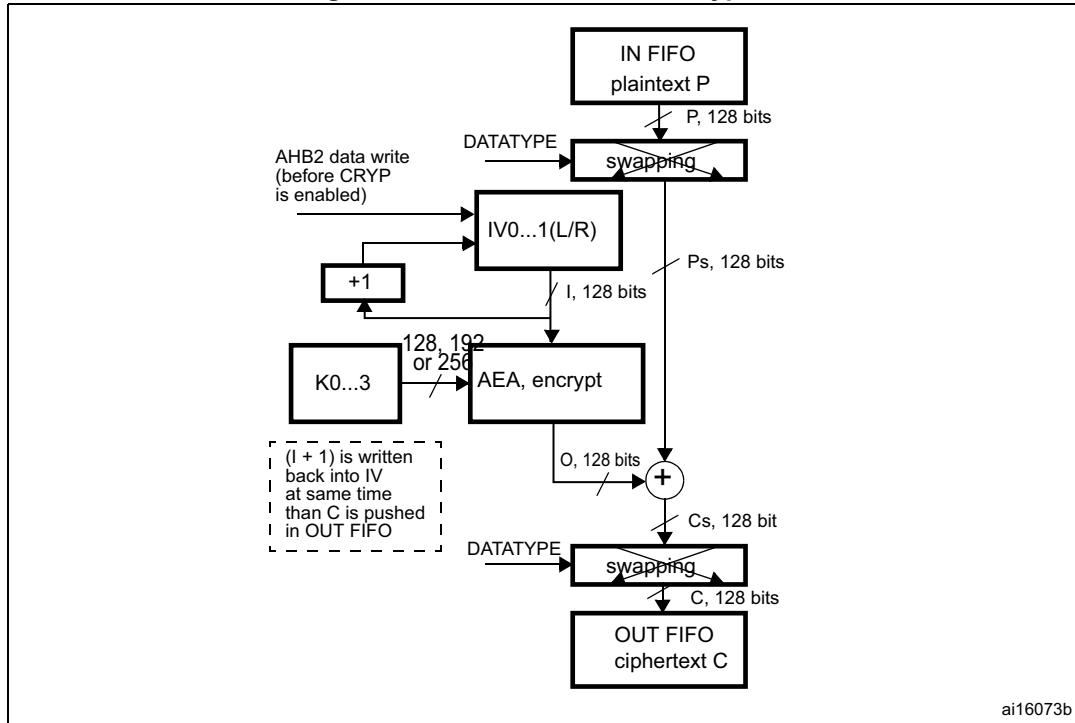
The structure of this message is as below:

- A 16-byte Initial Counter Block (ICB), composed of three distinct fields:
  - A *nonce*: a 32-bit, single-use value (i.e. a new nonce should be assigned to each new communication).
  - The *initialization vector* (IV): a 64-bit value that must be unique for each execution of the mode under a given key.
  - The *counter*: a 32-bit big-endian integer that is incremented each time a block has been processed. The initial value of the counter should be set to 1.
- The plaintext (P) is both authenticated and encrypted as ciphertext C, with a known length. This length can be non-multiple of 16 bytes, in which case a plaintext padding is required.

## AES CTR processing

*Figure 135* (respectively *Figure 136*) describes the AES-CTR encryption (respectively decryption) process implemented within this peripheral. This mode is selected by programming ALGOMODE bitfield to 0x6 in CRYP\_CR.

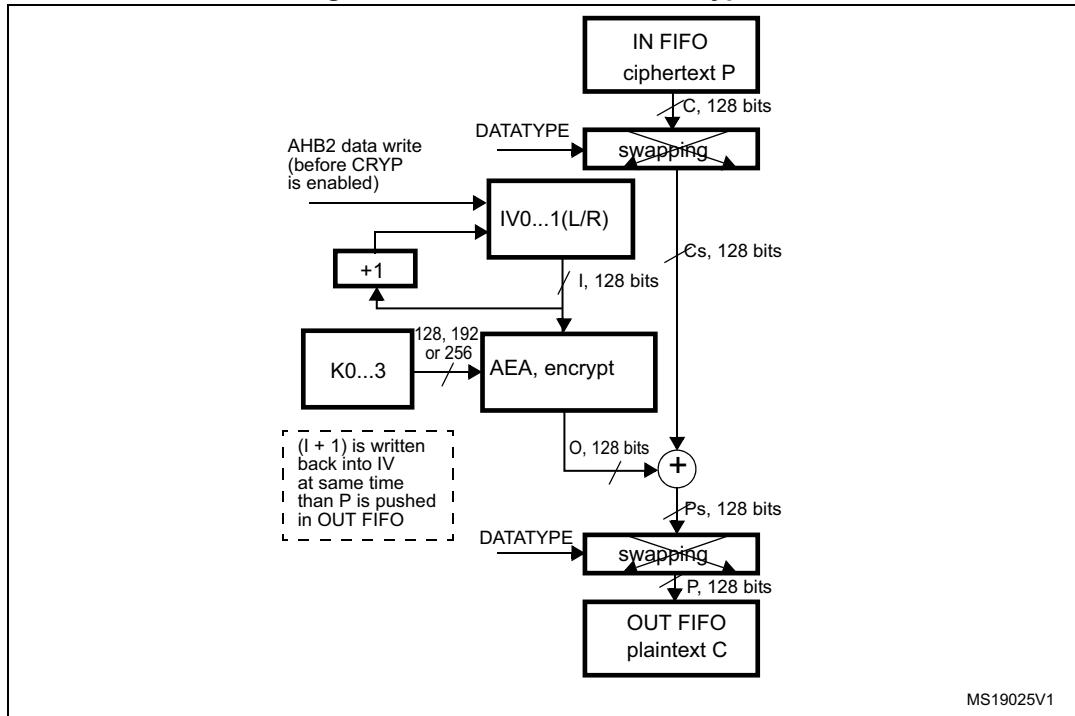
**Figure 135. AES-CTR mode encryption**



ai16073b

1. K: key; C: cipher text; I: input Block; o: output block; Ps: plain text before swapping (when decoding) or after swapping (when encoding); Cs: cipher text after swapping (when decoding) or before swapping (when encoding); P: plain text; IV: Initialization vectors.

Figure 136. AES-CTR mode decryption



1. K: key; C: cipher text; I: input Block; o: output block; Ps: plain text before swapping (when decoding) or after swapping (when encoding); Cs: cipher text after swapping (when decoding) or before swapping (when encoding); P: plain text; IV: Initialization vectors.

In CTR mode, the output block is XORed with the subsequent input block before it is input to the algorithm. Initialization vectors in the peripheral must be initialized as shown on [Table 126](#).

Table 126. Counter mode initialization vector

CRYP_IV1RR[31:0]	CRYP_IV1LR[31:0]	CRYP_IV0RR[31:0]	CRYP_IV0LR[31:0]
IVI[127:96]	IVI[95:64]	IVI[63:32]	IVI[31:0] 32-bit counter = 0x1

Unlike in CBC mode, which uses the CRYP\_IVx(L/R)R registers only once when processing the first data block, in CTR mode IV registers are used for processing each data block, and the peripheral increments the least significant 32 bits (leaving the other most significant 96 bits unchanged).

CTR decryption does not differ from CTR encryption, since the core always encrypts the current counter block to produce the key stream that will be XORed with the plaintext or cipher as input. Thus when ALGOMODE is set to 0x6, ALGODIR is don't care.

**Note:** *In this mode the key must NOT be prepared for decryption.*

The following sequence must be used to perform an encryption or a decryption in CTR chaining mode:

1. Make sure the cryptographic processor is disabled by clearing the CRYPEN bit in the CRYP\_CR register.
2. Configure CRYP\_CR as follows:
  - a) Program ALGOMODE bits to 0x6 to select CTR mode. ALGODIR can be set to any value.
  - b) Configure the data type (1, 8, 16 or 32 bits) through the DATATYPE bits.
  - c) Define the key length using KEYSIZE bits.
3. Initialize the key registers (128,192 or 256 bits) in CRYP\_KEYRx as well as the initialization vector (IV) as described in [Table 126](#).
4. Flush the IN and OUT FIFOs by writing the FFLUSH bit to 1 in the CRYP\_CR register.
5. If it is the last block, optionally pad the data with zeros to have a complete block.
6. Append data in the cryptographic processor and read the result. The three possible scenarios are described in [Section 20.4.5: CRYP procedure to perform a cipher operation](#).
7. Repeat the previous step until the second last block is processed. For the last block, execute the two previous steps. For this last block, the driver must discard the data that is not part of the data when the last block size is less than 16 bytes.

### Suspend/resume operations in CTR mode

Like for the CBC mode, it is possible to interrupt a message to send a higher priority message, and resume the message which was interrupted. Detailed CBC sequence can be found in [Section 20.4.11: CRYP AES basic chaining modes \(ECB, CBC\)](#).

*Note: Like for CBC mode, IV registers must be reloaded during the resume operation.*

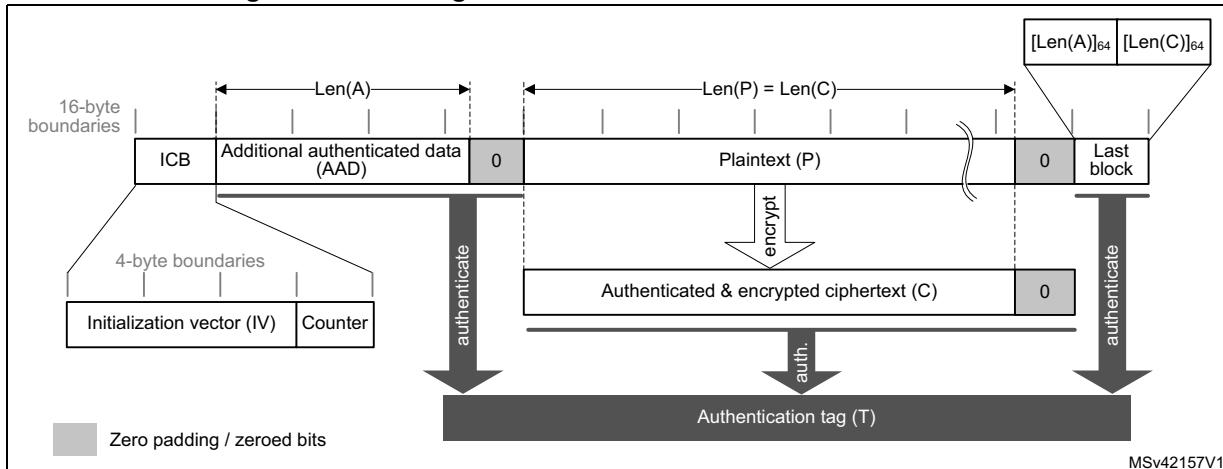
### 20.4.13 CRYP AES Galois/counter mode (GCM)

#### Overview

The AES Galois/counter mode (GCM) allows encrypting and authenticating the plaintext, and generating the correspondent ciphertext and tag (also known as message authentication code). To ensure confidentiality, GCM algorithm is based on AES counter mode. It uses a multiplier over a fixed finite field to generate the tag.

GCM chaining is defined in NIST Special Publication 800-38D, *Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) and GMAC*. A typical message construction in GCM mode is given in [Figure 137](#).

**Figure 137. Message construction for the Galois/counter mode**



The structure of this message is defined as below:

- A 16-byte Initial Counter Block (ICB), composed of two distinct fields:
  - The *initialization vector (IV)*: a 96-bit value that must be unique for each execution of the mode under a given key. Note that the GCM standard supports IV that are shorter than 96-bit, but in this case strict rules apply.
  - The *counter*: a 32-bit big-endian integer that is incremented each time a block has been processed. According to NIST specification, the counter value is 0x2 when processing the first block of payload.
- The authenticated header A (also known as Additional Authentication Data) has a known length *Len(A)* that can be non-multiple of 16 bytes and cannot exceed  $2^{64}-1$  bits. This part of the message is only authenticated, not encrypted.
- The plaintext message (P) is both authenticated and encrypted as ciphertext C, with a known length *Len(P)* that can be non-multiple of 16 bytes, and cannot exceed  $2^{32}-2$  blocks of 128-bits.

Note:

*GCM standard specifies that ciphertext C has same bit length as the plaintext P.*

- When a part of the message (AAD or P) has a length which is non-multiple of 16 bytes, a special padding scheme is required.
- The last block is composed of the length of A (on 64 bits) and the length of ciphertext C (on 64 bits) as shown in [Table 127](#).

**Table 127. GCM last block definition**

Endianness	Bit[0]	Bit[32]	Bit[64]	Bit[96]
Input data	0x0	Header length[31:0]	0x0	Payload length[31:0]

### AES GCM processing

This mode is selected by writing ALGOMODE bitfield to 0x8 in CRYP\_CR.

The mechanism for the confidentiality of the plaintext in GCM mode is a variation of the Counter mode, with a particular 32-bit incrementing function that generates the necessary sequence of counter blocks.

CRYP\_IVx(L/R)R registers are used for processing each data block. The cryptographic processor automatically increments the 32 least significant bits of the counter block. The first counter block (CB1) written by the application is equal to the Initial Counter Block incremented by one (see [Table 128](#)).

**Table 128. GCM mode IV registers initialization**

Register	CRYP_IV0LR[31:0]	CRYP_IV0RR[31:0]	CRYP_IV1LR[31:0]	CRYP_IV1RR[31:0]
Input data	ICB[127:96]	ICB[95:64]	ICB[63:32]	ICB[31:0] 32-bit counter = 0x2

**Note:**

*In this mode the key must NOT be prepared for decryption.*

The authentication mechanism in GCM mode is based on a hash function, called *GF2mul*, that performs multiplication by a fixed parameter, called the hash subkey (H), within a binary Galois field.

To process a GCM message, the driver must go through four phases, which are described in the following subsections.

- The Init phase: the peripheral prepares the GCM hash subkey (H) and performs the IV processing
- The Header phase: the peripheral processes the Additional Authenticated Data (AAD), with hash computation only.
- The Payload phase: the peripheral processes the plaintext (P) with hash computation, keystream encryption and data XORing. It operates in a similar way for ciphertext (C).
- The Final phase: the peripheral generates the authenticated tag (T) using the data last block.

## 1. GCM initialization phase

During this first step, the GCM hash subkey (H) is calculated and saved internally to be used for processing all the blocks. It is recommended to follow the sequence below:

- a) Make sure the cryptographic processor is disabled by clearing the CRYPEN bit in the CRYP\_CR register.
- b) Select the GCM chaining mode by programming ALGOMODE bits to 0x8 in CRYP\_CR. In the same register define the key length using KEYSIZE bits and the data type using DATATYPE bits.
- c) Configure GCM\_CCMPH bits to 0b00 in CRYP\_CR to indicate that the initialization phase is ongoing.
- d) Initialize the key registers (128, 192 or 256 bits) in CRYP\_KEYRx as well as the initialization vector (IV) as defined in [Table 128](#).
- e) Set CRYPEN bit to 1 to start the calculation of the hash key.
- f) Wait for the CRYPEN bit to be cleared to 0 by the cryptographic processor, before moving on to the next phase.

## 2. GCM header phase

The below sequence shall be performed after the GCM initialization phase. It must be complete before jumping to the payload phase. The sequence is identical for encryption and decryption.

- g) Set the GCM\_CCMPH bits to 0b01 in CRYP\_CR to indicate that the header phase is ongoing.
- h) Set the CRYPEN bit to 1 to start accepting data.
- i) If it is the last block of additional authenticated data, optionally pad the data with zeros to have a complete block.
- j) Append additional authenticated data in the cryptographic processor. The three possible scenarios are described in [Section 20.4.5: CRYP procedure to perform a cipher operation](#).
- k) Repeat the previous step until the second last additional authenticated data block is processed. For the last block, execute the two previous steps. Once all the additional authenticated data have been supplied, wait until the BUSY flag is cleared before moving on to the next phase.

*Note:* This phase can be skipped if there is no additional authenticated data, i.e.  $\text{Len}(A) = 0$ .

*In header and payload phases, CRYPEN bit is not automatically cleared by the cryptographic processor.*

### 3. GCM payload phase (encryption or decryption)

When the payload size is not null, this sequence must be executed after the GCM header phase. During this phase, the encrypted/decrypted payload is stored in the CRYP\_DOUT register.

- l) Set the CRYPEN bit to 0.
- m) Configure GCM\_CCMPH to 0b10 in the CRYP\_CR register to indicate that the payload phase is ongoing.
- n) Select the algorithm direction (0 for encryption, 1 for decryption) through the ALGODIR bit in CRYP\_CR.
- o) Set the CRYPEN bit to 1 to start accepting data.
- p) If it is the last block of cleartext or plaintext, optionally pad the data with zeros to have a complete block. For encryption, refer to [Section 20.4.8: CRYP stealing and data padding](#) for more details.
- q) Append payload data in the cryptographic processor, and read the result. The three possible scenarios are described in [Section 20.4.5: CRYP procedure to perform a cipher operation](#).
- r) Repeat the previous step until the second last plaintext block is encrypted or until the last block of ciphertext is decrypted. For the last block of plaintext (encryption only), execute the two previous steps. For the last block, the driver must discard the bits that are not part of the cleartext or the ciphertext when the last block size is less than 16 bytes. Once all payload data have been supplied, wait until the BUSY flag is cleared.

*Note:* This phase can be skipped if there is no payload data, i.e.  $\text{Len}(C) = 0$  (see GMAC mode).

### 4. GCM final phase

In this last step, the cryptographic processor generates the GCM authentication tag and stores it in CRYP\_DOUT register.

- s) Configure GCM\_CCMPH[1:0] to 0b11 in CRYP\_CR to indicate that the Final phase is ongoing. Set the ALGODIR bit to 0 in the same register.
- t) Write the input to the CRYP\_DIN register four times. The input must be composed of the length in bits of the additional authenticated data (coded on 64 bits) concatenated with the length in bits of the payload (coded of 64 bits), as show in [Table 127](#).

*Note:* In this final phase data have to be swapped according to the DATATYPE programmed in CRYP\_CR register.

- u) Wait until the OFNE flag (FIFO output not empty) is set to 1 in the CRYP\_SR register.
- v) Read the CRYP\_DOUT register four times: the output corresponds to the authentication tag.
- w) Disable the cryptographic processor (CRYPEN bit = 0 in CRYP\_CR)
- x) If an authenticated decryption is being performed, compare the generated tag with the expected tag passed with the message.

### Suspend/resume operations in GCM mode

Before interrupting the current message in header or payload phase, the user application must respect the following sequence:

1. If DMA is used, stop DMA transfers to the IN FIFO by clearing to 0 the DIEN bit in the CRYP\_DMACR register.
2. Wait until both the IN and the OUT FIFOs are empty (IFEM = 1 and OFNE = 0 in the CRYP\_SR register) and the BUSY bit is cleared.
3. If DMA is used, stop DMA transfers from the OUT FIFO by clearing to 0 the DOEN bit in the CRYP\_DMACR register.
4. Disable the cryptographic processor by setting the CRYPEN bit to 0 in CRYP\_CR, then save the current configuration (bits [9:2], bits [17:16] and bits 19 of the CRYP\_CR register). In addition, save the initialization vector registers, since CRYP\_IVx(L/R)R registers have changed from their initial values during data processing.

*Note:* *Key registers do not need to be saved as original their key value is known by the application.*

5. Save context swap registers: CRYP\_CSGCMCCM0..7R and CRYP\_CSGCM0..7R
6. If DMA is used, save the DMA controller status (pointers to IN and OUT data transfers, number of remaining bytes, etc.).

To resume message processing, the user must respect the following sequence:

1. If DMA is used, reconfigure the DMA controller to complete the rest of the FIFO IN and FIFO OUT transfers.
2. Make sure the cryptographic processor is disabled by reading the CRYPEN bit in CRYP\_CR (it must be 0).
3. Configure again the cryptographic processor with the initial setting in CRYP\_CR, as well as the key registers using the saved configuration.
4. Restore context swap registers: CRYP\_CSGCMCCM0..7R and CRYP\_CSGCM0..7R
5. Restore CRYP\_IVx(L/R)R registers using the saved configuration.
6. Enable the cryptographic processor by setting the CRYPEN bit to 1.
7. If DMA is used, enable again cryptographic processor DMA requests by setting to 1 the DIEN and DOEN bits in the CRYP\_DMACR register.

*Note:* *In Header phase, DMA OUT FIFO transfer is not used.*

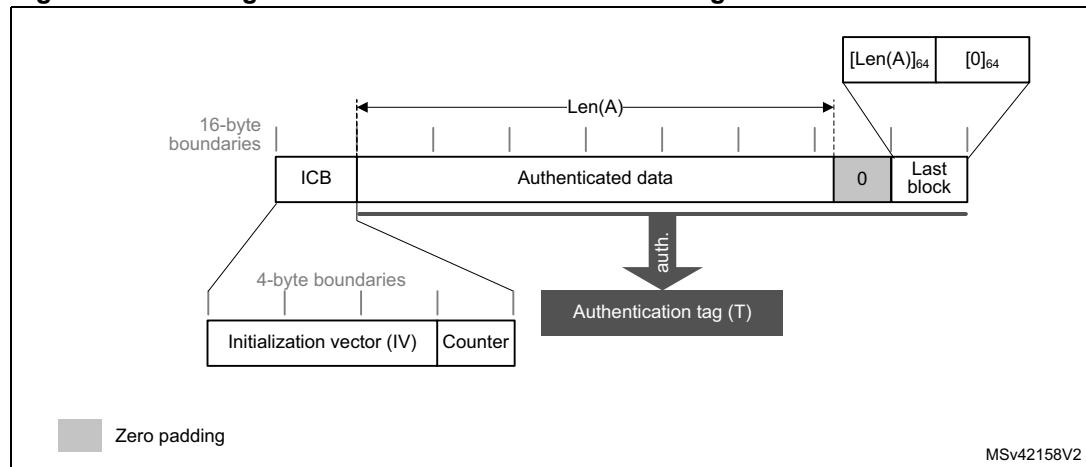
## 20.4.14 CRYP AES Galois message authentication code (GMAC)

### Overview

The Galois message authentication code (GMAC) allows authenticating a plaintext and generating the corresponding tag information (also known as message authentication code). It is based on GCM algorithm, as defined in NIST *Special Publication 800-38D, Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) and GMAC*.

A typical message construction in GMAC mode is given in [Figure 138](#).

**Figure 138. Message construction for the Galois Message Authentication Code mode**



### AES GMAC processing

This mode is selected by writing ALGOMODE bitfield to 0x8 in CRYP\_CR.

GMAC algorithm corresponds to the GCM algorithm applied on a message composed only of an header. As a consequence, all steps and settings are the same as in GCM mode, except that the payload phase (3) is not used.

### Suspend/resume operations in GMAC

GMAC is exactly the same as GCM algorithm except that only header phase (2) can be interrupted.

## 20.4.15 CRYP AES Counter with CBC-MAC (CCM)

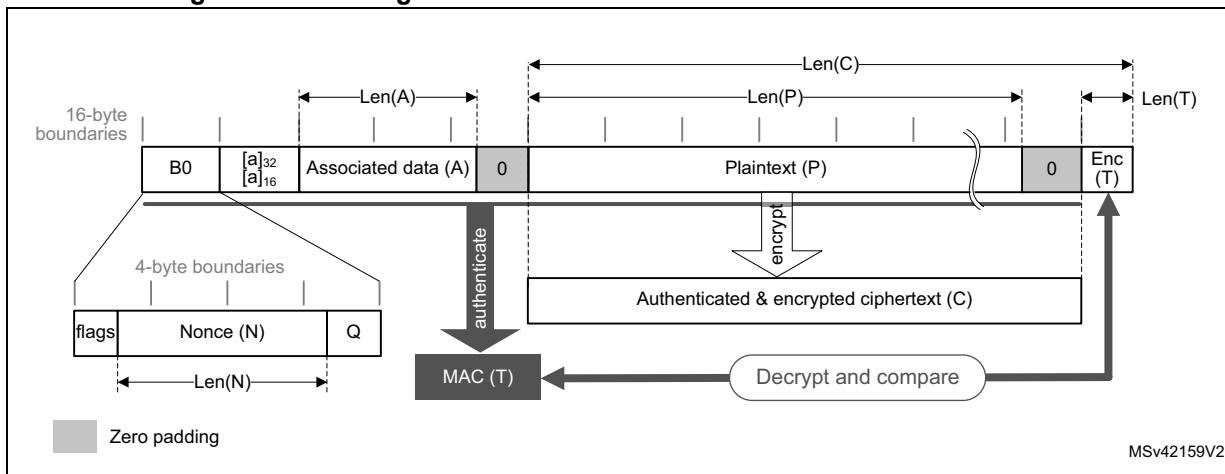
### Overview

The AES Counter with Cipher Block Chaining-Message Authentication Code (CCM) algorithm allows encrypting and authenticating the plaintext, and generating the correspondent ciphertext and tag (also known as message authentication code). To ensure confidentiality, CCM algorithm is based on AES counter mode. It uses Cipher Block Chaining technique to generate the message authentication code. This is commonly called CBC-MAC.

**Note:** *NIST does not approve this CBC-MAC as an authentication mode outside of the context of the CCM specification.*

CCM chaining is specified in NIST Special Publication 800-38C, *Recommendation for Block Cipher Modes of Operation - The CCM Mode for Authentication and Confidentiality*. A typical message construction in CCM mode is given in [Figure 139](#)

**Figure 139. Message construction for the Counter with CBC-MAC mode**



The structure of this message is as below:

- One 16-byte first authentication block (called B0 by the standard), composed of three distinct fields:
  - Q: a bit string representation of the byte length of P (Plen)
  - A *nonce* (N): single-use value (i.e. a new nonce should be assigned to each new communication). Size of nonce  $Nlen$  + size of  $Plen$  shall be equal to 15 bytes.
  - Flags: most significant byte containing four flags for control information, as specified by the standard. It contains two 3-bit strings to encode the values t (MAC length expressed in bytes) and q (plaintext length such as  $Plen < 2^{8q}$  bytes). Note that the counter blocks range associated to q is equal to  $2^{8q-4}$ , i.e. if q maximum value is 8, the counter blocks used in cipher shall be on 60 bits.

**Note:** *The cryptographic peripheral can only manage padded plaintext/ciphertext messages of length  $Plen < 2^{36} + 1$  bytes.*

- 16-bytes blocks (B) associated to the Associated Data (A).  
This part of the message is only authenticated, not encrypted. This section has a known length, *ALen*, that can be a non-multiple of 16 bytes (see [Figure 139](#)). The standard also states that, on the MSB bits of the first message block (B1), the associated data length expressed in bytes (a) must be encoded as defined below:
  - If  $0 < a < 2^{16}-2^8$ , then it is encoded as [a]<sub>16</sub>, i.e. two bytes.
  - If  $2^{16}-2^8 < a < 2^{32}$ , then it is encoded as 0xff || 0xfe || [a]<sub>32</sub>, i.e. six bytes.
  - If  $2^{32} < a < 2^{64}$ , then it is encoded as 0xff || 0xff || [a]<sub>64</sub>, i.e. ten bytes.
- 16-byte blocks (B) associated to the plaintext message (P), which is both authenticated and encrypted as ciphertext C, with a known length of *Plen*. This length can be a non-multiple of 16 bytes (see [Figure 139](#)) but cannot exceed  $2^{32}$  blocks of 128-bit.
- The encrypted MAC (T) of length *Tlen* appended to the ciphertext C of overall length *Clen*.
- When a part of the message (A or P) has a length which is a non-multiple of 16 bytes, a special padding scheme is required.

*Note:* *CCM chaining mode can also be used with associated data only (i.e. no payload).*

As an example, the C.1 section in *NIST Special Publication 800-38C* gives the following:

```

N: 10111213 141516 (Nlen = 56 bits or 0x7 bytes)
A: 00010203 04050607 (Alen = 64 bits or 0x8 bytes)
P: 20212223 (Plen = 32 bits i.e. Q = 0x4 bytes)
T: 6084341b (Tlen = 32 bits or t = 4)

B0: 4f101112 13141516 00000000 00000004
B1: 00080001 02030405 06070000 00000000
B2: 20212223 00000000 00000000 00000000
CTR0: 0710111213 141516 00000000 00000000
CTR1: 0710111213 141516 00000000 00000001
  
```

The usage of control blocks CTRx is explained in the following section. The generation of CTR0 from the first block (B0) must be managed by software.

## AES CCM processing

This mode is selected by writing ALGOMODE bitfield to 0x9 in CRYP\_CR.

The data input to the generation-encryption process are a valid nonce, a valid payload string, and a valid associated data string, all properly formatted. The CBC chaining mechanism is applied to the formatted data to generate a MAC, whose length is known. Counter mode encryption, which requires a sufficiently long sequence of counter blocks as input, is applied to the payload string and separately to the MAC. The resulting data, called the ciphertext C, is the output of the generation-encryption process on plaintext P.

CRYP\_IVx(L/R)R registers are used for processing each data block. The cryptographic processor automatically increments the CTR counter with a bit length defined by the first block (B0). The first counter written by application, CTR1, is equal to B0 with the first 5 bits zeroed and the most significant bits containing P byte length also zeroed, then incremented by one (see [Table 129](#)).

**Table 129. CCM mode IV registers initialization**

Register	CRYP_IV0LR[31:0]	CRYP_IV0RR[31:0]	CRYP_IV1LR[31:0]	CRYP_IV1RR[31:0]
Input data	B0[127:96], where the 5 most significant bits are set to 0 (flag bits)	B0[95:64]	B0[63:32]	B0[31:0], where Q length bits are set to 0, except for bit 0 that is set to 1

**Note:** *In this mode, the key must NOT be prepared for decryption.*

To process a CCM message, the driver must go through four phases, which are described below.

- The **Initialization phase**: the peripheral processes the first block and prepares the first counter block.
- The **Header phase**: the peripheral processes the Associated data (A), with hash computation only.
- The **Payload phase**: the peripheral processes the plaintext (P), with hash computation, counter block encryption and data XORing. It operates in a similar way for ciphertext (C).
- The **Final phase**: the peripheral generates the message authentication code (MAC).

## 1. CCM initialization phase

In this first step, the first block (B0) of the CCM message is programmed into the CRYP\_DIN register. During this phase, the CRYP\_DOUT register does not contain any output data. It is recommended to follow the sequence below:

- a) Make sure that the cryptographic processor is disabled by clearing the CRYPEN bit in the CRYP\_CR register.
- b) Select the CCM chaining mode by programming the ALGOMODE bits to 0x9 in the CRYP\_CR register. In the same register define the key length using KEYSIZE bits and the data type using DATATYPE bits.
- c) Configure the GCM\_CCMPH bits to 0b00 in CRYP\_CR to indicate that we are in the initialization phase.
- d) Initialize the key registers (128, 192 or 256 bits) in CRYP\_KEYRx as well as the initialization vector (IV) with CTR1 information, as defined in [Table 129](#).
- e) Set the CRYPEN bit to 1 in CRYP\_CR to start accepting data.
- f) Write the B0 packet into CRYP\_DIN register, then wait for the CRYPEN bit to be cleared to 0 by the cryptographic processor before moving on to the next phase.

*Note:* *In this initialization phase data have to be swapped according to the DATATYPE programmed in CRYP\_CR register.*

## 2. CCM header phase

The below sequence shall be performed after the CCM initialization phase. It must be complete before jumping to the payload phase. The sequence is identical for encryption and decryption. During this phase, the CRYP\_DOUT register does not contain any output data.

- g) Set the GCM\_CCMPH bit to 0b01 in CRYP\_CR to indicate that the header phase is ongoing.
- h) Set the CRYPEN bit to 1 to start accepting data.
- i) If it is the last block of associated data, optionally pad the data with zeros to have a complete block.
- j) Append the associated data in the cryptographic processor. The three possible scenarios are described in [Section 20.4.5: CRYP procedure to perform a cipher operation](#).
- k) Repeat the previous step until the second last associated data block is processed. For the last block, execute the two previous steps. Once all the additional authenticated data have been supplied, wait until the BUSY flag is cleared.

*Note:* *This phase can be skipped if there is no associated data (Alen = 0).*

*The first block of the associated data B1 must be formatted with the associated data length. This task must be managed by the driver.*

### 3. CCM payload phase (encryption or decryption)

When the payload size is not null, this sequence must be performed after the CCM header phase. During this phase, the encrypted/decrypted payload is stored in the CRYP\_DOUT register.

- l) Set the CRYPEN bit to 0.
- m) Configure GCM\_CCMPH bits to 0b10 in CRYP\_CR to indicate that the payload phase is ongoing.
- n) Select the algorithm direction (0 for encryption, 1 for decryption) through the ALGODIR bit in CRYP\_CR.
- o) Set the CRYPEN bit to 1 to start accepting data.
- p) If it is the last block of cleartext, optionally pad the data with zeros to have a complete block (encryption only). For decryption, refer to [Section 20.4.8: CRYP stealing and data padding](#) for more details.
- q) Append payload data in the cryptographic processor, and read the result. The three possible scenarios are described in [Section 20.4.5: CRYP procedure to perform a cipher operation](#).
- r) Repeat the previous step until the second last plaintext block is encrypted or until the last block of ciphertext is decrypted. For the last block of plaintext (encryption only), execute the two previous steps. For the last block of ciphertext (decryption only), the driver must discard the data that is not part of the cleartext when the last block size is less than 16 bytes. Once all payload data have been supplied, wait until the BUSY flag is cleared

*Note:* This phase can be skipped if there is no payload data, i.e. Plen = 0 or Clen = Tlen

*Note:* Do not forget to remove  $LSB_{Tlen}(C)$  encrypted tag information when decrypting ciphertext C.

### 4. CCM final phase

In this last step, the cryptographic processor generates the CCM authentication tag and stores it in the CRYP\_DOUT register.

- s) Configure GCM\_CCMPH[1:0] bits to 0b11 in CRYP\_CR to indicate that the final phase is ongoing and set the ALGODIR bit to 0 in the same register.
- t) Load in CRYP\_DIN, the CTR0 information which is described in [Table 129](#) with bit[0] set to 0.

*Note:* In this final phase, data have to be swapped according to the DATATYPE programmed in CRYP\_CR register.

- u) Wait until the OFNE flag (FIFO output not empty) is set to 1 in the CRYP\_SR register.
- v) Read the CRYP\_DOUT register four times: the output corresponds to the encrypted CCM tag.
- w) Disable the cryptographic processor (CRYPEN bit set to 0 in CRYP\_CR)
- x) If an authenticated decryption is being performed, compare the generated encrypted tag with the encrypted tag padded in the ciphertext, i.e.  $LSB_{Tlen}(C) = MSB_{Tlen}(CRYP\_DOUT\ data)$ .

### Suspend/resume operations in CCM mode

Before interrupting the current message in payload phase, the user application must respect the following sequence:

1. If DMA is used, stop the DMA transfers to the IN FIFO by clearing to 0 the DIEN bit in the CRYP\_DMACR register.
2. Wait until both the IN and the OUT FIFOs are empty (IFEM = 1 and OFNE = 0 in the CRYP\_SR register) and the BUSY bit is cleared.
3. If DMA is used, stop the DMA transfers from the OUT FIFO by clearing to 0 the DOEN bit in the CRYP\_DMACR register.
4. Disable the cryptographic processor by setting the CRYPEN bit to 0 in CRYP\_CR, then save the current configuration (bits [9:2], bits [17:16] and bits 19 in the CRYP\_CR register). In addition, save the initialization vector registers, since CRYP\_IVx(L/R)R registers have changed from their initial values during the data processing.

*Note:* *Key registers do not need to be saved as their original key value is known by the application.*

5. Save context swap registers: CRYP\_CSGCMCCM0..7R
6. If DMA is used, save the DMA controller status (pointers for IN and OUT data transfers, number of remaining bytes, etc.).

To resume message processing, the user application must respect the following sequence:

1. If DMA is used, reconfigure the DMA controller to complete the rest of the FIFO IN and FIFO OUT transfers.
2. Make sure the cryptographic processor is disabled by reading the CRYPEN bit in CRYP\_CR (must be 0).
3. Configure the cryptographic processor again with the initial setting in CRYP\_CR and key registers using the saved configuration.
4. Restore context swap registers: CRYP\_CSGCMCCM0..7R
5. Restore CRYP\_IVx(L/R)R registers using the saved configuration.
6. Enable the cryptographic processor by setting the CRYPEN bit to 1.
7. If DMA is used, enable again cryptographic processor DMA requests by setting to 1 the DIEN and DOEN bits in the CRYP\_DMACR register.

*Note:* *In Header phase DMA OUT FIFO transfer is not used.*

#### 20.4.16 CRYP data registers and data swapping

The CRYP\_DIN register is the 32-bit wide data input register of the peripheral. It is used to enter into the input FIFO up to four 64-bit blocks (TDES) or two 128-bit blocks (AES) of plaintext (when encrypting) or ciphertext (when decrypting), one 32-bit word at a time.

The four adjacent (respectively two) words of the AES (respectively DES/TDES) data block are organized in big-endian order, with the most significant byte of a word on the lowest address.

The cryptographic accelerator can be configured to perform a bit, byte, half-word, or no swapping on the input data word in the CRYP\_DINR register, before loading it to the CRYP processing core, and on the data output from the CRYP processing core, before sending it

to the CRYP\_DOUTR register. The choice depends on the type of data. For example, a byte swapping is used for an ASCII text stream.

The data swap type is selected through the DATATYPE[1:0] bitfield of the AES\_CR register. The selection applies both to the input and the output of the CRYP processing core.

**Note:** *The CRYP Key registers (CRYP\_Kx(L/R)) and initialization registers (CRYP\_IVx(L/R)) are not sensitive to the swap mode selected.*

More information on data input and data swapping can be found in the next subsections.

### DES/TDES data input and output

A 64-bit data block is entered into the cryptographic processor with two successive 32-bit word write operations to the CRYP\_DINR register (DIN[31:0] bitfield), the most significant word (bits [64:33]) first, the least significant word (bits [32:1]) last.

A 64-bit data block is retrieved from the cryptographic processor with two successive 32-bit word read operations from the CRYP\_DOUTR register (DOUT[31:0] bitfield), the most significant word (bits [64:33]) first, the least significant word (bits [32:1]) last.

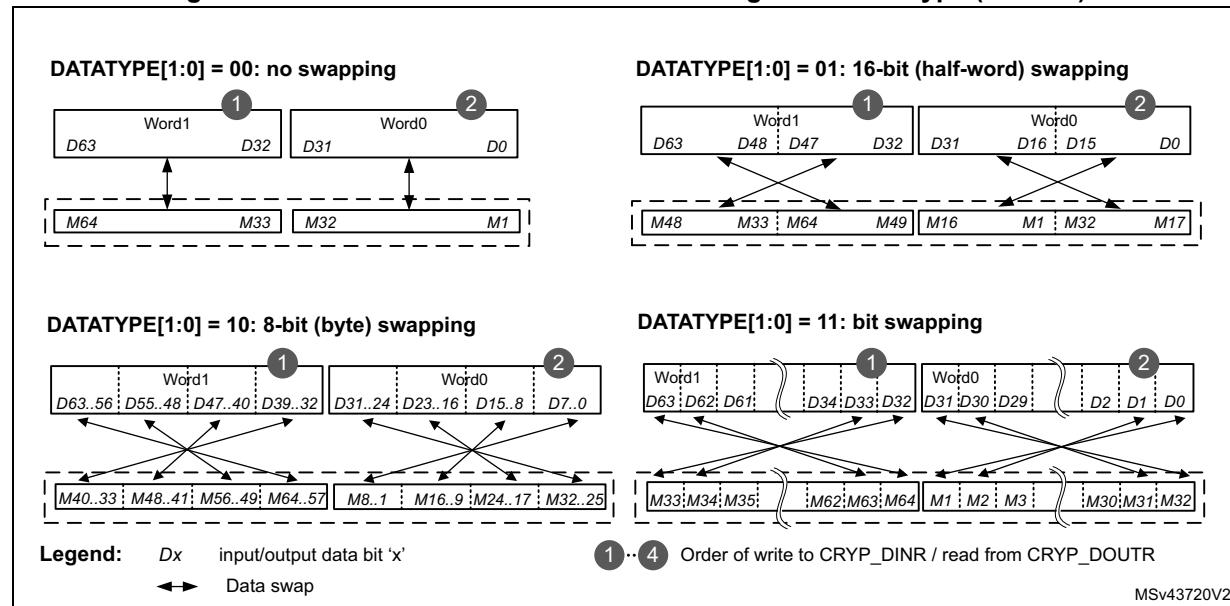
### DES/TDES data swapping feature

The cryptographic processor data swapping feature for DES/TDES is summarized in [Table 130](#) and [Figure 140](#).

**Table 130. DES/TDES data swapping example**

DATATYPE in CRYP_CR	Swapping performed	Data block representation (64-bit)	
		System memory data (big-endian)	
00	No swapping	Block[64..1]: 0xABCD7720 6973FE01 Address @, word[63..32]: 0xABCD 7720 Address @+4, word[31..0]: 0x6973 FE01	
01	Half-word (16-bit) swapping	Block[64..1]: 0xA <u>B</u> CD 7720 <b>69</b> 73 FE01 Address @, word[63..32]: 0x7720 <b>A</b> BCD Address @+4, word[31..0]: 0x <u>F</u> E01 <b>69</b> 73	
10	Byte (8-bit) swapping	Block[64..1]: 0x <u>A</u> B CD <b>77</b> 20 <b>69</b> 73 <b>FE</b> 01 Address @, word[63..32]: 0x20 <u>77</u> CD <u>A</u> B Address @+4, word[31..0]: 0x01 <u>F</u> E 73 <u>69</u>	
11	Bit swapping in 32-bit word	Block[64..33]: 0xABCD7720 1010 1011 1100 1101 0111 0111 0010 0000 Block[32..1]: 0x6973FE01 0110 1001 0111 0011 1111 1110 0000 0001 Address @, word[63..32]: 0x04EEB3D5 0000 0100 1110 1110 1011 0011 1101 0101 Address @+4, word[31..0]: 0x807FCE96 1000 0000 0111 1111 1100 1110 1001 0110	

Figure 140. 64-bit block construction according to the data type (IN FIFO)



### AES data input and output

A 128-bit data block is entered into the cryptographic processor with four successive 32-bit word writes into the CRYP\_DINR register (bitfield DIN[31:0]), the most significant word (bits [127:96]) first, the least significant word (bits [31:0]) last.

A 128-bit data block is retrieved from the cryptographic processor with four successive 32-bit word reads from the CRYP\_DOUTR register (bitfield DOUT[31:0]), the most significant word (bits [127:96]) first, the least significant word (bits [31:0]) last.

### AES data swapping feature

The cryptographic processor data swapping feature for AES is summarized in [Table 131](#) and [Figure 141](#).

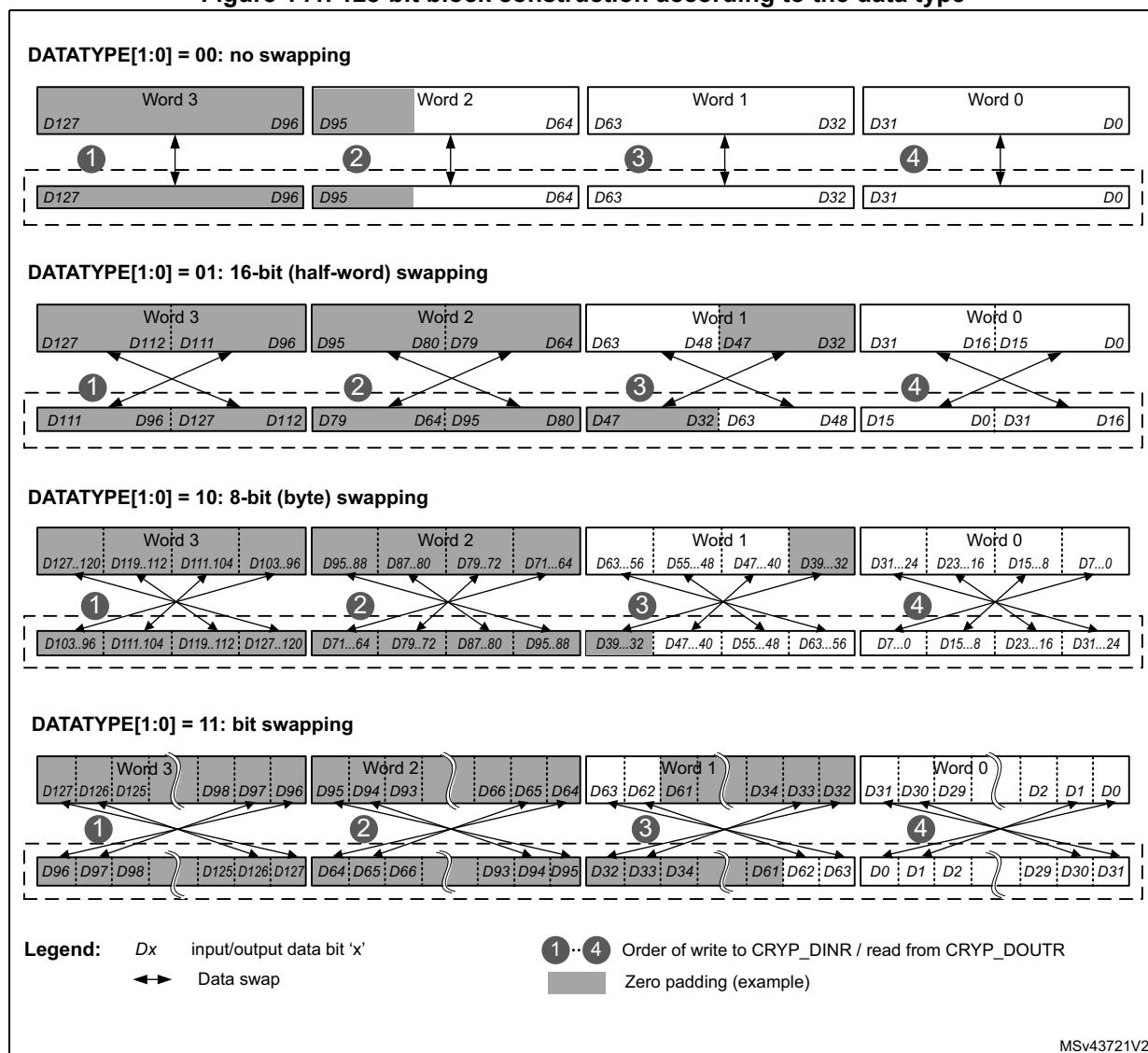
Table 131. AES data swapping example

DATATYPE in CRYP_CR	Swapping performed	First half data block (64-bit)
		System memory data (big-endian)
00	No swapping	Block[63..0]: 0x4E6F7720 69732074
		Address @, word[63..32]: 0x4E6F7720 Address @+4, word[31..0]: 0x69732074
01	Half-word (16-bit) swapping	Block[63..0]: 0x <b>4E6F</b> 7720 <b>6973</b> 2074
		Address @, word[63..32]: 0x7720 <b>4E6F</b> Address @+4, word[31..0]: 0x2074 <b>6973</b>
10	Byte (8-bit) swapping	Block[63..0]: 0x <b>4E</b> 6F 77 20 <b>69</b> 73 <b>20</b> 74
		Address @, word[63..32]: 0x2077 6F <b>4E</b> Address @+4, word[31..0]: 0x74 <b>20</b> 7369

Table 131. AES data swapping example (continued)

DATATYPE in CRYP_CR	Swapping performed	First half data block (64-bit)
		System memory data (big-endian)
11	Bit swapping	<p>Block[63..32]: 0x4E6F7720            0100 1110 0110 1111 0111 0111 0010 0000</p> <p>Block[31..0]: 0x69732074            0110 1001 0111 0011 0010 0000 0111 0100</p> <p>Address @, word[63..32]: 0x04EE F672            0000 0100 1110 1110 1111 0110 0111 0010</p> <p>Address @+4, word[31..0]: 0x2E04 CE96            0010 1110 0000 0100 1100 1110 1001 0110</p>

Figure 141. 128-bit block construction according to the data type



### 20.4.17 CRYP key registers

The CRYP\_Kx registers are write-only registers used to store the encryption or decryption keys. They are organized as four 64-bit registers, as shown in [Table 132](#) and [Table 133](#).

**Note:** *In memory and in CRYP key registers, AES and DES/TDES keys are stored in big-endian format, with most significant byte on the lowest address.*

**Table 132. Key endianness in CRYP\_KxR/LR registers (AES 128/192/256-bit keys)**

K0LR[31:0]	K0RR[31:0]	K1LR[31:0]	K1RR[31:0]	K2LR[31:0]	K2RR[31:0]	K3LR[31:0]	K3RR[31:0]
-	-	-	-	k[127:96]	k[95:64]	k[63:32]	k[31:0]
K0LR[31:0]	K0RR[31:0]	K1LR[31:0]	K1RR[31:0]	K2LR[31:0]	K2RR[31:0]	K3LR[31:0]	K3RR[31:0]
-	-	k[191:160]	k[159:128]	k[127:96]	k[95:64]	k[63:32]	k[31:0]
K0LR[31:0]	K0RR[31:0]	K1LR[31:0]	K1RR[31:0]	K2LR[31:0]	K2RR[31:0]	K3LR[31:0]	K3RR[31:0]
k[255:224]	k[223:192]	k[191:160]	k[159:128]	k[127:96]	k[95:64]	k[63:32]	k[31:0]

**Table 133. Key endianness in CRYP\_KxR/LR registers (DES K1 and TDES K1/2/3)**

K0LR[31:0]	K0RR[31:0]	K1LR[31:0]	K1RR[31:0]	K2LR[31:0]	K2RR[31:0]	K3LR[31:0]	K3RR[31:0]
-	-	K1[64:33]	K1[32:1]	-	-	-	-
-	-	K1[64:33]	K1[32:1]	K2[64:33]	K2[32:1]	K3[64:33]	K3[32:1]

As shown on [Table 133](#), when TDES is selected (ALGOMODE[2:0] = 000 or 001) a 3-key vector (K1, K2, and K3) is used. When DES is selected (ALGOMODE[2:0] = 010 or 011) only 1-key vector (K1) is used.

**Note:** *DES/TDES keys include 8-bit parity information that are not used by the cryptographic processor. In other words, bits 8, 16, 24, 32, 40, 48, 56 and 64 of each 64-bit key value Kx[1:64] are not used.*

Write operations to the CRYP\_Kx(L/R) registers when the BUSY bit is set to 1 in the CRYP\_SR register are ignored (which means that the register content is not modified). The application must thus check that the BUSY bit is cleared to 0 before modifying key registers.

Key registers are not affected by the data swapping controlled by DATATYPE value in CRYP\_CR register.

Refer to [Section 20.7: CRYP registers](#) for a detailed description of CRYP\_Kx(L/R) registers.

### 20.4.18 CRYP initialization vector registers

The CRYP\_IVxL/IVxR registers are used to store the initialization vector or the nonce, depending on the chaining mode selected. When used, these registers are updated by the core after each computation round of the TDES or AES core.

They are organized as four 64-bit registers, as shown in [Table 134](#) and [Table 135](#). In DES/TDES mode only CRYP\_IV0x are used.

**Note:** *In memory and in CRYP IV registers, AES and DES/TDES initialization vectors are stored in big-endian format, with most significant byte on the lowest address.*

**Table 134. Initialization vector endianness in CRYP\_IVx(L/R)R registers (AES)**

CRYP_IV0LR[31:0]	CRYP_IV0RR[31:0]	CRYP_IV1LR[31:0]	CRYP_IV1RR[31:0]
IVI[127:96]	IVI[95:64]	IVI[63:32]	IVI[31:0]

**Table 135. Initialization vector endianness in CRYP\_IVx(L/R)R registers (DES/TDES)**

CRYP_IV0LR[31:0]	CRYP_IV0RR[31:0]	CRYP_IV1LR[31:0]	CRYP_IV1RR[31:0]
IVI[64:32]	IVI[31:0]	-	-

Any write operation to the CRYP\_IV0...1(L/R)R registers when the BUSY bit is set to 1 in the CRYP\_SR register is disregarded (which means that register content not modified). The software must thus check that the BUSY bit is cleared to 0 in the CRYP\_SR register before modifying initialization vectors.

Reading the CRYP\_IV0...1(L/R)R register returns the latest counter value (useful for managing suspend mode) except for CCM/GCM.

The initialization vector registers are not affected by the data swapping feature controlled by DATATYPE value in CRYP\_CR register.

Refer to [Section 20.7: CRYP registers](#) for a detailed description of CRYP\_IVxL/IVxR registers.

#### 20.4.19 CRYP DMA interface

The cryptographic processor provides an interface to connect to the DMA (Direct Memory Access) controller. The DMA operation is controlled through the CRYP DMA control register (CRYP\_DMACR).

##### Data input using DMA

DMA can be enabled for writing data into the cryptographic peripheral by setting the DIEN bit in the CRYP\_DMACR register. When this bit is set, the cryptographic processor initiates a DMA request during the INPUT phase each time it requires a word to be written to the CRYP\_DIN register.

[Table 136](#) shows the recommended configuration to transfer data from memory to cryptographic processor through the DMA controller.

**Table 136. Cryptographic processor configuration for memory-to-peripheral DMA transfers**

DMA channel control register field	Programming recommendation
Transfer size	Message length, multiple of four 32-bit words. This 128-bit granularity corresponds to two blocks for DES/TDES, one block for AES. According to the algorithm and the mode selected, special padding/ciphertext stealing might be required. Refer to <a href="#">Section 20.4.8: CRYP stealing and data padding</a> for details.
Source burst size (memory)	CRYP FIFO_size /2 /transfer_width = 4

**Table 136. Cryptographic processor configuration for memory-to-peripheral DMA transfers (continued)**

DMA channel control register field	Programming recommendation
Destination burst size (peripheral)	CRYP FIFO_size /2 /transfer_width = <b>4</b> (FIFO_size = 8x32-bit, transfer_width = 32-bit)
DMA FIFO size	CRYP FIFO_size /2 = 16 bytes
Source transfer width (memory)	32-bit words
Destination transfer width (peripheral)	32-bit words
Source address increment (memory)	Yes, after each 32-bit transfer.
Destination address increment (peripheral)	Fixed address of CRYP_DIN shall be used (no increment).

### Data output using DMA

To enable the DMA for reading data from CRYP processor, set the DOEN bit in the CRYP\_DMACR register. When this bit is set, the cryptographic processor initiates a DMA request during the OUTPUT phase each time it requires a word to be read from the CRYP\_DOUT register.

*Table 137* shows the recommended configuration to transfer data from cryptographic processor to memory through the DMA controller.

**Table 137. Cryptographic processor configuration for peripheral-to-memory DMA transfers**

DMA channel control register field	Programming recommendation
Transfer size	Message length, multiple of four 32-bit words. This 128-bit granularity corresponds to two blocks for DES/TDES, one block for AES. Depending on the chaining mode selected, extra bits have to be discarded.
Source burst size (peripheral)	When <i>DES</i> is used: Single transfer (burst size = <b>1</b> ) When <i>AES</i> is used: CRYP FIFO_size /2 /transfer_width = <b>4</b> (FIFO_size = 8x32-bit, transfer_width = 32-bit)
Destination burst size (memory)	CRYP FIFO_size /2 /transfer_width = <b>4</b>
DMA FIFO size	CRYP FIFO_size /2 = 16 bytes
Source transfer width (peripheral)	32-bit words
memory transfer width (memory)	32-bit words

**Table 137. Cryptographic processor configuration for peripheral-to-memory DMA transfers (continued)**

DMA channel control register field	Programming recommendation
Source address increment (peripheral)	Fixed address of CRYP_DOUT shall be used (no increment).
Destination address increment (memory)	Yes, after each 32-bit transfer.

### DMA mode

When AES is used, the cryptographic processor manages two DMA transfer requests through `cryp_in_dma` and `cryp_out_dma` internal input/output signals, which are asserted:

- for IN FIFO: every time a block has been read from FIFO by CRYP,
- for OUT FIFO: every time a block has been written into the FIFO by the cryptographic processor.

When DES/TDES is used, the cryptographic processor manages two DMA transfer requests through `cryp_in_dma` and `cryp_out_dma` internal input/output signals, which are asserted:

- for IN FIFO: every time two blocks have been read from FIFO by the cryptographic processor

for OUT FIFO: every time a word has been written into the FIFO by the cryptographic processor (single transfer). Note that a burst transfer is also triggered when two blocks have been written into the FIFO.

All request signals are deasserted if the cryptographic peripheral is disabled or the DMA enable bit is cleared (DIEN bit for the IN FIFO and DOEN bit for the OUT FIFO in the CRYP\_DMACR register).

**Caution:** It is important that DMA controller empties the cryptographic peripheral output FIFO before filling up the CRYP input FIFO. To achieve it, the DMA controller should be configured so that the transfer from the peripheral to the memory has a higher priority than the transfer from the memory to the peripheral.

For more detailed information on DMA operations, refer to [Section 20.4.5: CRYP procedure to perform a cipher operation](#).

### 20.4.20 CRYP error management

No error flags are generated by the cryptographic processor.

## 20.5 CRYP interrupts

### Overview

There are two individual maskable interrupt sources generated by the cryptographic processor to signal the following events:

- Input FIFO empty or not full
- Output FIFO full or not empty

These two sources are combined into a single interrupt signal which is the only interrupt signal from the CRYP processor that drives the Cortex® CPU interrupt controller. You can enable or disable CRYP interrupt sources individually by changing the mask bits in the CRYP\_IMSCR register. Setting the appropriate mask bit to 1 enables the interrupt.

The status of the individual maskable interrupt sources can be read either from the CRYP\_RISR register, for raw interrupt status, or from the CRYP\_MISR register for masked interrupt status. The status of the individual source of event flags can be read from the CRYP\_SR register.

*Table 138* gives a summary of the available features.

**Table 138. CRYP interrupt requests**

Interrupt acronym	Interrupt event	Event flag		Enable bit	Interrupt clear method
		in CRYP_xISR <sup>(1)</sup>	in CRYP_SR		
CRYP	Output FIFO full	OUTRIS (not masked) OUTMIS (masked)	OFFU	OUTIM <sup>(2)</sup> and CRYPEN	Read one data from output FIFO
	Output FIFO not empty		OFNE		Read all data from output FIFO
	Input FIFO not full	INRIS (not masked) INMIS (masked)	IFNF	INIM <sup>(2)</sup> and CRYPEN	Write data until input FIFO is full
	Input FIFO empty		IFEM		Write at least one word in input FIFO

1. The flags belong to CRYP\_RISR registers (unmasked or raw) or CRYP\_MISR registers (masked).

2. The flags belong to CRYP\_IMSCR register.

### Output FIFO service interrupt - OUTMIS

The output FIFO service interrupt is asserted when there is one or more (32-bit word) data items in the output FIFO. This interrupt is cleared by reading data from the output FIFO until there is no valid (32-bit) word left (that is when the interrupt follows the state of the output FIFO not empty flag OFNE).

The output FIFO service interrupt OUTMIS is NOT enabled with the CRYP processor enable bit. Consequently, disabling the CRYP processor does not force the OUTMIS signal low if the output FIFO is not empty.

### Input FIFO service interrupt - INMIS

The input FIFO service interrupt is asserted when there are less than four words in the input FIFO. It is cleared by performing write operations to the input FIFO until it holds four or more words.

The input FIFO service interrupt INMIS is enabled with the CRYP enable bit. Consequently, when CRYP is disabled, the INMIS signal is low even if the input FIFO is empty.

## 20.6 CRYP processing time

The time required to process a block for each mode of operation is summarized below. The block size is 128 bits for AES and 64 bits for DES/TDES.

**Table 139. Processing latency for ECB, CBC and CTR**

Key size	Operating modes	Chaining algorithm	Clock cycles
64 bits	DES encryption or decryption	ECD, CBC	16
3x64 bits	TDES encryption or decryption		48
128 bits	AES encryption or decryption <sup>(1)</sup>	ECD, CBC, CTR	14
	AES key preparation	-	12
192 bits	AES encryption or decryption <sup>(1)</sup>	ECD, CBC, CTR	16
	AES key preparation	-	14
256 bits	AES encryption or decryption <sup>(1)</sup>	ECD, CBC, CTR	18
	AES key preparation	-	16

1. Excluding key preparation time (ECB and CBC only).

**Table 140. Processing time (in clock cycle) for GCM and CCM per 128-bit block**

Key size	Operating modes	Chaining algorithm	Initialization phase	Header phase	Payload phase	Tag phase
128 bits		GCM	24	10	14	14
		CCM	12	14	25	14
192 bits	AES encryption or decryption	GCM	28	10	16	16
		CCM	14	16	29	16
256 bits		GCM	32	10	18	18
		CCM	16	18	33	18

## 20.7 CRYP registers

The cryptographic core is associated with several control and status registers, eight key registers and four initialization vectors registers.

### 20.7.1 CRYP control register (CRYP\_CR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ALGOM ODE3	Res.	GCM_CCMPH [1:0]	
												rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRYPEN	FFLUSH	Res.	Res.	Res.	Res.	KEYSIZE[1:0]	DATATYPE[1:0]	ALGOMODE[2:0]				ALGODIR	Res.	Res.	
rw	rw					rw	rw	rw	rw	rw	rw	rw	rw		

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

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

Bit 18 Reserved, must be kept at reset value.

Bits 17:16 **GCM\_CCMPH[1:0]**: GCM or CCM Phase selection

This bitfield has no effect if GCM, GMAC or CCM algorithm is not selected in ALGOMODE field.

00: Initialization phase

01: Header phase

10: Payload phase

11: Final phase

Writing to GCM\_CCMPH bits while BUSY = 1 has no effect.

Bit 15 **CRYPEN**: CRYP processor Enable

0: Cryptographic processor peripheral is disabled

1: Cryptographic processor peripheral is enabled

This bit is automatically cleared by hardware when the key preparation process ends (ALGOMODE = 0111) or after GCM/GMAC or CCM Initialization phase.

Bit 14 **FFLUSH**: CRYP FIFO Flush

0: No FIFO flush

1: FIFO flush enabled

When CRYPEN = 0, writing this bit to 1 flushes the IN and OUT FIFOs (that is read and write pointers of the FIFOs are reset). Writing this bit to 0 has no effect. When CRYPEN = 1, writing this bit to 0 or 1 has no effect.

Reading this bit always returns 0.

FFLUSH bit has to be set only when BUSY = 0. If not, the FIFO is flushed, but the block being processed may be pushed into the output FIFO just after the flush operation, resulting in a non-empty FIFO condition.

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

**Bits 9:8 KEYSIZE[1:0]: Key size selection (AES mode only)**

This bitfield defines the bit-length of the key used for the AES cryptographic core. This bitfield is ‘don’t care’ in the DES or TDES modes.

00: 128-bit key length

01: 192-bit key length

10: 256-bit key length

11: Reserved, do not use this value

Writing KEYSIZE bits while BUSY = 1 has no effect.

**Bits 7:6 DATATYPE[1:0]: Data Type selection**

This bitfield defines the format of data written in CRYP\_DIN or read from CRYP\_DOUT registers. For more details refer to [Section 20.4.16: CRYP data registers and data swapping](#).

00: 32-bit data. No swapping for each word. First word pushed into the IN FIFO (or popped off the OUT FIFO) forms bits 1...32 of the data block, the second word forms bits 33...64 etc.

01: 16-bit data, or half-word. Each word pushed into the IN FIFO (or popped off the OUT FIFO) is considered as 2 half-words, which are swapped with each other.

10: 8-bit data, or bytes. Each word pushed into the IN FIFO (or popped off the OUT FIFO) is considered as 4 bytes, which are swapped with each other.

11: bit data, or bit-string. Each word pushed into the IN FIFO (or popped off the OUT FIFO) is considered as 32 bits (1st bit of the string at position 0), which are swapped with each other.

Writing DATATYPE bits while BUSY = 1 has no effect.

**Bits 19, 5:3 ALGOMODE[3:0]: Algorithm mode**

Below definition includes the bit 19:

0000: TDES-ECB (triple-DES Electronic Codebook).

0001: TDES-CBC (triple-DES Cipher Block Chaining).

0010: DES-ECB (simple DES Electronic Codebook).

0011: DES-CBC (simple DES Cipher Block Chaining).

0100: AES-ECB (AES Electronic Codebook).

0101: AES-CBC (AES Cipher Block Chaining).

0110: AES-CTR (AES Counter mode).

0111: AES key preparation for ECB or CBC decryption.

1000: AES-GCM (Galois Counter mode) and AES-GMAC (Galois Message Authentication Code mode).

1001: AES-CCM (Counter with CBC-MAC).

Writing ALGOMODE bits while BUSY = 1 has no effect.

Others: Reserved, must not be used

**Bit 2 ALGODIR: Algorithm Direction**

0: Encrypt

1: Decrypt

Writing ALGODIR bit while BUSY = 1 has no effect.

Bits 1:0 Reserved, must be kept at reset value.

## 20.7.2 CRYP status register (CRYP\_SR)

Address offset: 0x04

Reset value: 0x0000 0003

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.	BUSY	OFFU	OFNE	IFNF	IFEM										
											r	r	r	r	r

Bits 31:5 Reserved, must be kept at reset value.

### Bit 4 **BUSY**: Busy bit

0: The CRYP core is not processing any data. The reason is:

- either that the CRYP core is disabled (CRYPPEN = 0 in the CRYP\_CR register) and the last processing has completed,
- or the CRYP core is waiting for enough data in the input FIFO or enough free space in the output FIFO (that is in each case at least 2 words in the DES, 4 words in the AES).

1: The CRYP core is currently processing a block of data or a key preparation is ongoing (AES ECB or CBC decryption only).

### Bit 3 **OFFU**: Output FIFO full flag

0: Output FIFO is not full

1: Output FIFO is full

### Bit 2 **OFNE**: Output FIFO not empty flag

0: Output FIFO is empty

1: Output FIFO is not empty

### Bit 1 **IFNF**: Input FIFO not full flag

0: Input FIFO is full

1: Input FIFO is not full

### Bit 0 **IFEM**: Input FIFO empty flag

0: Input FIFO is not empty

1: Input FIFO is empty

### 20.7.3 CRYP data input register (CRYP\_DIN)

Address offset: 0x08

Reset value: 0x0000 0000

The CRYP\_DIN register is the data input register. It is 32-bit wide. It is used to enter into the input FIFO up to four 64-bit blocks (TDES) or two 128-bit blocks (AES) of plaintext (when encrypting) or ciphertext (when decrypting), one 32-bit word at a time.

To fit different data sizes, the data can be swapped after processing by configuring the DATATYPE bits in the CRYP\_CR register. Refer to [Section 20.4.16: CRYP data registers and data swapping](#) for more details.

When CRYP\_DIN register is written to the data are pushed into the input FIFO.

- If CRYPTEN = 1, when at least four 32-bit words in the AES mode have been pushed into the input FIFO (two words in the DES/TDES mode), and when at least four words are free in the output FIFO (two words in the DES/TDES mode), the CRYP engine starts an encrypting or decrypting process.

When CRYP\_DIN register is read:

- If CRYPTEN = 0, the FIFO is popped, and then the data present in the Input FIFO are returned, from the oldest one (first reading) to the newest one (last reading). The IFEM flag must be checked before each read operation to make sure that the FIFO is not empty.
- If CRYPTEN = 1, an undefined value is returned.

**Note:** After the CRYP\_DIN register has been read once or several times, the FIFO must be flushed by setting the FFLUSH bit prior to processing new data.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATAIN[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
DATAIN[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DATAIN[31:0]: Data input**

On read FIFO is popped (last written value is returned), and its value is returned if CRYPTEN = 0. If CRYPTEN = 1 DATAIN register returns an undefined value.

On write current register content is pushed inside the FIFO.

### 20.7.4 CRYP data output register (CRYP\_DOUT)

Address offset: 0x0C

Reset value: 0x0000 0000

The CRYP\_DOUT register is the data output register. It is read-only and 32-bit wide. It is used to retrieve from the output FIFO up to four 64-bit blocks (TDES) or two 128-bit blocks

(AES) of plaintext (when encrypting) or ciphertext (when decrypting), one 32-bit word at a time.

To fit different data sizes, the data can be swapped after processing by configuring the DATATYPE bits in the CRYP\_CR register. Refer to [Section 20.4.16: CRYP data registers and data swapping](#) for more details.

When CRYP\_DOUT register is read, the last data entered into the output FIFO (pointed to by the read pointer) is returned.

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

Bits 31:0 **DATAOUT[31:0]: Data output**

On read returns output FIFO content (pointed to by read pointer), else returns an undefined value.

On write, no effect.

## 20.7.5 CRYP DMA control register (CRYP\_DMCR)

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.	DOEN	DIEN													
														rw	rw

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

Bit 1 **DOEN: DMA output enable**

When this bit is set, DMA requests are automatically generated by the peripheral during the output data phase.

0: Outgoing data transfer from CRYP via DMA is disabled

1: Outgoing data transfer from CRYP via DMA is enabled

Bit 0 **DIEN: DMA input enable**

When this bit is set, DMA requests are automatically generated by the peripheral during the input data phase.

0: Incoming data transfer to CRYP via DMA is disabled

1: Incoming data transfer to CRYP via DMA is enabled

## 20.7.6 CRYP interrupt mask set/clear register (CRYP\_IMSCR)

Address offset: 0x14

Reset value: 0x0000 0000

The CRYP\_IMSCR register is the interrupt mask set or clear register. It is a read/write register. When a read operation is performed, this register gives the current value of the mask applied to the relevant interrupt. Writing 1 to the particular bit sets the mask, thus enabling the interrupt to be read. Writing 0 to this bit clears the corresponding mask. All the bits are cleared to 0 when the peripheral is reset.

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.	OUTIM	INIM													
														rw	rw

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

Bit 1 **OUTIM**: Output FIFO service interrupt mask

- 0: Output FIFO service interrupt is masked
- 1: Output FIFO service interrupt is not masked

Bit 0 **INIM**: Input FIFO service interrupt mask

- 0: Input FIFO service interrupt is masked
- 1: Input FIFO service interrupt is not masked

## 20.7.7 CRYP raw interrupt status register (CRYP\_RISR)

Address offset: 0x18

Reset value: 0x0000 0001

The CRYP\_RISR register is the raw interrupt status register. It is a read-only register. When a read operation is performed, this register gives the current raw status of the corresponding interrupt, i.e. the interrupt information without taking CRYP\_IMSCR mask into account. Write operations have no effect.

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.	OUTRIS	INRIS													
														r	r

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

Bit 1 **OUTRIS**: Output FIFO service raw interrupt status

This bit gives the output FIFO interrupt information without taking CRYP\_IMSCR corresponding mask into account.

- 0: Raw interrupt not pending
- 1: Raw interrupt pending

Bit 0 **INRIS:** Input FIFO service raw interrupt status

This bit gives the input FIFO interrupt information without taking CRYP\_IMSCR corresponding mask into account.

- 0: Raw interrupt not pending
- 1: Raw interrupt pending

## 20.7.8 CRYP masked interrupt status register (CRYP\_MISR)

Address offset: 0x1C

Reset value: 0x0000 0000

The CRYP\_MISR register is the masked interrupt status register. It is a read-only register. When a read operation is performed, this register gives the current masked status of the corresponding interrupt, i.e. the interrupt information taking CRYP\_IMSCR mask into account. Write operations have no effect.

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.	OUTMIS	INMIS													
														r	r

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

Bit 1 **OUTMIS:** Output FIFO service masked interrupt status

This bit gives the output FIFO interrupt information taking into account the corresponding CRYP\_IMSCR mask.

- 0: Output FIFO interrupt not pending
- 1: Output FIFO interrupt pending

Bit 0 **INMIS:** Input FIFO service masked interrupt status

This bit gives the input FIFO interrupt information taking into account the corresponding CRYP\_IMSCR mask.

- 0: Input FIFO interrupt not pending
- 1: Input FIFO interrupt pending when CRYPTEN = 1

### 20.7.9 CRYP key register 0L (CRYP\_K0LR)

Address offset: 0x20

Reset value: 0x0000 0000

CRYP key registers contain the cryptographic keys.

For more information refer to [Section 20.4.17: CRYP key registers](#).

**Note:** *Write accesses to these registers are disregarded when the cryptographic processor is busy (bit BUSY = 1 in the CRYP\_SR register).*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
K[255:240]															
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
K[239:224]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **K[255:224]**: Key bit x (x = 255 to 224)

This write-only bitfield contains the bits [255:224] of the AES encryption or decryption key, depending on the operating mode. This register is not used in DES/TDES mode.

### 20.7.10 CRYP key register 0R (CRYP\_K0RR)

Address offset: 0x24

Reset value: 0x0000 0000

Refer to [Section 20.7.9: CRYP key register 0L \(CRYP\\_K0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
K[223:208]															
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
K[207:192]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **K[223:192]**: Key bit x (x = 223 to 192)

This write-only bitfield contains the bits [223:192] of the AES encryption or decryption key, depending on the operating mode. This register is not used in DES/TDES mode.

### 20.7.11 CRYP key register 1L (CRYP\_K1LR)

Address offset: 0x28

Reset value: 0x0000 0000

Refer to [Section 20.7.9: CRYP key register 0L \(CRYP\\_K0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
K[191:176]															
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
K[175:160]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **K[191:160]**: Key bit x (x = 191 to 160)

This write-only bitfield contains the bits [191:160] of the AES encryption or decryption key, depending on the operating mode. In DES/TDES mode this bitfield contains the bits [64:33] of the key K1, with parity bits unused.

### 20.7.12 CRYP key register 1R (CRYP\_K1RR)

Address offset: 0x2C

Reset value: 0x0000 0000

Refer to [Section 20.7.9: CRYP key register 0L \(CRYP\\_K0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
K[159:144]															
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
K[143:128]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **K[159:128]**: Key bit x (x = 159 to 128)

This write-only bitfield contains the bits [159:128] of the AES encryption or decryption key, depending on the operating mode. In DES/TDES mode this bitfield contains the bits [32:1] of the key K1, with parity bits unused.

### 20.7.13 CRYP key register 2L (CRYP\_K2LR)

Address offset: 0x30

Reset value: 0x0000 0000

Refer to [Section 20.7.9: CRYP key register 0L \(CRYP\\_K0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
K[127:112]															
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
K[111:96]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **K[127:96]**: Key bit x (x = 127 to 96)

This write-only bitfield contains the bits [127:96] of the AES encryption or decryption key, depending on the operating mode. In DES/TDES mode this bitfield contains the bits [64:33] of the key K2, with parity bits unused.

### 20.7.14 CRYP key register 2R (CRYP\_K2RR)

Address offset: 0x34

Reset value: 0x0000 0000

Refer to [Section 20.7.9: CRYP key register 0L \(CRYP\\_K0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
K[95:80]															
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
K[79:64]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **K[95:64]**: Key bit x (x = 95 to 64)

This write-only bitfield contains the bits [95:64] of the AES encryption or decryption key, depending on the operating mode. In DES/TDES mode this bitfield contains the bits [32:1] of the key K2, with parity bits unused.

### 20.7.15 CRYP key register 3L (CRYP\_K3LR)

Address offset: 0x38

Reset value: 0x0000 0000

Refer to [Section 20.7.9: CRYP key register 0L \(CRYP\\_K0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
K[63:48]															
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
K[47:32]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **K[63:32]**: Key bit x (x = 63 to 32)

This write-only bitfield contains the bits [63:32] of the AES encryption or decryption key, depending on the operating mode. In DES/TDES mode this bitfield contains the bits [64:33] of the key K3, with parity bits unused.

### 20.7.16 CRYP key register 3R (CRYP\_K3RR)

Address offset: 0x3C

Reset value: 0x0000 0000

Refer to [Section 20.7.9: CRYP key register 0L \(CRYP\\_K0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
K[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
K[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **K[31:0]**: Key bit x (x = 31 to 0)

This write-only bitfield contains the bits [31:0] of the AES encryption or decryption key, depending on the operating mode. In DES/TDES mode this bitfield contains the bits [32:1] of the key K3, with parity bits unused.

### 20.7.17 CRYP initialization vector register 0L (CRYP\_IV0LR)

Address offset: 0x40

Reset value: 0x0000 0000

The CRYP\_IVx(L/R)R registers store the initialization vector or the nonce, depending on the chaining mode selected. The size of the IV data is 64 bits for DES/TDES and 128 bits for AES. For more information refer to [Section 20.4.18: CRYP initialization vector registers](#).

**Note:** *Write accesses to these registers are disregarded when the cryptographic processor is busy (BUSY = 1 in the CRYP\_SR register).*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IVI[127:112]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IVI[111:96]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IVI[127:96]**: Initialization vector bit x (x = 127 to 96)

This bitfield stores the initialization vector bits [127:96] for AES chaining modes other than ECB. In DES/TDES mode it corresponds to IVI bits [63:32].

IV registers are updated by the core after each computation round of the DES/TDES or AES core.

### 20.7.18 CRYP initialization vector register 0R (CRYP\_IV0RR)

Address offset: 0x44

Reset value: 0x0000 0000

Refer to [Section 20.7.17: CRYP initialization vector register 0L \(CRYP\\_IV0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IVI[95:80]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IVI[79:64]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IVI[95:64]**: Initialization vector bit x (x = 95 to 64)

This bitfield stores the initialization vector bits [95:64] for AES chaining modes other than ECB. In DES/TDES mode it corresponds to IVI bits [31:0].

IV registers are updated by the core after each computation round of the DES/TDES or AES core.

### 20.7.19 CRYP initialization vector register 1L (CRYP\_IV1LR)

Address offset: 0x48

Reset value: 0x0000 0000

Refer to [Section 20.7.17: CRYP initialization vector register 0L \(CRYP\\_IV0LR\)](#) for details.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IVI[63:48]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IVI[47:32]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IVI[63:32]**: Initialization vector bit x (x = 63 to 32)

This bitfield stores the initialization vector bits [63:32] for AES chaining modes other than ECB. *This register is not used in DES mode.*

IV registers are updated by the core after each computation round of the AES core.

### 20.7.20 CRYP initialization vector register 1R (CRYP\_IV1RR)

Address offset: 0x4C

Reset value: 0x0000 0000

Refer to [Section 20.7.17: CRYP initialization vector register 0L \(CRYP\\_IV0LR\)](#) for details.

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

Bits 31:0 **IVI[31:0]**: Initialization vector bit x (x = 31 to 0)

This bitfield stores the initialization vector bits [31:0] for AES chaining modes other than ECB. *This register is not used in DES mode.*

IV registers are updated by the core after each computation round of the AES core.

### 20.7.21 CRYP register map

Table 141. CRYP register map and reset values

Offset	Register name reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
0x00	CRYP_CR	Res.	ALGOMODE[3]	Res.	GCM_CCMFH	CRYOPEN	FFFLUSH	KEYSIZE								
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 141. CRYP register map and reset values (continued)

Offset	Register name reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x04	CRYP_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																																
0x08	CRYP_DIN	DATAIN																															
	Reset value	0	0	0	0	0	0	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	CRYP_DOUT	DATAOUT																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x10	CRYP_DMACR	Res.	Res.	Res.	Res.	Res.	Res.	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	CRYP_IMSCR	Res.	Res.	Res.	Res.	Res.	Res.	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	CRYP_RISR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																
0x1C	CRYP_MISR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																
0x20	CRYP_K0LR	K[255:224]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x24	CRYP_K0RR	K[223:192]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
...																																	
0x38	CRYP_K3LR	K[63:32]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x3C	CRYP_K3RR	K[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		
0x40	CRYP_IV0LR	IVI[127:96]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x44	CRYP_IV0RR	IVI[95:64]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x48	CRYP_IV1LR	IVI[63:32]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x4C	CRYP_IV1RR	IVI[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 21 Hash processor (HASH)

This section applies to the whole STM32F756xx and STM32F750xx devices, unless otherwise specified.

### 21.1 Introduction

The hash processor is a fully compliant implementation of the secure hash algorithm (SHA-1, SHA2-224, SHA2-256), the MD5 (message-digest algorithm 5) hash algorithm and the HMAC (keyed-hash message authentication code) algorithm. HMAC is suitable for applications requiring message authentication.

The hash processor computes FIPS (Federal Information Processing Standards) approved digests of length of 160, 224, 256 bits, for messages of up to  $(2^{64} - 1)$  bits. It also computes 128-bit digests for the MD5 algorithm.

### 21.2 HASH main features

- Suitable for data authentication applications, compliant with:
  - Federal Information Processing Standards Publication FIPS PUB 180-4, *Secure Hash Standard* (SHA-1 and SHA-2 family)
  - Federal Information Processing Standards Publication FIPS PUB 186-4, *Digital Signature Standard (DSS)*
  - Internet Engineering Task Force (IETF) Request For Comments RFC 1321, *MD5 Message-Digest Algorithm*
  - Internet Engineering Task Force (IETF) Request For Comments RFC 2104, *HMAC: Keyed-Hashing for Message Authentication* and Federal Information Processing Standards Publication FIPS PUB 198-1, *The Keyed-Hash Message Authentication Code (HMAC)*
- Fast computation of SHA-1, SHA2-224, SHA2-256, and MD5
  - 82 (respectively 66) clock cycles for processing one 512-bit block of data using SHA-1 (respectively SHA2-256) algorithm
  - 66 clock cycles for processing one 512-bit block of data using MD5 algorithm
- Corresponding 32-bit words of the digest from consecutive message blocks are added to each other to form the digest of the whole message
  - Automatic 32-bit words swapping to comply with the internal little-endian representation of the input bit-string
  - Word swapping supported: bits, bytes, half-words and 32-bit words
- Automatic padding to complete the input bit string to fit digest minimum block size of 512 bits ( $16 \times 32$  bits)
- Single 32-bit input register associated to an internal input FIFO, corresponding to one block size
- AHB slave peripheral, accessible through 32-bit word accesses only (else an AHB error is generated)
- $8 \times 32$ -bit words (H0 to H7) for output message digest

- Automatic data flow control with support of direct memory access (DMA) using one channel.
- Single or fixed DMA burst transfers of four words
- Interruptible message digest computation, on a per-block basis
  - Re-loadable digest registers
  - Hashing computation suspend/resume mechanism, including DMA

## 21.3 HASH implementation

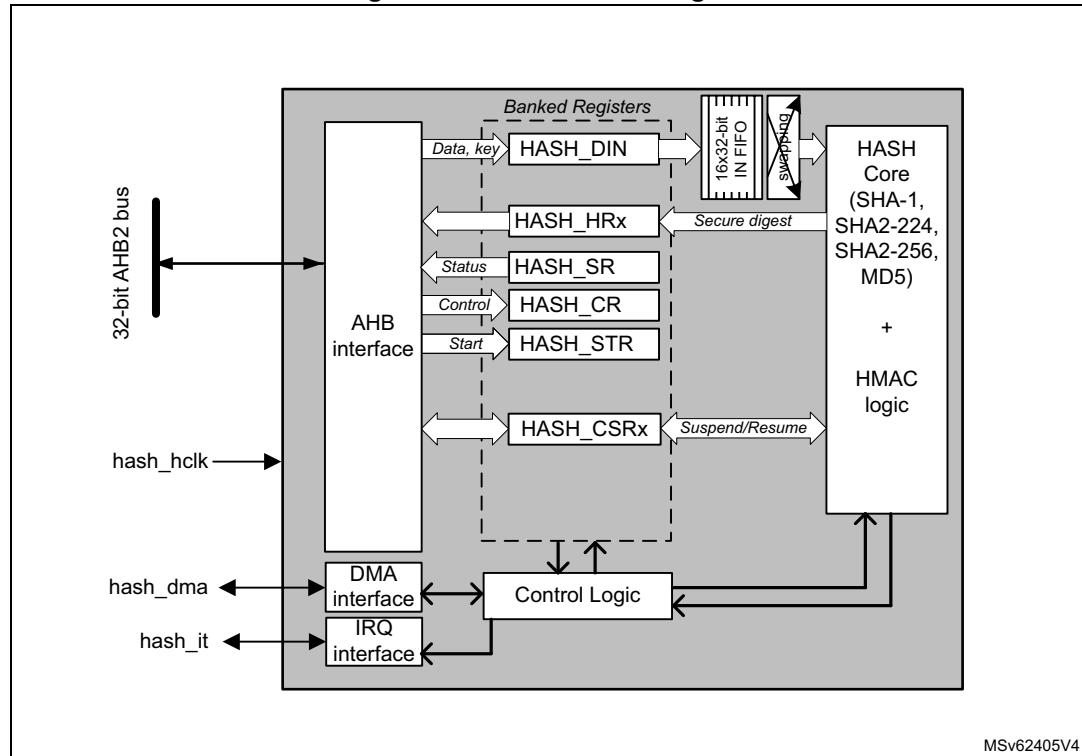
The devices have a single instance of HASH peripheral.

## 21.4 HASH functional description

### 21.4.1 HASH block diagram

*Figure 142* shows the block diagram of the hash processor.

**Figure 142. HASH block diagram**



### 21.4.2 HASH internal signals

*Table 142* describes a list of useful to know internal signals available at HASH level, not at product level (on pads).

**Table 142. HASH internal input/output signals**

Signal name	Signal type	Description
hash_hclk	digital input	AHB bus clock
hash_it	digital output	Hash processor global interrupt request
hash_dma	digital input/output	DMA transfer request/ acknowledge

### 21.4.3 About secure hash algorithms

The hash processor is a fully compliant implementation of the secure hash algorithm defined by FIPS PUB 180-4 standard and the IETF RFC1321 publication (MD5).

With each algorithm, the HASH computes a condensed representation of a message or data file. More specifically, when a message of any length below  $2^{64}$  bits is provided on input, the HASH processing core produces respectively a fixed-length output string called a message digest, defined as follows:

- For MD5 digest size is 128-bit
- For SHA-1 digest size is 160-bit
- For SHA2-224 and SHA2-256, the digest size is 224 bits and 256 bits, respectively

The message digest can then be processed with a digital signature algorithm in order to generate or verify the signature for the message.

Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The verifier of a digital signature has to use the same hash algorithm as the one used by the creator of the digital signature.

The SHA-2 functions supported by the hash processor are qualified as “secure” by NIST because it is computationally infeasible to find a message that corresponds to a given message digest, or to find two different messages that produce the same message digest (SHA-1 does not qualify as secure since February 2017). Any change to a message in transit, with very high probability, results in a different message digest, and the signature fails to verify.

### 21.4.4 Message data feeding

The message (or data file) to be processed by the HASH is considered as a bit string. Per FIPS PUB 180-4 standard this message bit string grows from left to right, with hexadecimal words expressed in “big-endian” convention, so that within each word, the most significant bit is stored in the left-most bit position. For example message string “abc” with a bit string representation of “01100001 01100010 01100011” is represented by a 32-bit word **0x00636261**, and 8-bit words **0x61626300**.

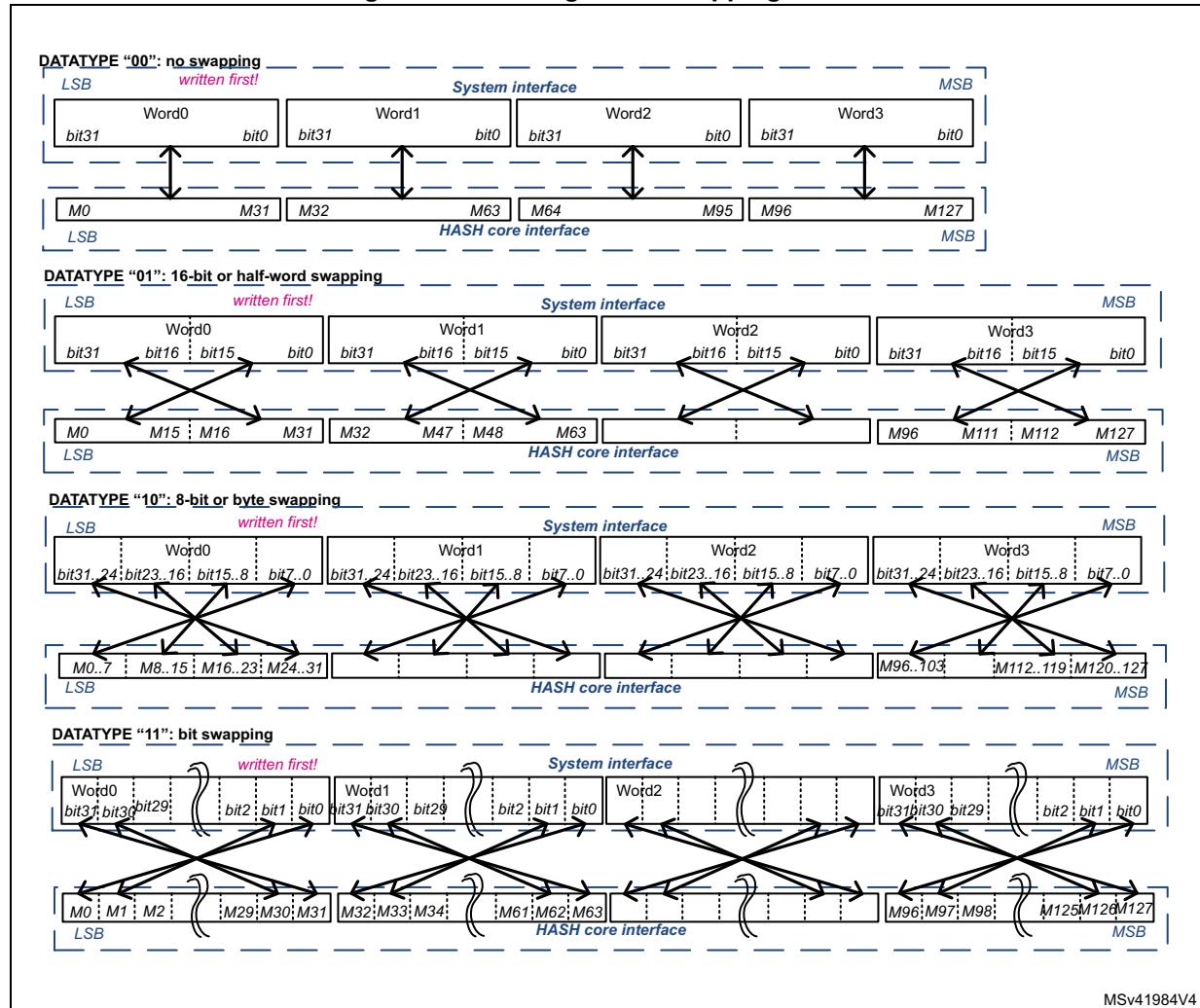
Data are entered into the HASH one 32-bit word at a time, by writing them into the HASH\_DIN register. The current contents of the HASH\_DIN register are transferred to the 16 words input FIFO each time the register is written with new data. Hence HASH\_DIN and the FIFO form a seventeen 32-bit words length FIFO (named the IN buffer).

In accordance to the kind of data to be processed (e.g. byte swapping when data are ASCII text stream) there must be a bit, byte, half-word or no swapping operation to be performed on data from the input FIFO before entering the little-endian hash processing core.

**Figure 143** shows how the hash processing core 32-bit data block M0...31 is constructed from one 32-bit words popped into input FIFO by the driver, according to the DATATYPE bitfield in the HASH control register (HASH\_CR).

HASH\_DIN data endianness when bit swapping is disabled (DATATYPE = 00) can be described as following: the least significant bit of the message has to be at MSB position in the first word entered into the hash processor, the 32nd bit of the bit string has to be at MSB position in the second word entered into the hash processor and so on.

**Figure 143. Message data swapping feature**



### 21.4.5 Message digest computing

The hash processor sequentially processes several blocks when computing the message digest. For MD5, SHA1 and SHA2, the block size is 512 bits.

Each time the DMA or the CPU writes a block to the hash processor, the HASH automatically starts computing the message digest. This operation is known as partial digest computation.

As described in [Section 21.4.4: Message data feeding](#), the message to be processed is entered into the HASH 32-bit word at a time, writing to the HASH\_DIN register to fill the input FIFO.

In order to perform the hash computation on this data below sequence must be used by the application:

1. Initialize the hash processor using the HASH\_CR register:
  - a) Select the right algorithm using the ALGO bitfield. If needed program the correct swapping operation on the message input words using DATATYPE bitfield in HASH\_CR.
  - b) When the HMAC mode is required, set the MODE bit, as well as the LKEY bit if the HMAC key size is greater than the known block size of the algorithm (else keep LKEY cleared). Refer to [Section 21.4.7: HMAC operation](#) for details.
  - c) Update NBLW[4:0] to define the number of valid bits in last word of the message if it is different from 32 bits. NBLW[4:0] information are used to correctly perform the automatic message padding before the final message digest computation.
2. Complete the initialization by setting to 1 the INIT bit in HASH\_CR. Also set the bit DMAE to 1 if data are transferred via DMA.

**Caution:** When programming step 2, it is important to set up before or at the same time the correct configuration values (ALGO, DATATYPE, HMAC mode, key length, NBLW[4:0]).

3. Start filling data by writing to HASH\_DIN register, unless data are automatically transferred via DMA. Note that the processing of a block can start only once the last value of the block has entered the input FIFO. The way the partial or final digest computation is managed depends on the way data are fed into the processor:
  - a) When data are filled by software:
    - Partial digest computation are triggered each time the application writes the first word of the next block. Once the processor is ready again (DINIS = 1 in HASH\_SR), the software can write new data to HASH\_DIN. This mechanism avoids the introduction of wait states by the HASH.
    - The final digest computation is triggered when the last block is entered and the software writes the DCAL bit to 1. If the message length is not an exact multiple of the block size, the NBLW[4:0] bitfield in HASH\_STR register must be written prior to writing DCAL bit (see [Section 21.4.6](#) for details).
  - b) When data are filled by DMA as a single DMA transfer (MDMAT bit = 0):
    - Partial digest computations are triggered automatically each time the FIFO is full. The final digest computation is triggered automatically when the last block has been transferred to the HASH\_DIN register (DCAL bit is set to 1 by hardware). If the message length is not an exact multiple of the block size, the NBLW[4:0] field

in HASH\_STR register must be written prior to enabling the DMA (see [Section 21.4.6](#) for details).

- c) When data are filled by DMA using multiple DMA transfers (MDMAT bit = 1):
  - Partial digest computations are triggered as for single DMA transfers. However the final digest computation is not triggered automatically when the last block has been transferred by DMA to the HASH\_DIN register (DCAL bit is not set to 1 by hardware). It allows the hash processor to receive a new DMA transfer as part of this digest computation. To launch the final digest computation, the software must set MDMAT bit to 0 before the last DMA transfer in order to trigger the final digest computation as it is done for single DMA transfers (see description before).
- 4. Once the digest computation is complete (DCIS = 1), the resulting digest can be read from the output registers as described in [Table 143](#).

**Table 143. Hash processor outputs**

Algorithm	Valid output registers	Most significant bit	Digest size (in bits)
MD5	HASH_H0 to HASH_H3	HASH_H0[31]	128
SHA-1	HASH_H0 to HASH_H4	HASH_H0[31]	160
SHA2-224	HASH_H0 to HASH_H6	HASH_H0[31]	224
SHA2-256	HASH_H0 to HASH_H7		256

For more information about HMAC detailed instructions, refer to [Section 21.4.7: HMAC operation](#).

## 21.4.6 Message padding

### Overview

When computing a condensed representation of a message, the process of feeding data into the hash processor (with automatic partial digest computation every block transfer) loops until the last bits of the original message are written to the HASH\_DIN register.

As the length (number of bits) of a message can be any integer value, the last word written to the hash processor may have a valid number of bits between 1 and 32. This number of valid bits in the last word, NBLW[4:0], has to be written to the HASH\_STR register, so that message padding is correctly performed before the final message digest computation.

### Padding processing

Detailed padding sequences with DMA enabled or disabled are described in [Section 21.4.5: Message digest computing](#).

### Padding example

As specified by Federal Information Processing Standards PUB 180-4, the message padding consists in appending a “1” followed by  $k$  “0”s, itself followed by a 64-bit integer that is equal to the length  $L$  in bits of the message. These three padding operations generate a padded message of length  $L + 1 + k + 64$ , which by construction is a multiple of 512 bits.

For the hash processor, the “1” is added to the last word written to the HASH\_DIN register at the bit position defined by the NBLW[4:0] bitfield, and the remaining upper bits are cleared (“0”s).

#### Example from FIPS PUB180-4

Let us assume that the original message is the ASCII binary-coded form of “abc”, of length L = 24:

```
byte 0      byte 1      byte 2      byte 3
01100001 01100010 01100011 UUUUUUUU
<-- 1st word written to HASH_DIN -->
```

NBLW[4:0] has to be loaded with the value 24: a “1” is appended at bit location 24 in the bit string (starting counting from left to right in the above bit string), which corresponds to bit 31 in the HASH\_DIN register (little-endian convention):

```
01100001 01100010 01100011 1UUUUUUU
```

Since L = 24, the number of bits in the above bit string is 25, and 423 “0” bits are appended, making now 448 bits.

This gives in hexadecimal (byte words in big-endian format):

```
61626380 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000018
```

The message length value, L, in two-word format (that is 00000000 00000018) is appended. Hence the final padded message in hexadecimal (byte words in big-endian format):

```
61626380 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000018
```

If the hash processor is programmed to swap byte within HASH\_DIN input register (DATATYPE = 10 in HASH\_CR), the above message has to be entered by following the below sequence:

1. **0xUU636261** is written to the HASH\_DIN register (where ‘U’ means don’t care).
2. **0x18** is written to the HASH\_STR register (the number of valid bits in the last word written to the HASH\_DIN register is 24, as the original message length is 24 bits).
3. **0x10** is written to the HASH\_STR register to start the message padding (described above) and then perform the digest computation.
4. The hash computing is complete with the message digest available in the HASH\_HRx registers (x = 0...4) for the SHA-1 algorithm. For this FIPS example, the expected value is as follows:

```
HASH_HR0 = 0xA9993E36
HASH_HR1 = 0x4706816A
HASH_HR2 = 0xBA3E2571
HASH_HR3 = 0x7850C26C
HASH_HR4 = 0x9CD0D89D
```

## 21.4.7 HMAC operation

### Overview

As specified by Internet Engineering Task Force RFC2104 and NIST FIPS PUB 198-1, the HMAC algorithm is used for message authentication by irreversibly binding the message being processed to a key chosen by the user. The algorithm consists of two nested hash operations:

```
HMAC(message) = Hash((Key | pad) XOR opad |  
                      Hash((Key | pad) XOR ipad | message))
```

where:

- **opad** = [0x5C]<sub>n</sub> (outer pad) and **ipad** = [0x36]<sub>n</sub> (inner pad)
- [X]<sub>n</sub> represents a repetition of X n times, where n equal to the size of the underlying hash function data block (n = 64 for 512-bit blocks).
- **pad** is a sequence of zeroes needed to extend the key to the length n defined above. If the key length is greater than n, the application must first hash the key using Hash() function and then use the resultant byte string as the actual key to HMAC.
- | represents the concatenation operator.

*Note:*

*HMAC mode of the hash processor can be used with all supported algorithms.*

### HMAC processing

Four different steps are required to compute the HMAC:

1. The software writes the INIT bit to 1 with the MODE bit at 1 and the ALGO bits set to the value corresponding to the desired algorithm. The LKEY bit must also be set to 1 if the key being used is longer than 64 bytes. In this case, as required by HMAC specifications, the hash processor uses the hash of the key instead of the real key.
2. The software provides the key to be used for the inner hash function, using the same mechanism as the message string loading, that is writing the key data into HASH\_DIN register then completing the transfer by writing DCAL bit to 1 and the correct NBLW[4:0] to HASH\_STR register.

*Note:*

*Endianness details can be found in [Section 21.4.4: Message data feeding](#).*

3. Once the processor is ready again (DINIS = 1 in HASH\_SR), the software can write the message string to HASH\_DIN. When the last word of the last block is entered and the software writes DCAL bit to 1 in HASH\_STR register, the NBLW[4:0] bitfield must be written at the same time to a value different from zero if the message length is not an exact multiple of the block size. Note that the DMA can also be used to feed the message string, as described in [Section 21.4.4: Message data feeding](#).
4. Once the processor is ready again (DINIS = 1 in HASH\_SR), the software provides the key to be used for the outer hash function, writing the key data into HASH\_DIN register then completing the transfer by writing DCAL bit to 1 and the correct NBLW[4:0] to HASH\_STR register. The HMAC result can be found in the valid output registers (HASH\_HRx) as soon as DCIS bit is set to 1.

*Note:*

*The computation latency of the HMAC primitive depends on the lengths of the keys and message, as described in [Section 21.4.11: HASH processing time](#).*

### HMAC example

Below is an example of HMAC SHA-1 algorithm (ALGO = 00 and MODE = 1 in HASH\_CR) as specified by NIST.

Let us assume that the original message is the ASCII binary-coded form of “**Sample message for keylen = blocklen**”, of length L = 34 bytes. If the HASH is programmed in no swapping mode (DATATYPE = 00 in HASH\_CR), the following data must be loaded sequentially into HASH\_DIN register:

1. **Inner hash key** input (length = 64, that is no padding), specified by NIST. As key length = 64, LKEY bit is set to 0 in HASH\_CR register  
`00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617  
18191A1B 1C1D1E1F 20212223 24252627 28292A2B 2C2D2E2F  
30313233 34353637 38393A3B 3C3D3E3F`
2. **Message** input (length = 34, that is padding required). HASH\_STR must be set to **0x20** to start message padding and inner hash computation (see ‘U’ as don’t care)  
`53616D70 6C65206D 65737361 67652066 6F72206B 65796C65  
6E3D626C 6F636B6C 656EUUUU`
3. **Outer hash key** input (length = 64, that is no padding). A key identical to the inner hash key is entered here.
4. **Final outer hash computing** is then performed by the HASH. The HMAC-SHA1 digest result is available in the HASH\_HRx registers (x = 0 to 4), as shown below:

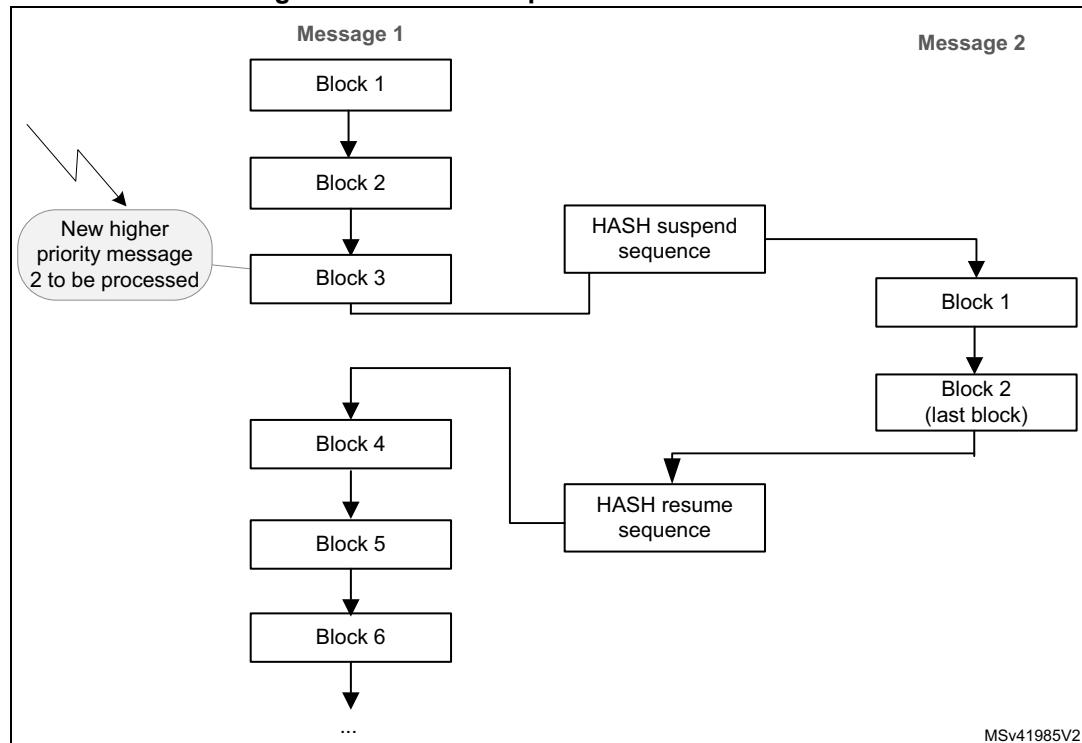
```
HASH_HR0 = 0x5FD596EE  
HASH_HR1 = 0x78D5553C  
HASH_HR2 = 0x8FF4E72D  
HASH_HR3 = 0x266DFD19  
HASH_HR4 = 0x2366DA29
```

## 21.4.8 HASH suspend/resume operations

### Overview

It is possible to interrupt a hash/HMAC operation to perform another processing with a higher priority. The interrupted process completes later when the higher-priority task has been processed, as shown in [Figure 144](#).

**Figure 144. HASH suspend/resume mechanism**



To do so, the context of the interrupted task must be saved from the HASH registers to memory, and then be restored from memory to the HASH registers.

The procedures where the data flow is controlled by software or by DMA are described hereafter.

### Data loaded by software

When the DMA is not used to load the message into the hash processor, the context can be saved only when no block processing is ongoing.

**To suspend the processing of a message**, proceed as follows after writing 16 words 32-bit (plus one if it is the first block):

1. In Polling mode, wait for BUSY = 0, then poll if the DINIS status bit is set to 1.  
In Interrupt mode, implement the next step in DINIS interrupt handler (recommended).
2. Store the contents of the following registers into memory:
  - HASH\_IMR
  - HASH\_STR
  - HASH\_CR
  - HASH\_CSR0 to HASH\_CSR37. HASH\_CSR38 to HASH\_CSR53 registers must also be saved if an HMAC operation was ongoing.

**To resume the processing of a message**, proceed as follows:

1. Write the following registers with the values saved in memory: HASH\_IMR, HASH\_STR and HASH\_CR.
2. Initialize the hash processor by setting the INIT bit in the HASH\_CR register.
3. Write the HASH\_CSRx registers with the values saved in memory.
4. Restart the processing from the point where it has been interrupted.

*Note:*

*To optimize the resume process when NBW[3:0] = 0x0, HASH\_CSR22 to HASH\_CSR37 registers do not need to be saved then restored as the FIFO is empty.*

### Data loaded by DMA

When the DMA is used to load the message into the hash processor, it is recommended to suspend and then restore a secure digest computing is described below.

**To suspend the processing of a message using DMA**, proceed as follows:

1. In Polling mode, wait for BUSY = 0. If DCIS is set in HASH\_SR, the hash result is available and the context swapping is useless. Else go to step 2.
2. In Polling mode, wait for BUSY = 1.
3. Disable the DMA channel. Then clear DMAE bit in HASH\_CR register.
4. In Polling mode, wait for BUSY = 0. If DCIS is set in HASH\_SR, the hash result is available and the context swapping is useless. Else go to step 5.
5. Save HASH\_IMR, HASH\_STR, HASH\_CR, and HASH\_CSR0 to HASH\_CSR37 registers. HASH\_CSR38 to HASH\_CSR53 registers must also be saved if an HMAC operation was ongoing.

**To resume the processing of a message using DMA**, proceed as follows:

1. Reconfigure the DMA controller so that it proceeds with the transfer of the message up to the end if it is not interrupted again. Do not forget to take into account the words that have been already pushed into the FIFO if NBW[3:0] is higher than 0x0.
2. Program the values saved in memory to HASH\_IMR, HASH\_STR and HASH\_CR registers.
3. Initialize the hash processor by setting the INIT bit in the HASH\_CR register.
4. Program the values saved in memory to the HASH\_CSRx registers.
5. Restart the processing from the point where it was interrupted by setting the DMAE bit.

**Note:** *To optimize the resume process when NBW[3:0] = 0x0, HASH\_CSR22 to HASH\_CSR37 registers do not need to be saved then restored as the FIFO is empty.*

#### 21.4.9 HASH DMA interface

The HASH supports both single and fixed DMA burst transfers of four words.

The hash processor provides an interface to connect to the DMA controller. This DMA can be used to write data to the HASH by setting the DMAE bit in the HASH\_CR register. When this bit is set, the HASH initiates a DMA request each time a block has to be written to the HASH\_DIN register.

Once four 32-bit words have been received, the HASH automatically triggers a new request to the DMA. For more information refer to [Section 21.4.5: Message digest computing](#).

Before starting the DMA transfer, the software must program the number of valid bits in the last word that is copied into HASH\_DIN register. This is done by writing in HASH\_STR register the following value:

**NBLW[4:0] = Len(Message) % 32** where “x%32” gives the remainder of x divided by 32.

The DMAS bit of the HASH\_SR register provides information on the DMA interface activity. This bit is set with DMAE and cleared when DMAE is cleared and no DMA transfer is ongoing.

**Note:** *No interrupt is associated to DMAS bit.*

*When MDMAT is set, the size of the transfer must be a multiple of four words.*

#### 21.4.10 HASH error management

No error flags are generated by the hash processor.

#### 21.4.11 HASH processing time

[Table 144](#) summarizes the time required to process an intermediate block for each mode of operation.

**Table 144. Processing time (in clock cycle)**

Mode of operation	FIFO load <sup>(1)</sup>	Computation phase	Total
MD5	16	50	<b>66</b>
SHA-1	16	66	<b>82</b>

**Table 144. Processing time (in clock cycle) (continued)**

Mode of operation	FIFO load <sup>(1)</sup>	Computation phase	Total
SHA2-224	16	50	<b>66</b>
SHA2-256			

1. Add the time required to load the block into the processor.

The time required to process the last block of a message (or of a key in HMAC) can be longer. This time depends on the length of the last block and the size of the key (in HMAC mode).

Compared to the processing of an intermediate block, it can be increased by the factor below:

- **1 to 2.5** for a hash message
- **~2.5** for an HMAC input-key
- **1 to 2.5** for an HMAC message
- **~2.5** for an HMAC output key in case of a short key
- **3.5 to 5** for an HMAC output key in case of a long key

## 21.5 HASH interrupts

Two individual maskable interrupt sources are generated by the hash processor to signal the following events:

- Digest calculation completion (DCIS)
- Data input buffer ready (DINIS)

Both interrupt sources are connected to the same global interrupt request signal (hash\_it), which is in turn connected to the NVIC (nested vectored interrupt controller). Each interrupt source can individually be enabled or disabled by changing the mask bits in the HASH\_IMR register. Setting the appropriate mask bit to 1 enables the interrupt.

The status of each maskable interrupt source can be read from the HASH\_SR register. [Table 145](#) gives a summary of the available features.

**Table 145. HASH interrupt requests**

Interrupt acronym	Interrupt event	Event flag	Enable control bit	Interrupt clear method
HASH	Digest computation completed	DCIS	DCIE	Clear DCIS or set INIT
	Data input buffer ready to get a new block	DINIS	DINIE	Clear DINIS or write to HASH_DIN

## 21.6 HASH registers

The HASH core is associated with several control and status registers and several message digest registers. All these registers are accessible through 32-bit word accesses only, else an AHB error is generated.

### 21.6.1 HASH control register (HASH\_CR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ALGO1	Res.	LKEY
													rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	MDMAT	DINNE	NBW[3:0]				ALGO0	MODE	DATATYPE[1:0]	DMAE	INIT	Res.	Res.	
		rw	r	r	r	r	r	rw	rw	rw	rw	rw	rw		

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

Bit 17 Reserved, must be kept at reset value.

Bit 16 **LKEY:** Long key selection

This bit selects between short key ( $\leq$  64 bytes) or long key ( $>$  64 bytes) in HMAC mode.

0: the HMAC key is shorter or equal to 64 bytes. The actual key value written to HASH\_DIN is used during the HMAC computation.

1: the HMAC key is longer than 64 bytes. The hash of the key is used instead of the real key during the HMAC computation.

This selection is only taken into account when the INIT and MODE bits are both set.

Changing this bit during a computation has no effect.

Bit 15 Reserved, must be kept at reset value.

Bit 14 Reserved, must be kept at reset value.

Bit 13 **MDMAT:** Multiple DMA transfers

This bit is set when hashing large files when multiple DMA transfers are needed.

0: DCAL is automatically set at the end of a DMA transfer.

1: DCAL is not automatically set at the end of a DMA transfer.

Bit 12 **DINNE:** DIN not empty

This bit is set when the HASH\_DIN register holds valid data (that is after being written at least once). It is cleared when either the INIT bit (initialization) or the DCAL bit (completion of the previous message processing) is written to 1.

0: No data are present in the data input buffer

1: The input buffer contains at least one word of data

This bit is read-only.

Bits 11:8 **NBW[3:0]**: Number of words already pushed

This bitfield reflects the number of words in the message that have already been pushed into the IN FIFO. NBW is incremented by one when a write access to the HASH\_DIN register is performed (except if DINNE = 0 and the DMA is not used, see below description). NBW goes to zero when the INIT bit is written to 1.

This bitfield is read-only.

**If the DMA is not used**

0000: if DINNE = 0, no word has been pushed into the DIN buffer (both HASH\_DIN register and IN FIFO are empty), otherwise one word has been pushed into the DIN buffer (HASH\_DIN register contains one word and IN FIFO is empty)

0001: two words have been pushed into the DIN buffer (that is HASH\_DIN register and the IN FIFO contain one word each)

...

1111: 16 words have been pushed into the DIN buffer.

**If the DMA is used**

NBW contains the exact number of words that have been pushed into the IN FIFO by the DMA.

Bits 18, 7 **ALGO[1:0]**: Algorithm selection

These bits select the hash algorithm.

00: SHA-1

01: MD5

10: SHA2-224

11: SHA2-256

This selection is only taken into account when the INIT bit is set. Changing this bitfield during a computation has no effect.

Bit 6 **MODE**: Mode selection

This bit selects the HASH or HMAC mode for the selected algorithm:

0: Hash mode selected

1: HMAC mode selected. LKEY must be set if the key being used is longer than 64 bytes.

This selection is only taken into account when the INIT bit is set. Changing this bit during a computation has no effect.

Bits 5:4 **DATATYPE[1:0]**: Data type selection

Defines the format of the data entered into the HASH\_DIN register:

00: 32-bit data. The data written into HASH\_DIN are directly used by the HASH processing, without reordering.

01: 16-bit data, or half-word. The data written into HASH\_DIN are considered as two half-words, and are swapped before being used by the HASH processing.

10: 8-bit data, or bytes. The data written into HASH\_DIN are considered as four bytes, and are swapped before being used by the HASH processing.

11: bit data, or bit-string. The data written into HASH\_DIN are considered as 32 bits (1st bit of the string at position 0), and are swapped before being used by the HASH processing (1st bit of the string at position 31).

Bit 3 **DMAE:** DMA enable

0: DMA transfers disabled

1: DMA transfers enabled. A DMA request is sent as soon as the HASH core is ready to receive data.

After this bit is set it is cleared by hardware while the last data of the message is written into the hash processor.

Setting this bit to 0 while a DMA transfer is ongoing is not aborting this current transfer.

Instead, the DMA interface of the IP remains internally enabled until the transfer is completed or INIT is written to 1.

Setting INIT bit to 1 does not clear DMAE bit.

Bit 2 **INIT:** Initialize message digest calculation

Writing this bit to 1 resets the hash processor core, so that the HASH is ready to compute the message digest of a new message.

Writing this bit to 0 has no effect. Reading this bit always return 0.

Bits 1:0 Reserved, must be kept at reset value.

**21.6.2 HASH data input register (HASH\_DIN)**

Address offset: 0x04

Reset value: 0x0000 0000

HASH\_DIN is the data input register. It is 32-bit wide. This register is used to enter the message by blocks. When the HASH\_DIN register is programmed, the value presented on the AHB databus is ‘pushed’ into the hash core and the register takes the new value presented on the AHB databus. To get a correct message format, the DATATYPE bits must have been previously configured in the HASH\_CR register.

When a complete block has been written to the HASH\_DIN register, an intermediate digest calculation is launched:

- by writing new data into the HASH\_DIN register (the first word of the next block) if the DMA is not used (intermediate digest calculation),
- automatically if the DMA is used.

When the last block has been written to the HASH\_DIN register, the final digest calculation (including padding) is launched by writing the DCAL bit to 1 in the HASH\_STR register (final digest calculation). This operation is automatic if the DMA is used and MDMAT bit is set to 0.

Reading the HASH\_DIN register returns the last word written to this location (zero after reset).

**Note:** *When the HASH is busy, a write access to the HASH\_DIN register might stall the AHB bus if the digest calculation (intermediate or final) is not complete.*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATAIN[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
DATAIN[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DATAIN[31:0]**: Data input

Writing this register pushes the current register content into the IN FIFO, and the register takes the new value presented on the AHB databus.

Reading this register returns the current register content.

### 21.6.3 HASH start register (HASH\_STR)

Address offset: 0x08

Reset value: 0x0000 0000

The HASH\_STR register has two functions:

- It is used to define the number of valid bits in the last word of the message entered in the hash processor (that is the number of valid least significant bits in the last data written to the HASH\_DIN register).
- It is used to start the processing of the last block in the message by writing the DCAL bit to 1.

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

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

Bit 8 **DCAL**: Digest calculation

Writing this bit to 1 starts the message padding, using the previously written value of NBLW[4:0], and starts the calculation of the final message digest with all data words written to the input FIFO since the INIT bit was last written to 1.

Reading this bit returns 0.

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

Bits 4:0 **NBLW[4:0]**: Number of valid bits in the last word

When the last word of the message bit string is written in HASH\_DIN register, the hash processor takes only the valid bits specified as below, after internal data swapping:

0x00: All 32 bits of the last data written are valid message bits that is M[31:0]

0x01: Only one bit of the last data written (after swapping) is valid that is M[0]

0x02: Only two bits of the last data written (after swapping) are valid that is M[1:0]

0x03: Only three bits of the last data written (after swapping) are valid that is M[2:0]

...

0x1F: Only 31 bits of the last data written (after swapping) are valid that is M[30:0]

The above mechanism is valid only if DCAL = 0. If NBLW[4:0] bitfield is written while DCAL is set to 1, the NBLW[4:0] bitfield remains unchanged. In other words it is not possible to configure NBLW[4:0] and set DCAL at the same time.

Reading NBLW[4:0] bitfield returns the last value written to NBLW[4:0].

## 21.6.4 HASH digest registers

These registers contain the message digest result named as follows:

- HASH\_HR0, HASH\_HR1, HASH\_HR2, HASH\_HR3 and HASH\_HR4 registers return the SHA-1 digest result
- HASH\_HR0, HASH\_HR1, HASH\_HR2 and HASH\_HR3 registers return A, B, C and D (respectively), as defined by MD5.
- HASH\_HR0 to HASH\_HR6 registers return the SHA2-224 digest result.
- HASH\_HR0 to HASH\_HR7 registers return the SHA2-256 digest result.

In all cases, the digest most significant bit is stored in HASH\_H0[31] and unused HASH\_HRx registers are read as zeros.

If a read access to one of these registers is performed while the hash core is calculating an intermediate digest or a final message digest (DCIS bit equals 0), then the read operation is stalled until the hash calculation has completed.

*Note:* When starting a digest computation for a new message (by writing the INIT bit to 1), HASH\_HRx registers are forced to their reset values.

HASH\_HR0 to HASH\_HR4 registers can be accessed through two different addresses.

### HASH aliased digest register x (HASH\_HRAx)

Address offset: 0x0C + 0x4 \* x, (x = 0 to 4)

Reset value: 0x0000 0000

The content of the HASH\_HRAx registers is identical to the one of the HASH\_HRx registers located at address offset 0x310.

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

Bits 31:0 Hx[31:0]: Hash data x

Refer to [Section 21.6.4: HASH digest registers](#) introduction.

### HASH digest register x (HASH\_HRx)

Address offset: 0x310 + 0x4 \* x, (x = 0 to 4)

Reset value: 0x0000 0000

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

Bits 31:0 **Hx[31:0]**: Hash data x

Refer to [Section 21.6.4: HASH digest registers](#) introduction.

### HASH supplementary digest register x (HASH\_HRx)

Address offset: 0x310 + 0x4 \* x, (x = 5 to 7)

Reset value: 0x0000 0000

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

Bits 31:0 **Hx[31:0]**: Hash data x

Refer to [Section 21.6.4: HASH digest registers](#) introduction.

### 21.6.5 HASH interrupt enable register (HASH\_IMR)

Address offset: 0x20

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.	DCIE	DINIE													
														rw	rw

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

Bit 1 **DCIE**: Digest calculation completion interrupt enable

0: Digest calculation completion interrupt disabled

1: Digest calculation completion interrupt enabled.

Bit 0 **DINIE**: Data input interrupt enable

0: Data input interrupt disabled

1: Data input interrupt enabled

## 21.6.6 HASH status register (HASH\_SR)

Address offset: 0x24

Reset value: 0x0000 0001

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.	rc_w0	DINIS													
												r	r	rc_w0	

Bits 31:4 Reserved, must be kept at reset value.

Bit 3 **BUSY**: Busy bit

- 0: No block is currently being processed
- 1: The hash core is processing a block of data

Bit 2 **DMAS**: DMA Status

This bit provides information on the DMA interface activity. It is set with DMAE and cleared when DMAE = 0 and no DMA transfer is ongoing. No interrupt is associated with this bit.

- 0: DMA interface is disabled (DMAE = 0) and no transfer is ongoing
- 1: DMA interface is enabled (DMAE = 1) or a transfer is ongoing

Bit 1 **DCIS**: Digest calculation completion interrupt status

This bit is set by hardware when a digest becomes ready (the whole message has been processed). It is cleared by writing it to 0 or by writing the INIT bit to 1 in the HASH\_CR register.

- 0: No digest available in the HASH\_HRx registers (zeros are returned)
- 1: Digest calculation complete, a digest is available in the HASH\_HRx registers. An interrupt is generated if the DCIE bit is set in the HASH\_IMR register.

Bit 0 **DINIS**: Data input interrupt status

This bit is set by hardware when the FIFO is ready to get a new block (16 locations are free). It is cleared by writing it to 0 or by writing the HASH\_DIN register.

- 0: Less than 16 locations are free in the input buffer
- 1: A new block can be entered into the input buffer. An interrupt is generated if the DINIE bit is set in the HASH\_IMR register.

When DINIS=0, HASH\_CSRx registers reads as zero.

## 21.6.7 HASH context swap registers

These registers contain the complete internal register states of the hash processor. They are useful when a suspend/resume operation has to be performed because a high-priority task needs to use the hash processor while it is already used by another task.

When such an event occurs, the HASH\_CSRx registers have to be read and the read values have to be saved in the system memory space. Then the hash processor can be used by the preemptive task, and when the hash computation is complete, the saved context can be read from memory and written back into the HASH\_CSRx registers.

HASH\_CSRx registers can be read only when DINIS equals to 1, otherwise zeros are returned.

**HASH context swap register x (HASH\_CSRx)**

Address offset: 0x0F8 + x \* 0x4, (x = 0 to 53)

Reset value: 0x0000 0002 (HASH\_CSR0)

Reset value: 0x0000 0000 (HASH\_CSR1 to 53)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CSx[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
CSx[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **CSx[31:0]**: Context swap xRefer to [Section 21.6.7: HASH context swap registers](#) introduction.**21.6.8 HASH register map****Table 146. HASH register map and reset values**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00	HASH_CR	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	
0x04	HASH_DIN	DATAIN[31:16]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x08	HASH_STR	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	
0x0C	HASH_HRA0	H0[31:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x10	HASH_HRA1	H1[31:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x14	HASH_HRA2	H2[31:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x18	HASH_HRA3	H3[31:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x1C	HASH_HRA4	H4[31:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x20	HASH_IMR	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	
0x24	HASH_SR	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	

Table 146. HASH register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28- 0xF4	Reserved																																
0x0F8	HASH_CSR0																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
0x0F8 + 0x4 * x, (x=1 to 53) Last address: 0x1CC	HASH_CSRx																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
...																																	
0x1D0- 0x30C	Reserved																																
0x310	HASH_HR0																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x314	HASH_HR1																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x318	HASH_HR2																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x31C	HASH_HR3																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x320	HASH_HR4																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x324	HASH_HR5																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x328	HASH_HR6																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x32C	HASH_HR7																																
	Reset value	0	0	0	0	0	0	0	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: Memory organization](#) for the register boundary addresses.

## 22 Advanced-control timers (TIM1/TIM8)

### 22.1 TIM1/TIM8 introduction

The advanced-control timers (TIM1/TIM8) consist of a 16-bit auto-reload counter driven by a programmable prescaler.

It may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM, complementary PWM with dead-time insertion).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

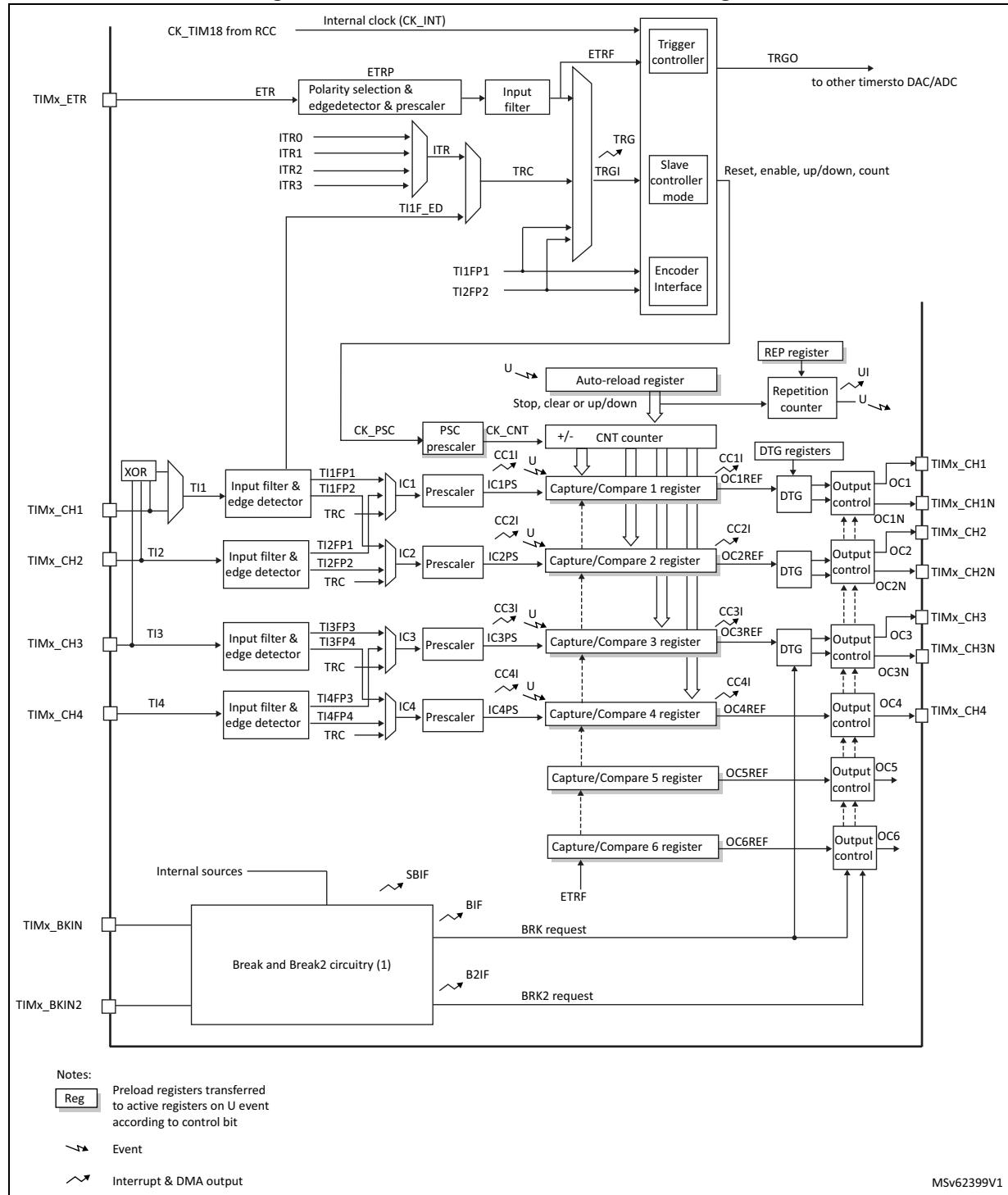
The advanced-control (TIM1/TIM8) and general-purpose (TIMy) timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 22.3.25: Timer synchronization](#).

### 22.2 TIM1/TIM8 main features

TIM1/TIM8 timer features include:

- 16-bit up, down, up/down auto-reload counter.
- 16-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency either by any factor between 1 and 65536.
- Up to 6 independent channels for:
  - Input Capture (but channels 5 and 6)
  - Output Compare
  - PWM generation (Edge and Center-aligned Mode)
  - One-pulse mode output
- Complementary outputs with programmable dead-time
- Synchronization circuit to control the timer with external signals and to interconnect several timers together.
- Repetition counter to update the timer registers only after a given number of cycles of the counter.
- 2 break inputs to put the timer’s output signals in a safe user selectable configuration.
- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
- Supports incremental (quadrature) encoder and Hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

Figure 145. Advanced-control timer block diagram



1. See [Figure 187: Break and Break2 circuitry overview](#) for details

MSv62399V1

## 22.3 TIM1/TIM8 functional description

### 22.3.1 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.

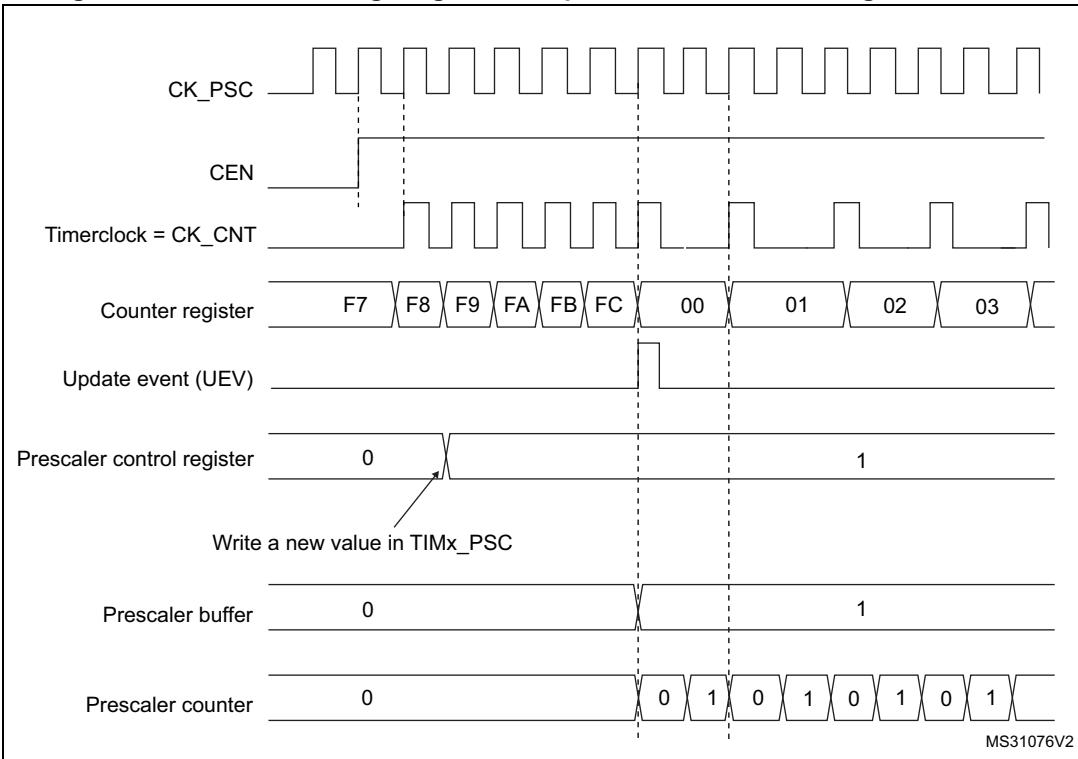
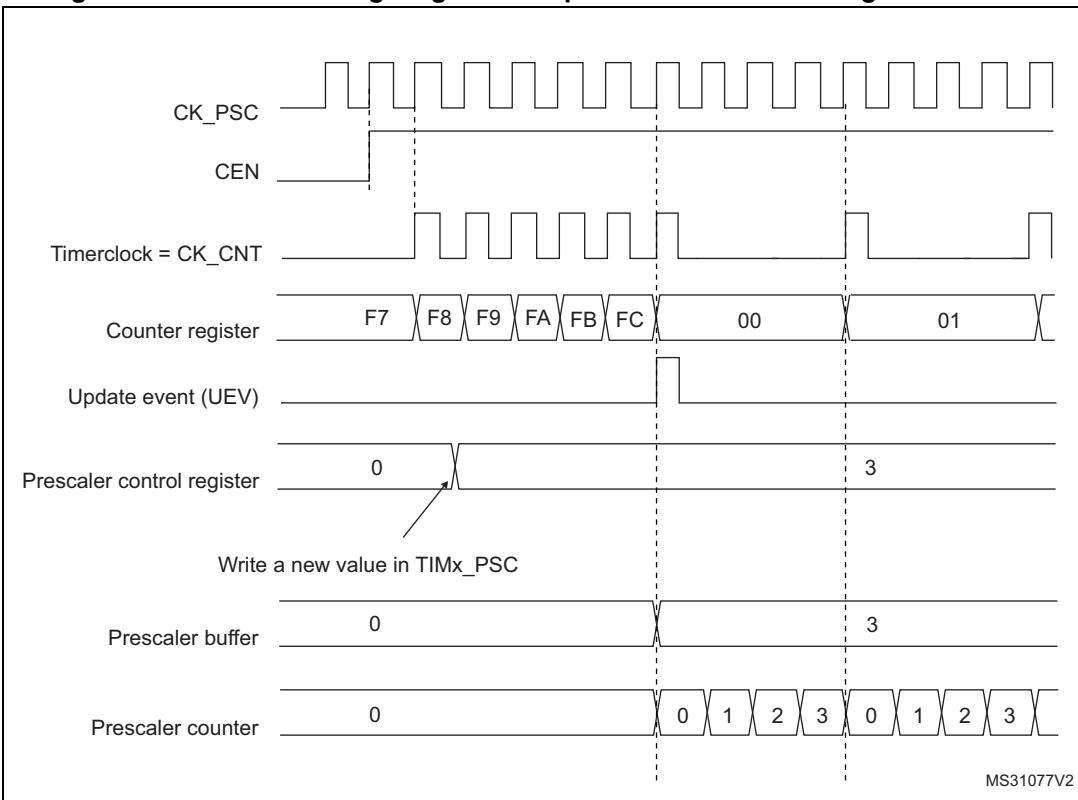
The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

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 146* and *Figure 147* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 146. Counter timing diagram with prescaler division change from 1 to 2****Figure 147. Counter timing diagram with prescaler division change from 1 to 4**

## 22.3.2 Counter modes

### Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register), then restarts from 0 and generates a counter overflow event.

If the repetition counter is used, the update event (UEV) is generated after upcounting is repeated for the number of times programmed in the repetition counter register (TIMx\_RCR) + 1. Else the update event is generated at each counter overflow.

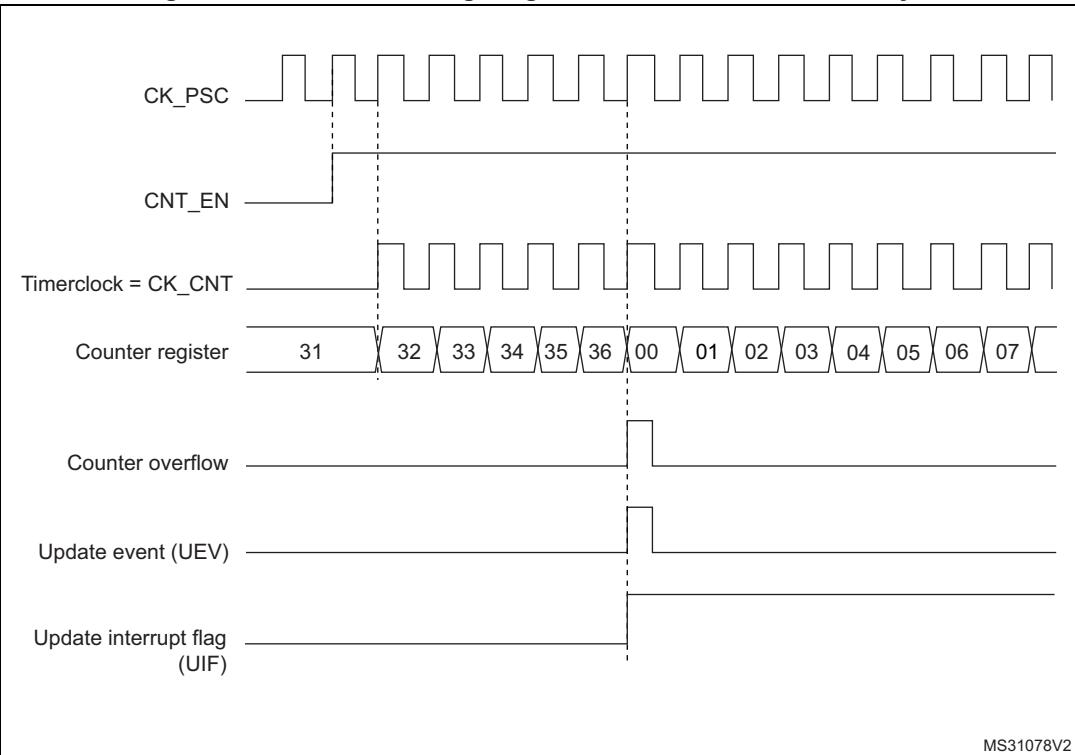
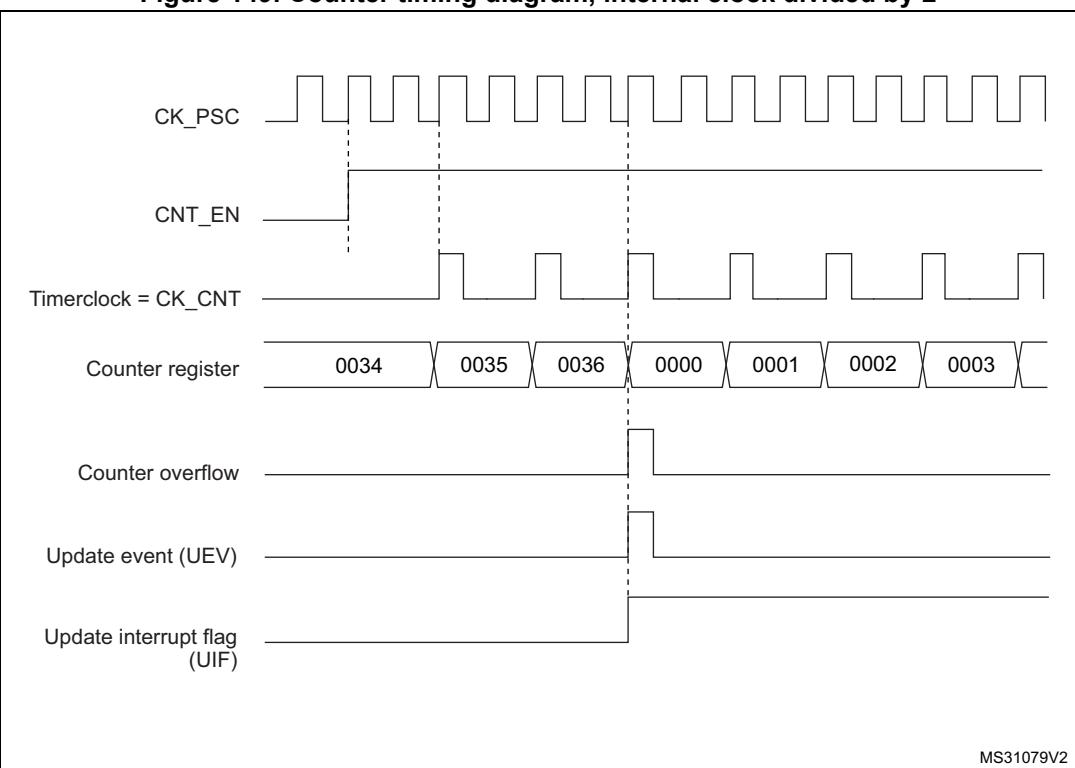
Setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller) also generates an update event.

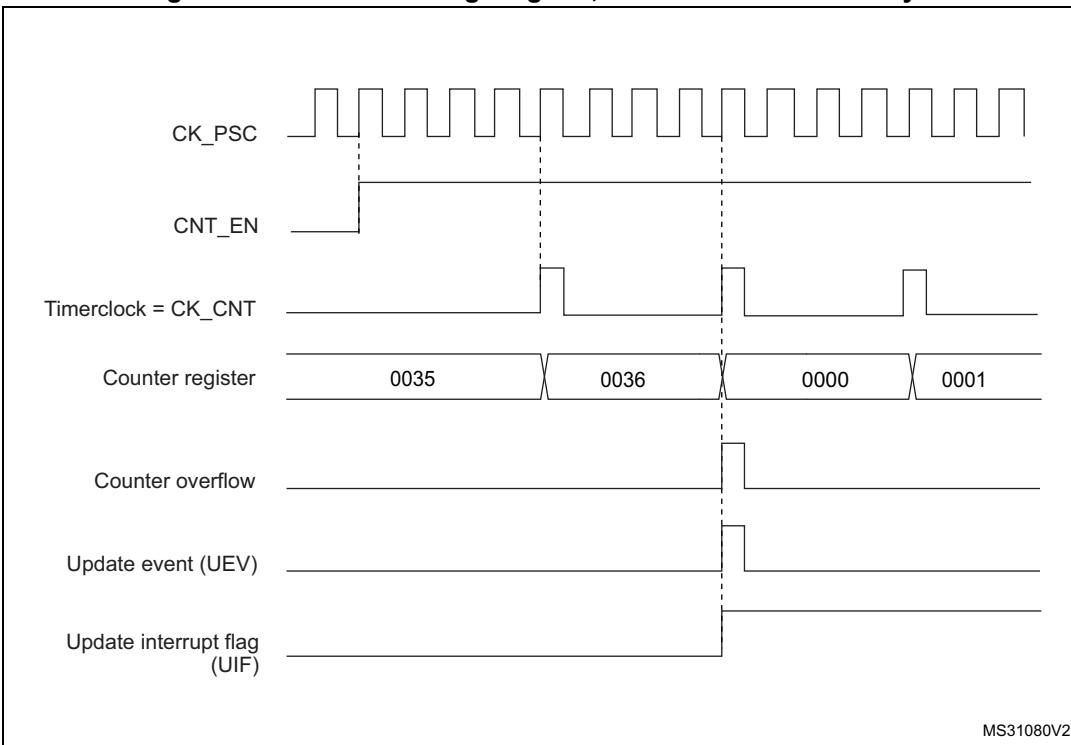
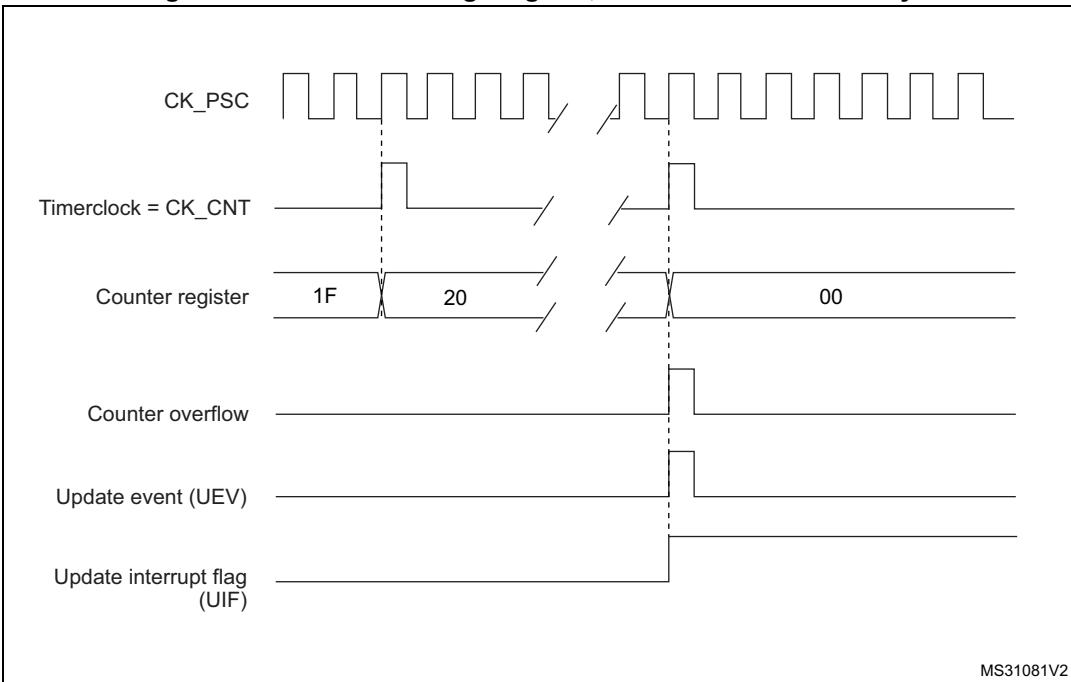
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

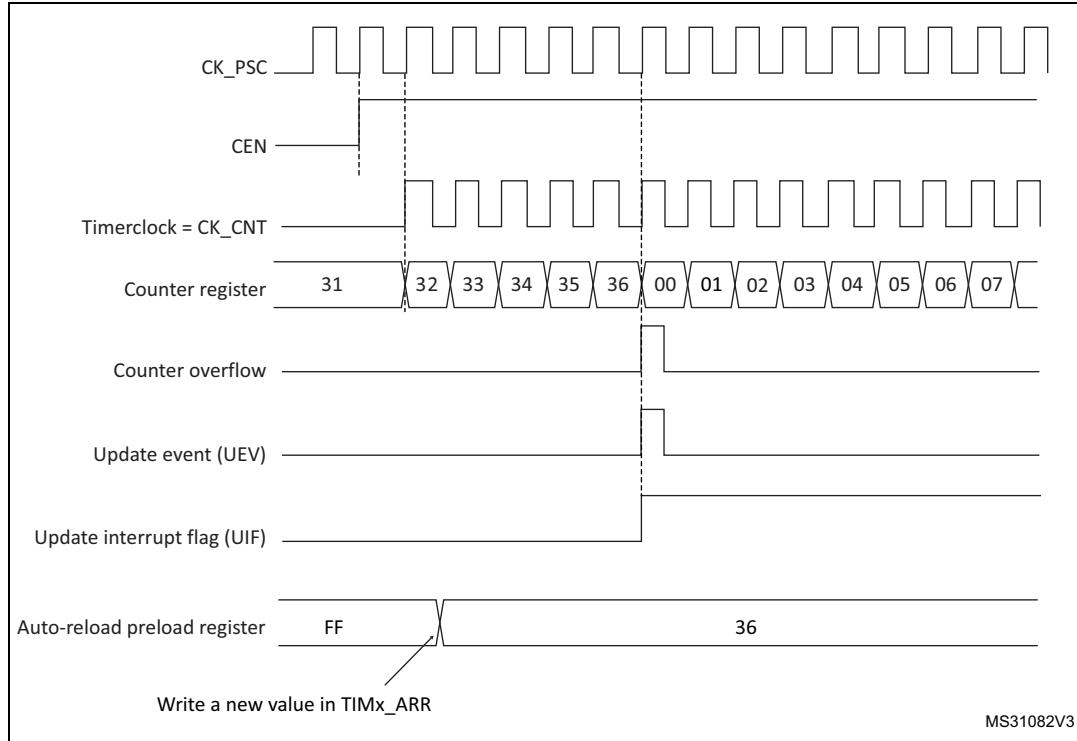
- The repetition counter is reloaded with the content of TIMx\_RCR register,
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

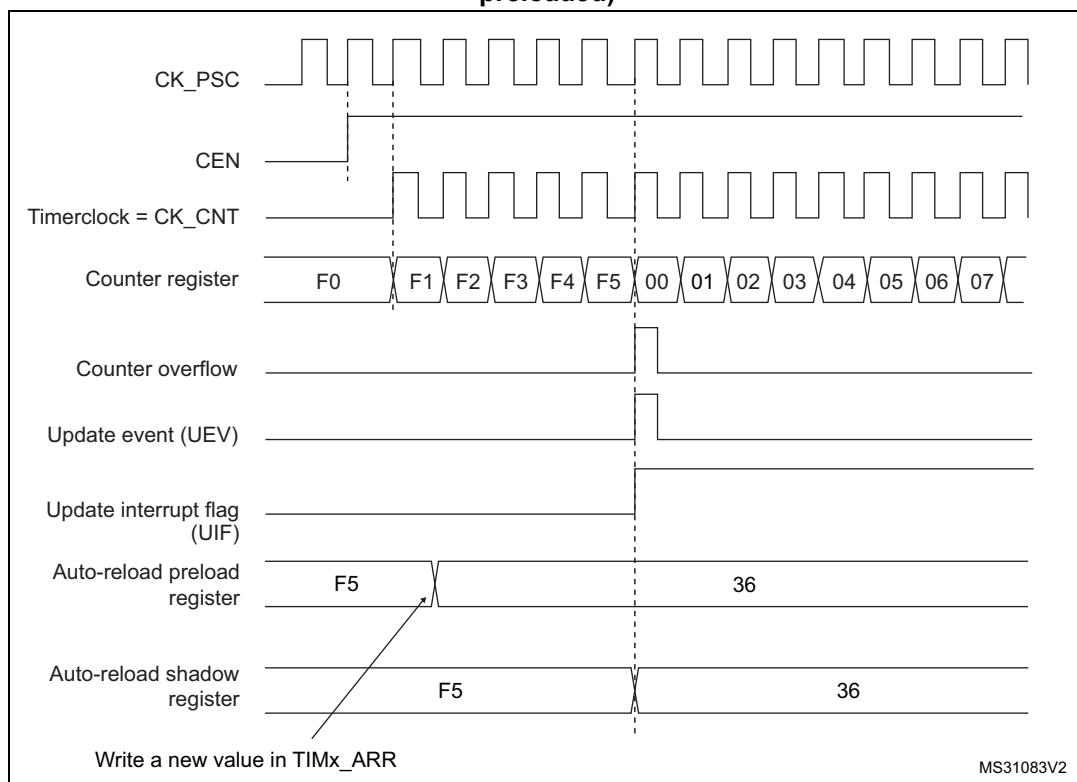
**Figure 148. Counter timing diagram, internal clock divided by 1****Figure 149. Counter timing diagram, internal clock divided by 2**

**Figure 150. Counter timing diagram, internal clock divided by 4****Figure 151. Counter timing diagram, internal clock divided by N**

**Figure 152. Counter timing diagram, update event when ARPE=0 (TIMx\_ARR not preloaded)**



**Figure 153. Counter timing diagram, update event when ARPE=1 (TIMx\_ARR preloaded)**



## Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIMx\_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

If the repetition counter is used, the update event (UEV) is generated after downcounting is repeated for the number of times programmed in the repetition counter register (TIMx\_RCR) + 1. Else the update event is generated at each counter underflow.

Setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller) also generates an update event.

The UEV update event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

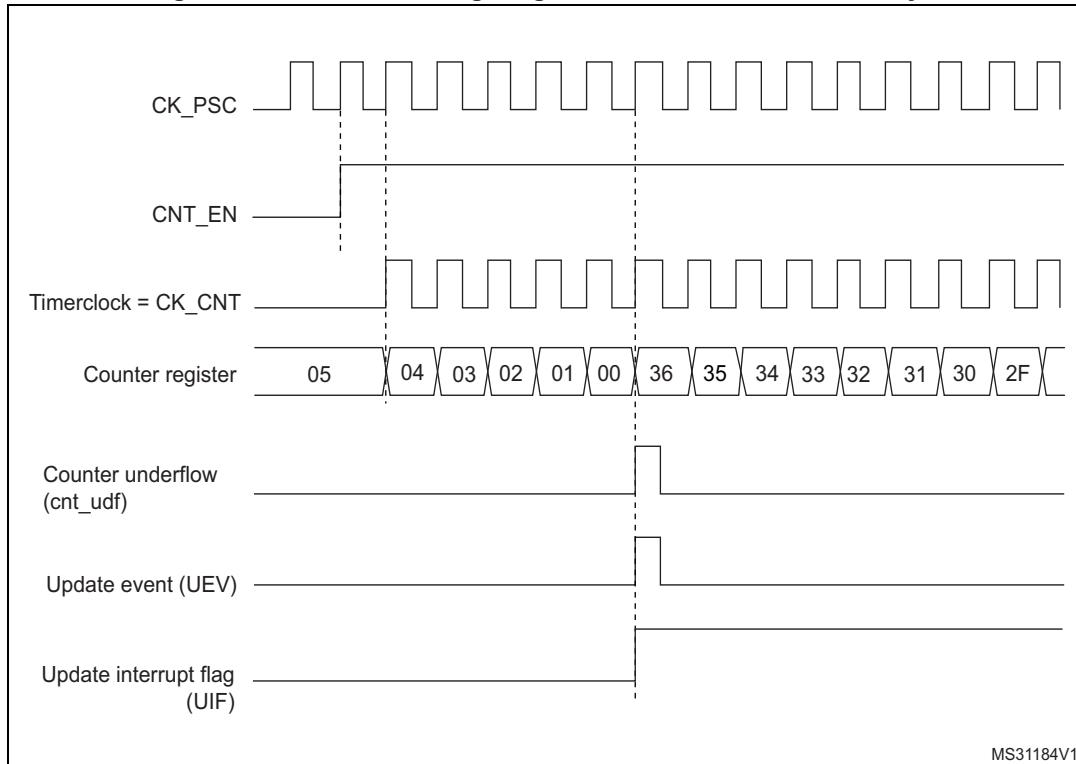
In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on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

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

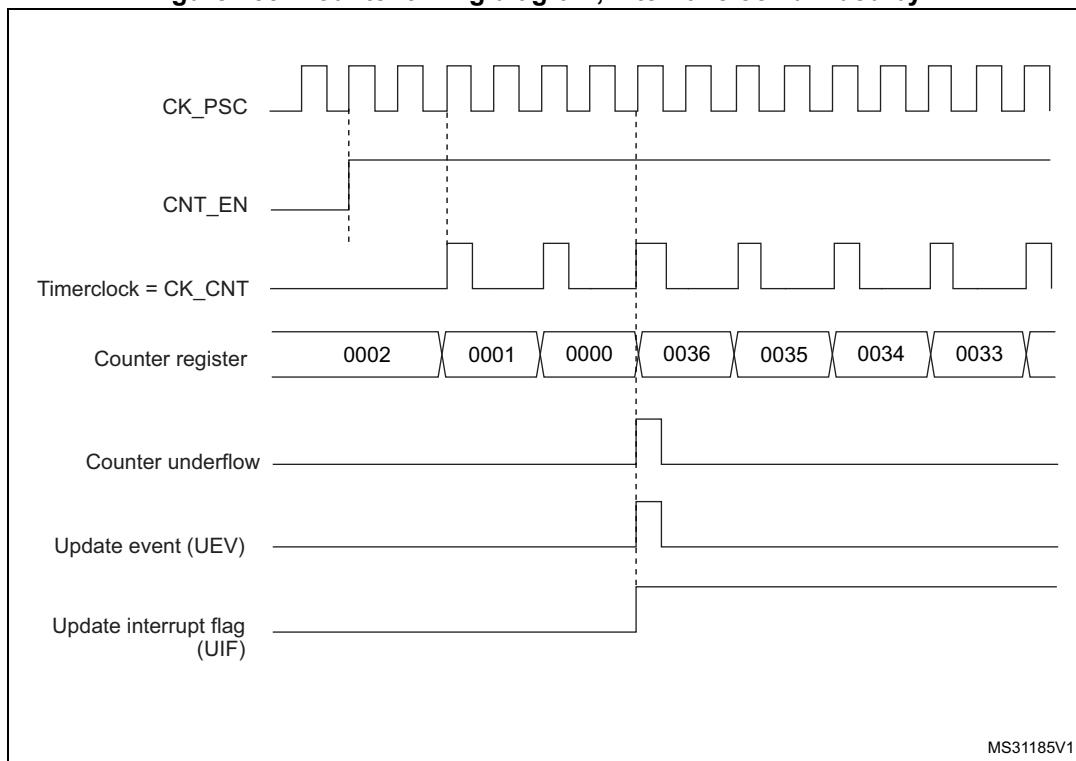
- The repetition counter is reloaded with the content of TIMx\_RCR register.
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

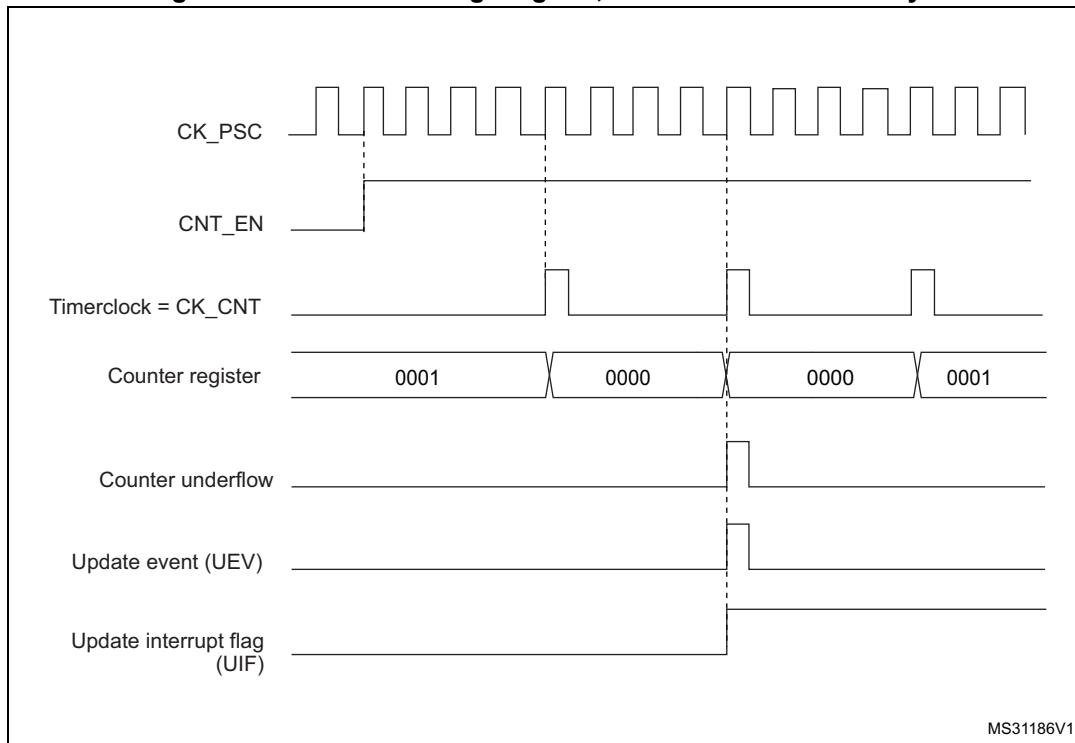
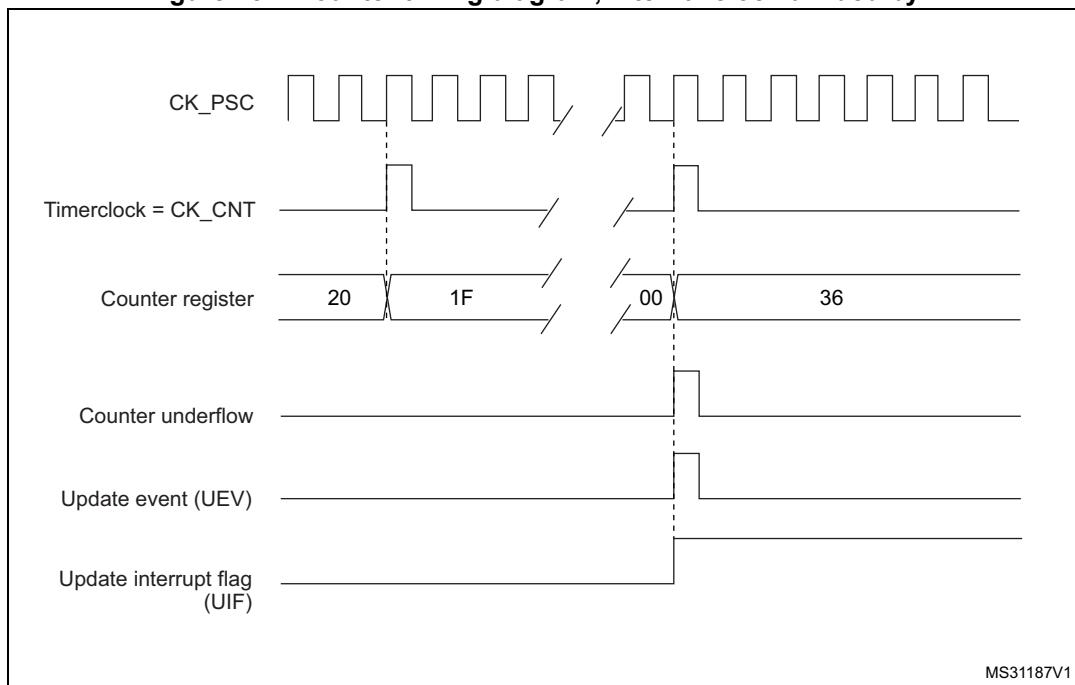
The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

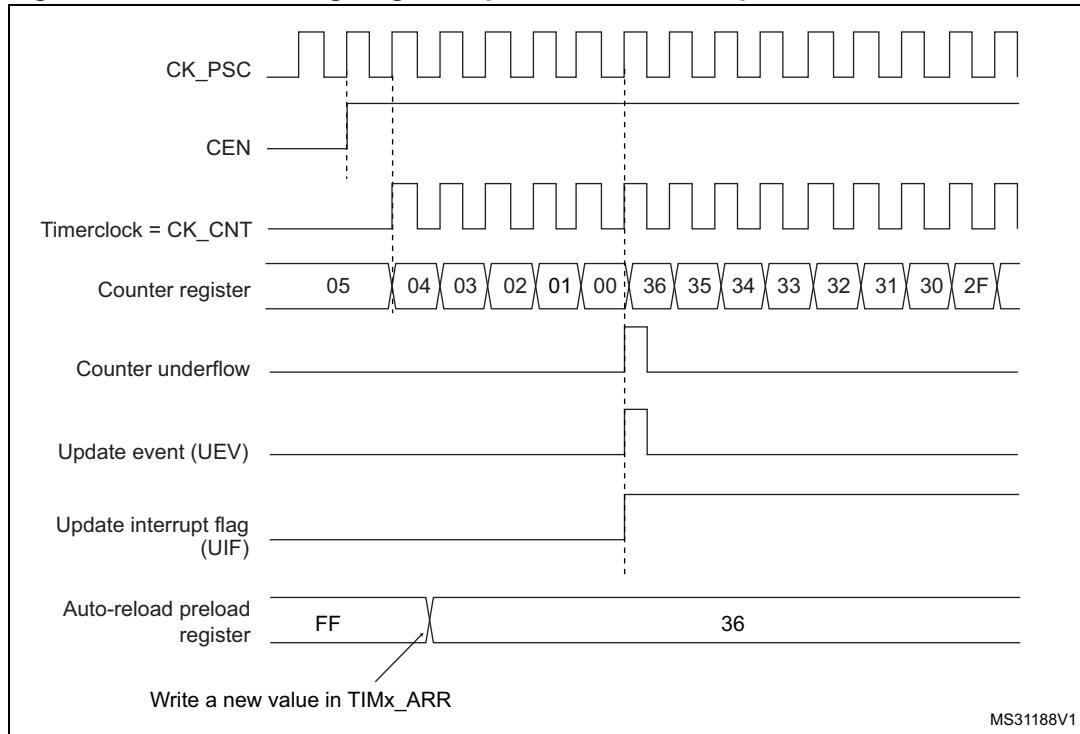
**Figure 154.** Counter timing diagram, internal clock divided by 1



**Figure 155.** Counter timing diagram, internal clock divided by 2



**Figure 156. Counter timing diagram, internal clock divided by 4****Figure 157. Counter timing diagram, internal clock divided by N**

**Figure 158. 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

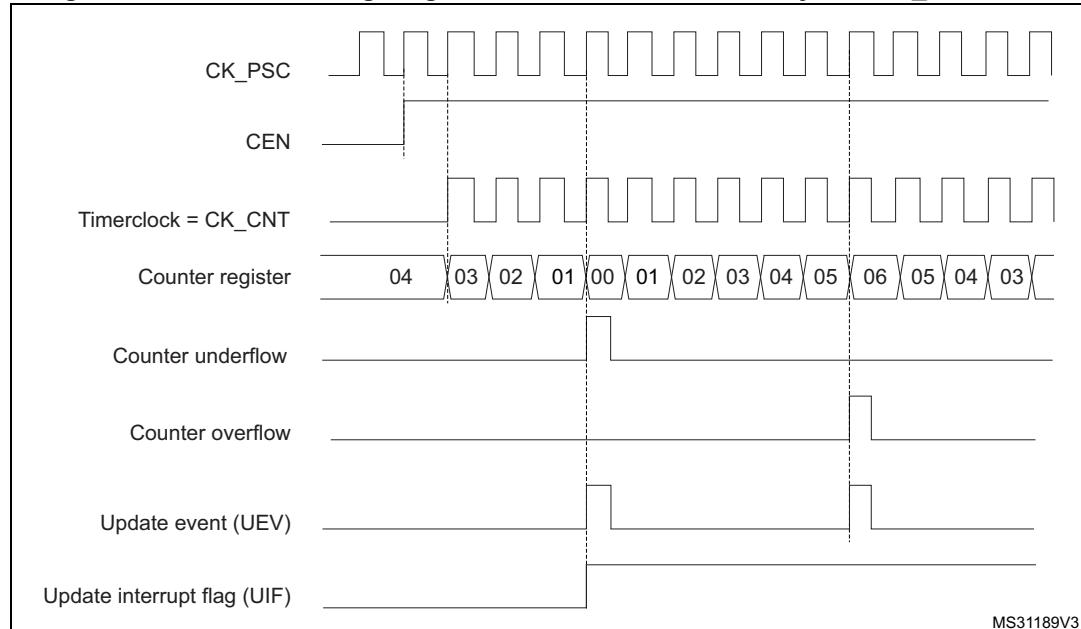
DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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):

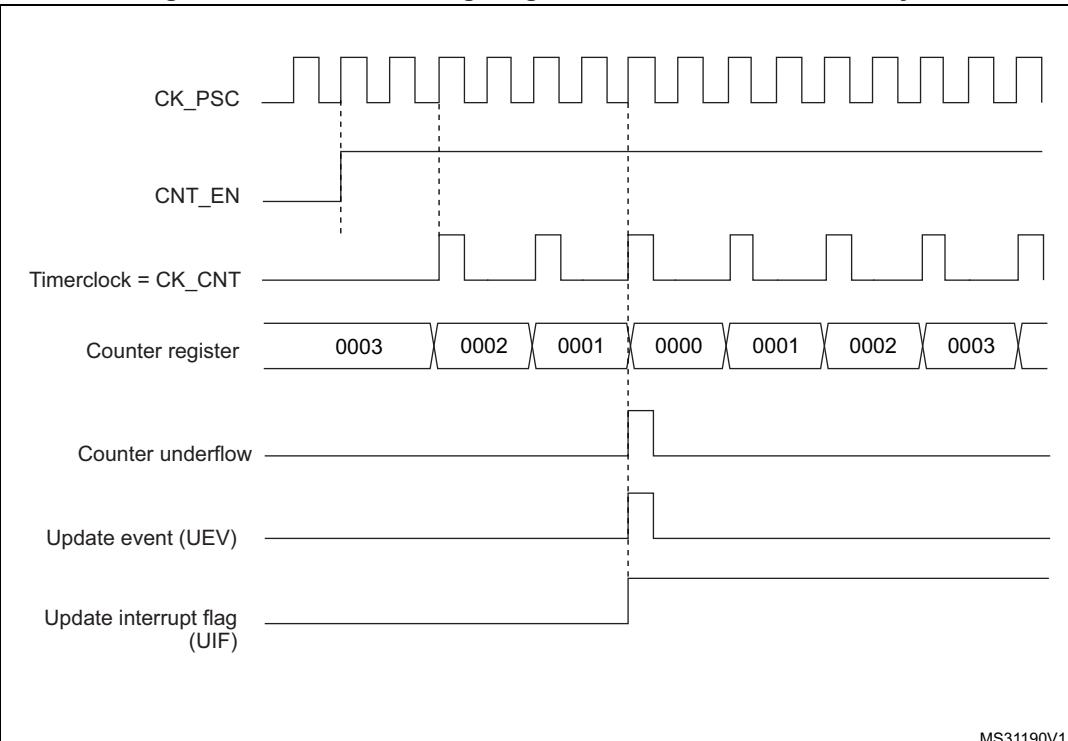
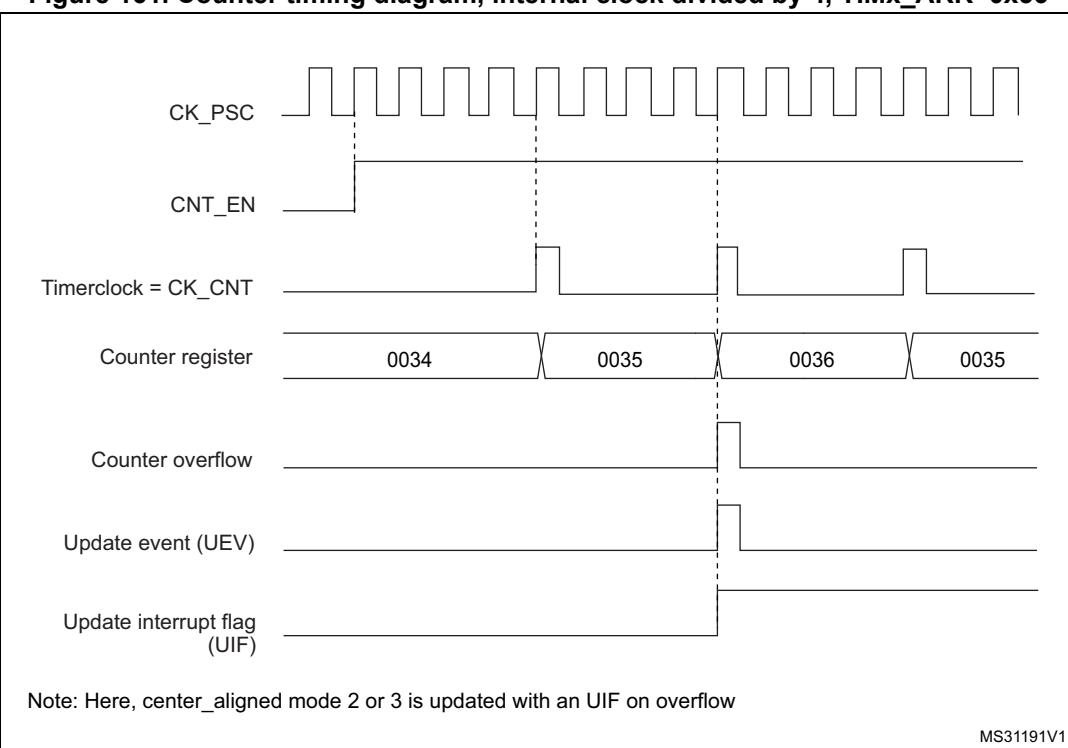
- The repetition counter is reloaded with the content of TIMx\_RCR register
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register)
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that if the update 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).

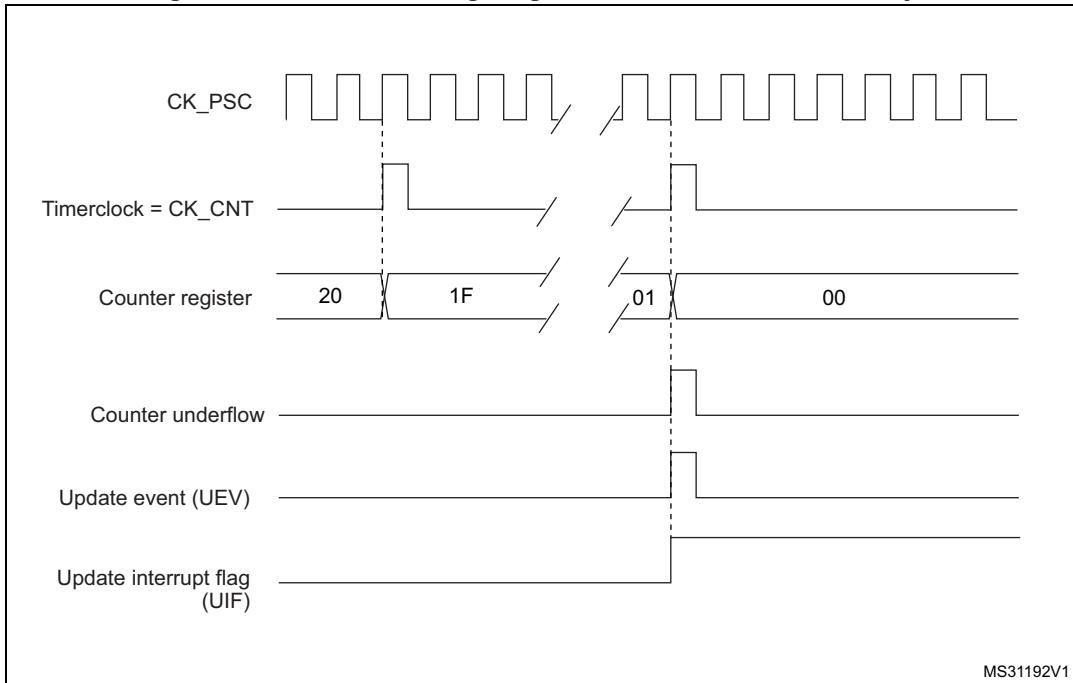
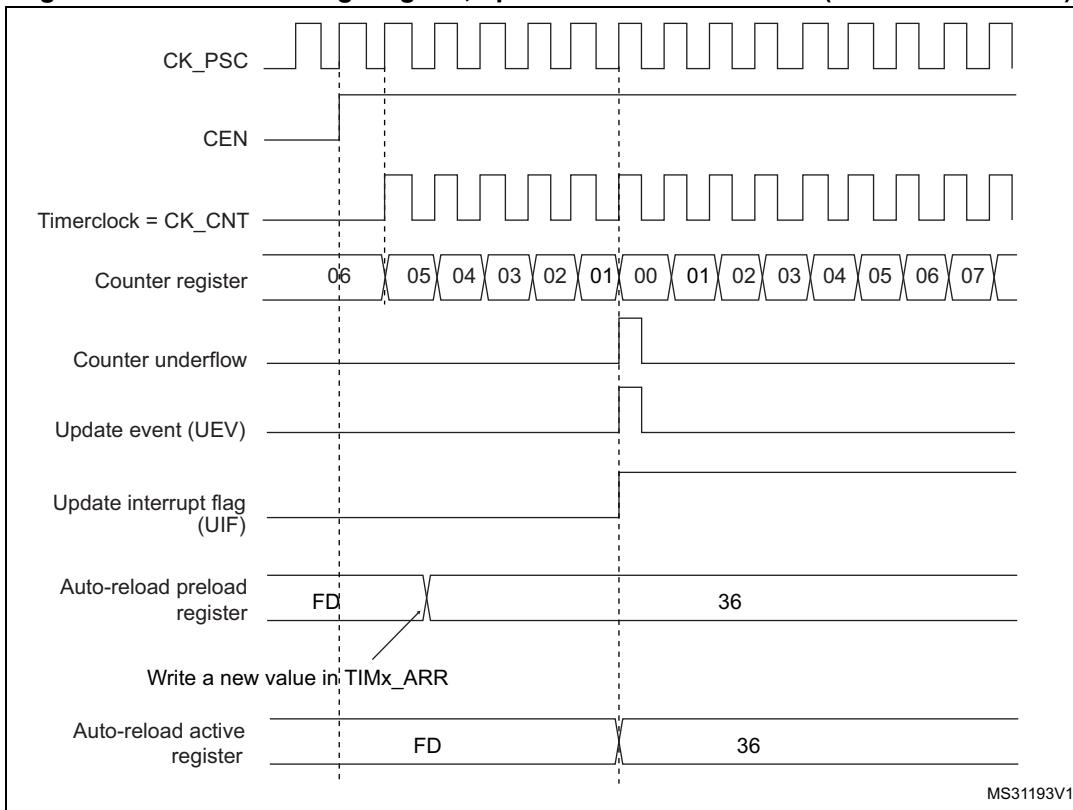
The following figures show some examples of the counter behavior for different clock frequencies.

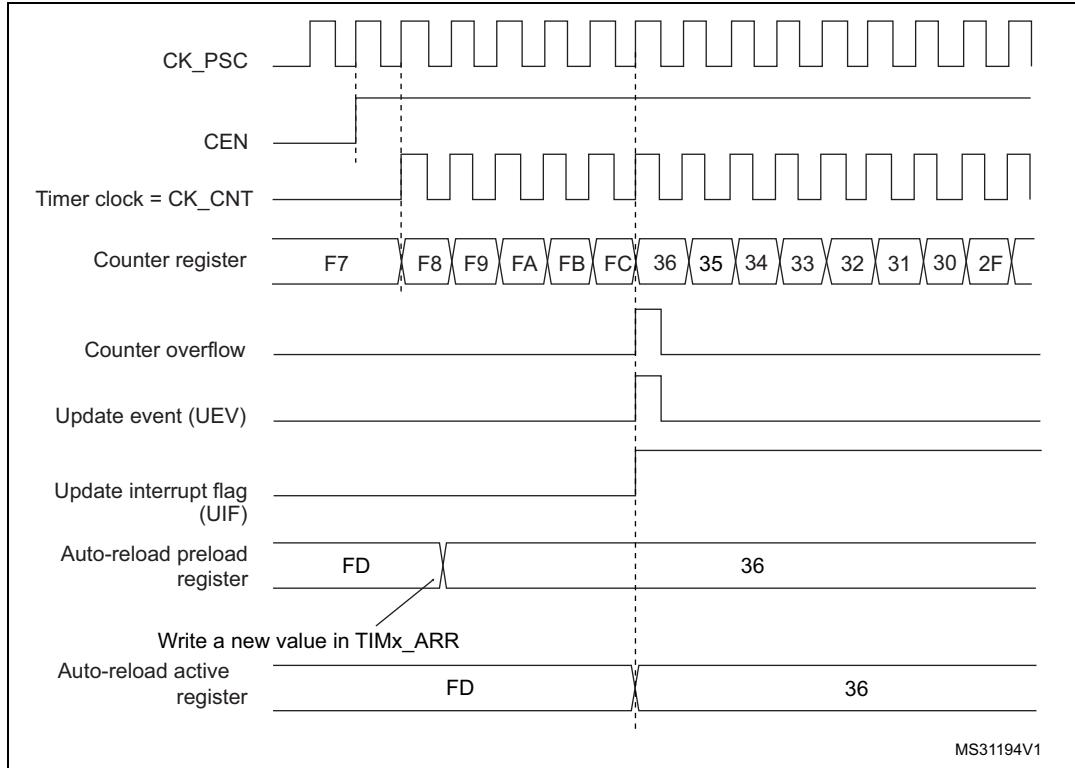
**Figure 159. Counter timing diagram, internal clock divided by 1, TIMx\_ARR = 0x6**



1. Here, center-aligned mode 1 is used (for more details refer to [Section 22.4: TIM1/TIM8 registers](#)).

**Figure 160. Counter timing diagram, internal clock divided by 2****Figure 161. Counter timing diagram, internal clock divided by 4, TIMx\_ARR=0x36**

**Figure 162. Counter timing diagram, internal clock divided by N****Figure 163. Counter timing diagram, update event with ARPE=1 (counter underflow)**

**Figure 164. Counter timing diagram, Update event with ARPE=1 (counter overflow)**

### 22.3.3 Repetition counter

[Section 22.3.1: Time-base unit](#) describes how the update event (UEV) is generated with respect to the counter overflows/underflows. It is actually generated only when the repetition counter has reached zero. This can be useful when generating PWM signals.

This means that data are transferred from the preload registers to the shadow registers (TIMx\_ARR auto-reload register, TIMx\_PSC prescaler register, but also TIMx\_CCRx capture/compare registers in compare mode) every N+1 counter overflows or underflows, where N is the value in the TIMx\_RCR repetition counter register.

The repetition counter is decremented:

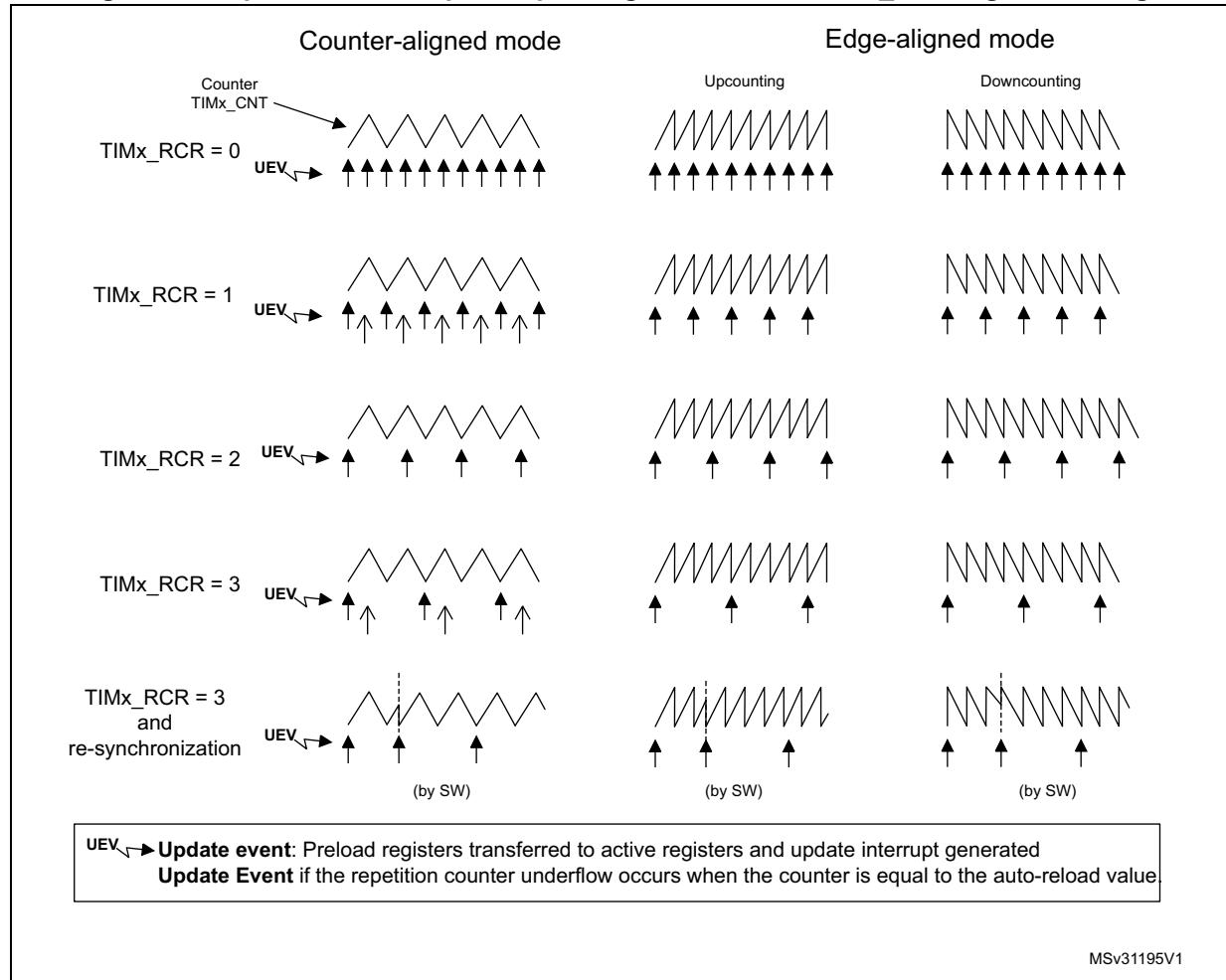
- At each counter overflow in upcounting mode,
  - At each counter underflow in downcounting mode,
  - At each counter overflow and at each counter underflow in center-aligned mode.
- Although this limits the maximum number of repetition to 32768 PWM cycles, it makes it possible to update the duty cycle twice per PWM period. When refreshing compare registers only once per PWM period in center-aligned mode, maximum resolution is  $2 \times T_{ck}$ , due to the symmetry of the pattern.

The repetition counter is an auto-reload type; the repetition rate is maintained as defined by the TIMx\_RCR register value (refer to [Figure 165](#)). When the update event is generated by software (by setting the UG bit in TIMx\_EGR register) or by hardware through the slave mode controller, it occurs immediately whatever the value of the repetition counter is and the repetition counter is reloaded with the content of the TIMx\_RCR register.

In Center aligned mode, for odd values of RCR, the update event occurs either on the overflow or on the underflow depending on when the RCR register was written and when the counter was launched: if the RCR was written before launching the counter, the UEV occurs on the underflow. If the RCR was written after launching the counter, the UEV occurs on the overflow.

For example, for RCR = 3, the UEV is generated each 4th overflow or underflow event depending on when the RCR was written.

**Figure 165. Update rate examples depending on mode and TIMx\_RCR register settings**



MSv31195V1

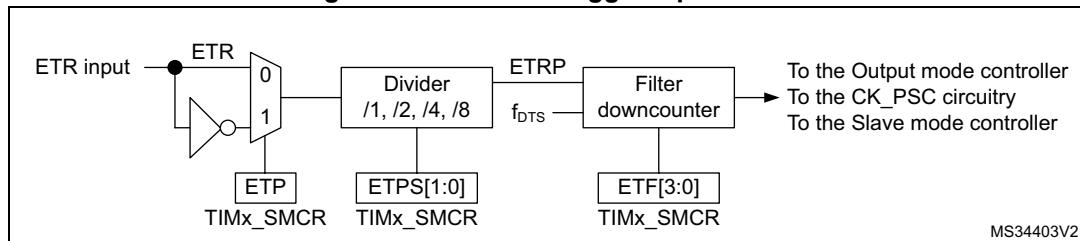
### 22.3.4 External trigger input

The timer features an external trigger input ETR. It can be used as:

- external clock (external clock mode 2, see [Section 22.3.5](#))
- trigger for the slave mode (see [Section 22.3.25](#))
- PWM reset input for cycle-by-cycle current regulation (see [Section 22.3.7](#))

[Figure 166](#) below describes the ETR input conditioning. The input polarity is defined with the ETP bit in TIMxSMCR register. The trigger can be prescaled with the divider programmed by the ETPS[1:0] bitfield and digitally filtered with the ETF[3:0] bitfield.

**Figure 166. External trigger input block**



### 22.3.5 Clock selection

The counter clock can be provided by the following clock sources:

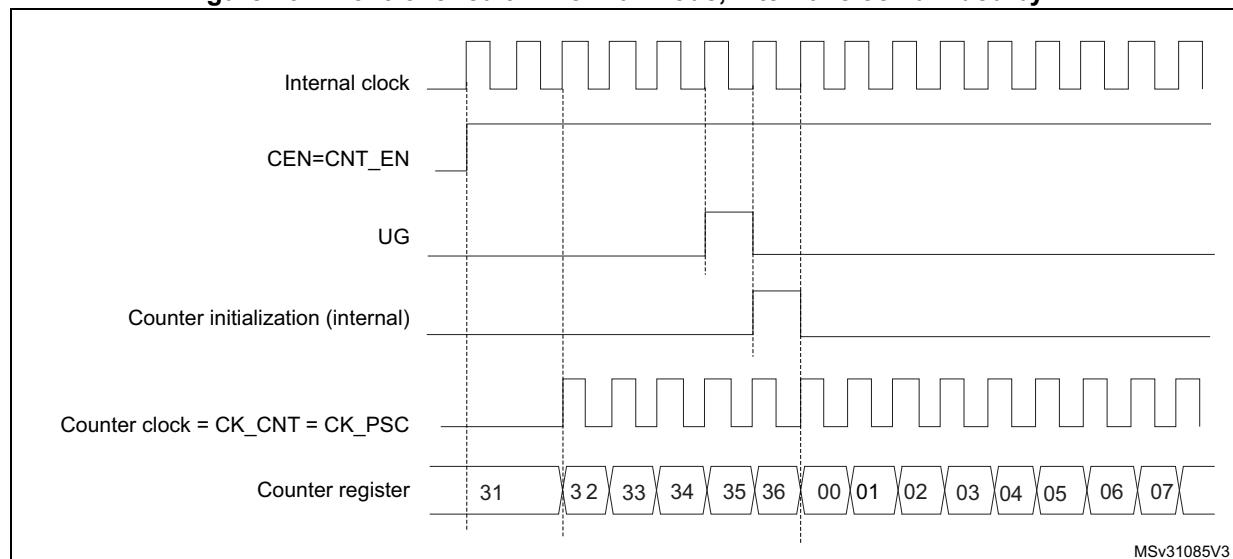
- Internal clock (CK\_INT)
- External clock mode1: external input pin
- External clock mode2: external trigger input ETR
- Encoder mode

#### Internal clock source (CK\_INT)

If the slave mode controller is disabled (SMS=000), then the CEN, DIR (in the TIMx\_CR1 register) and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

*Figure 167* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

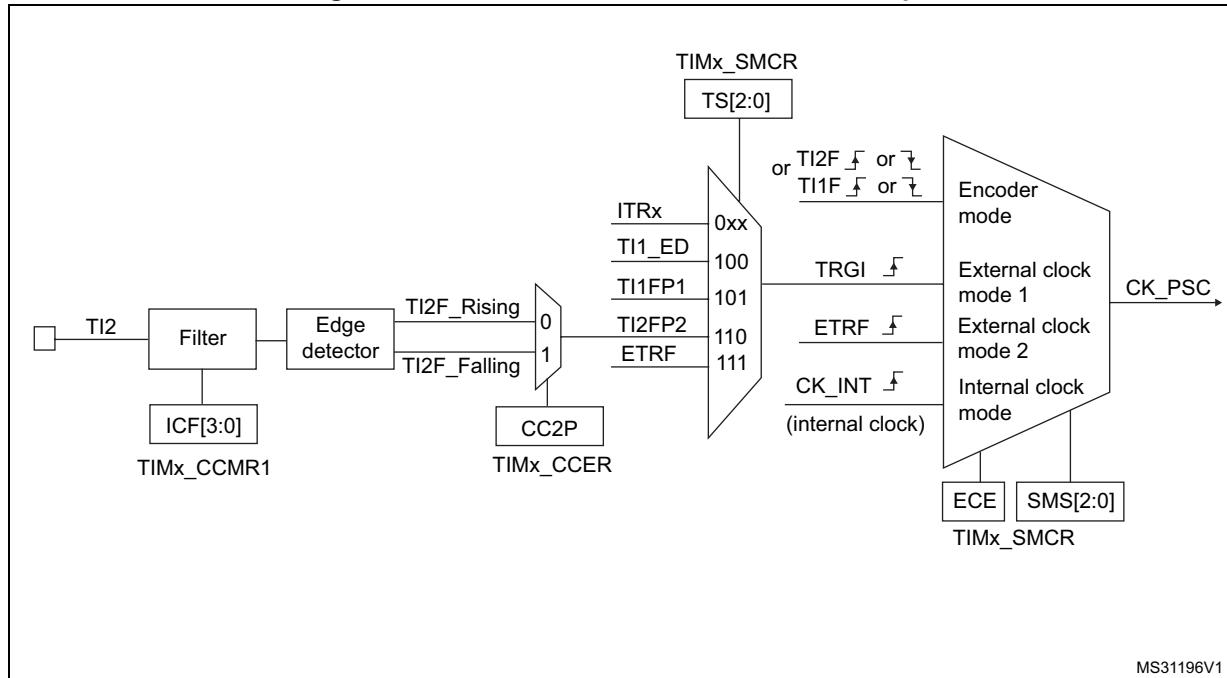
**Figure 167. Control circuit in normal mode, internal clock divided by 1**



#### 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 168. TI2 external clock connection example



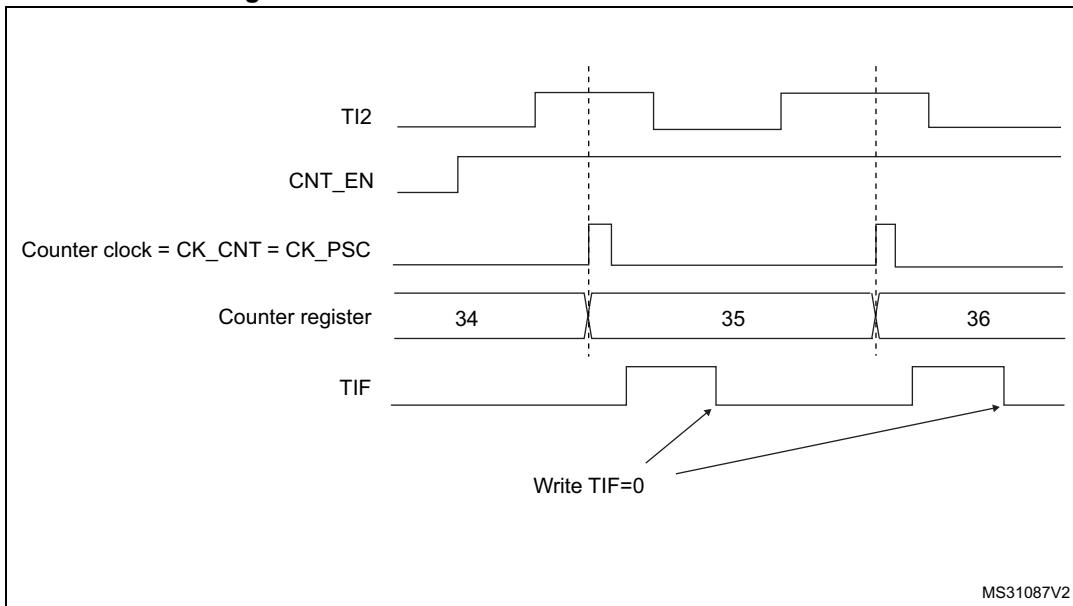
For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

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 the user does not need to configure it.

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

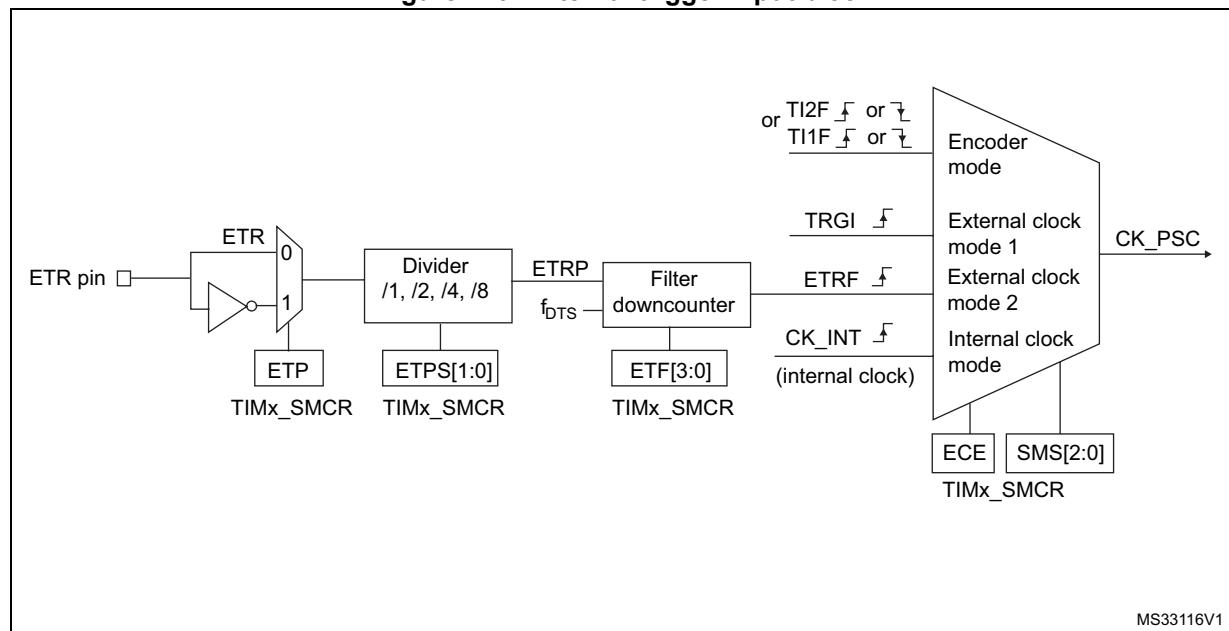
The delay between the rising edge on TI2 and the actual clock of the counter is due to the resynchronization circuit on TI2 input.

**Figure 169. Control circuit in external clock mode 1****External clock source mode 2**

This mode is selected by writing ECE=1 in the TIMx\_SMCR register.

The counter can count at each rising or falling edge on the external trigger input ETR.

The [Figure 170](#) gives an overview of the external trigger input block.

**Figure 170. External trigger input block**

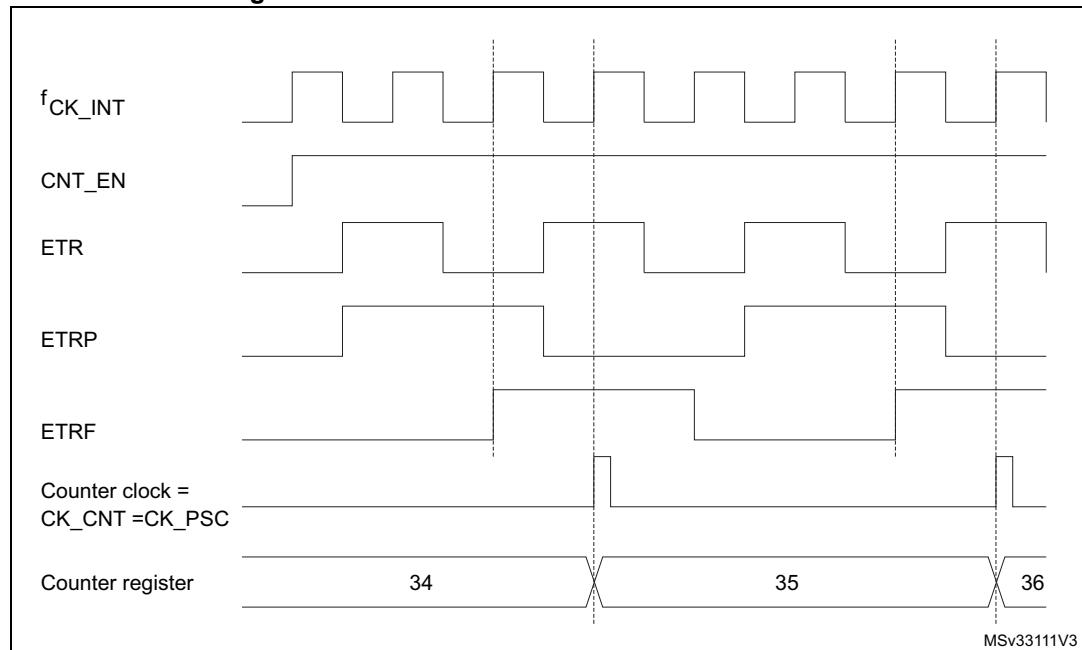
For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

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. As a consequence, the maximum frequency which can be correctly captured by the counter is at most  $\frac{1}{4}$  of TIMxCLK frequency. When the ETRP signal is faster, the user should apply a division of the external signal by proper ETPS prescaler setting.

**Figure 171. Control circuit in external clock mode 2**



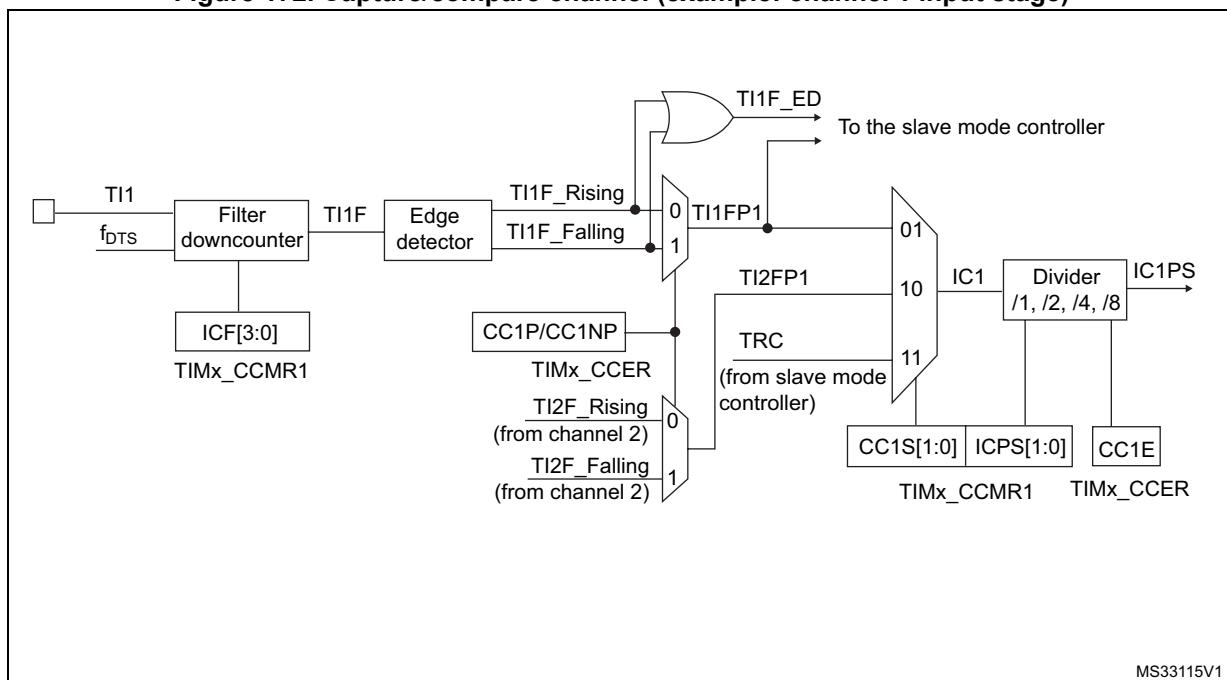
### **22.3.6 Capture/compare channels**

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), an input stage for capture (with digital filter, multiplexing, and prescaler, except for channels 5 and 6) and an output stage (with comparator and output control).

*Figure 172* to *Figure 175* 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 172. 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 173. Capture/compare channel 1 main circuit

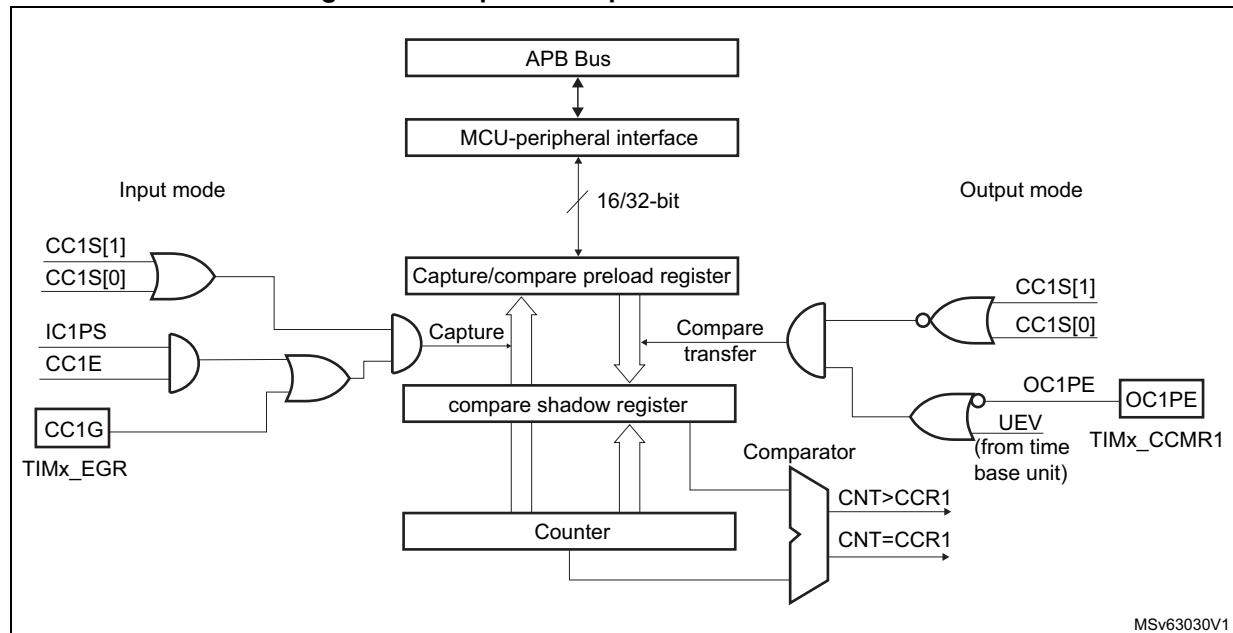
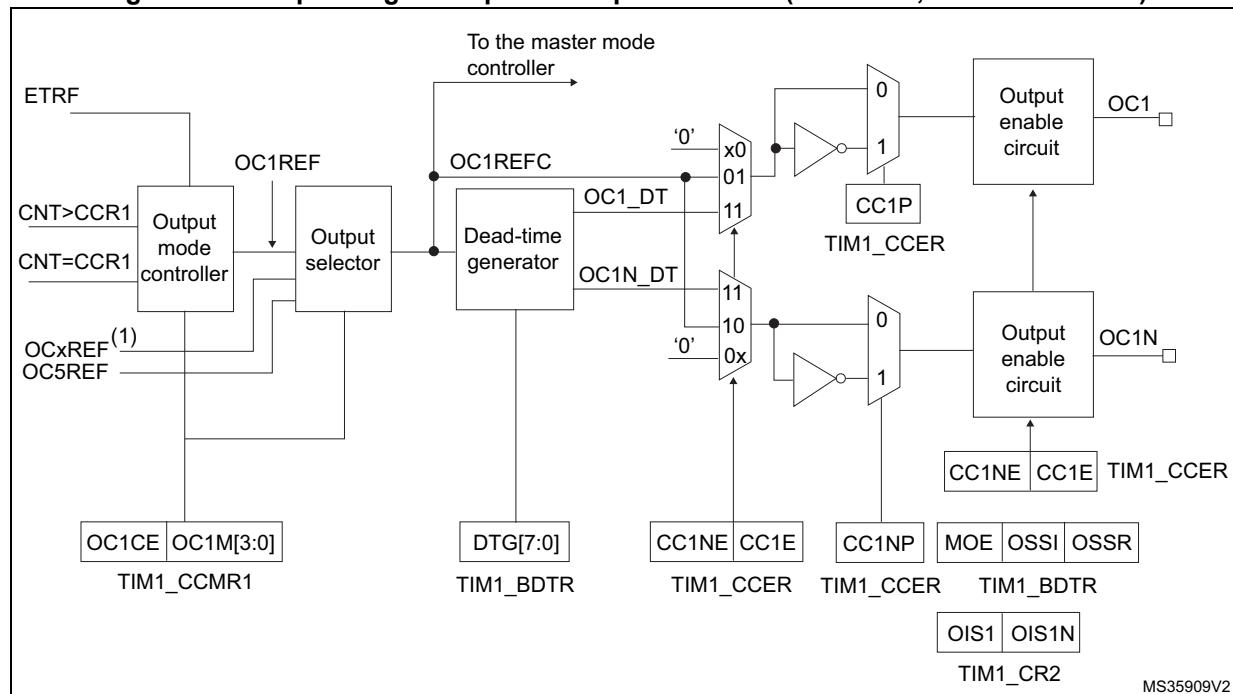
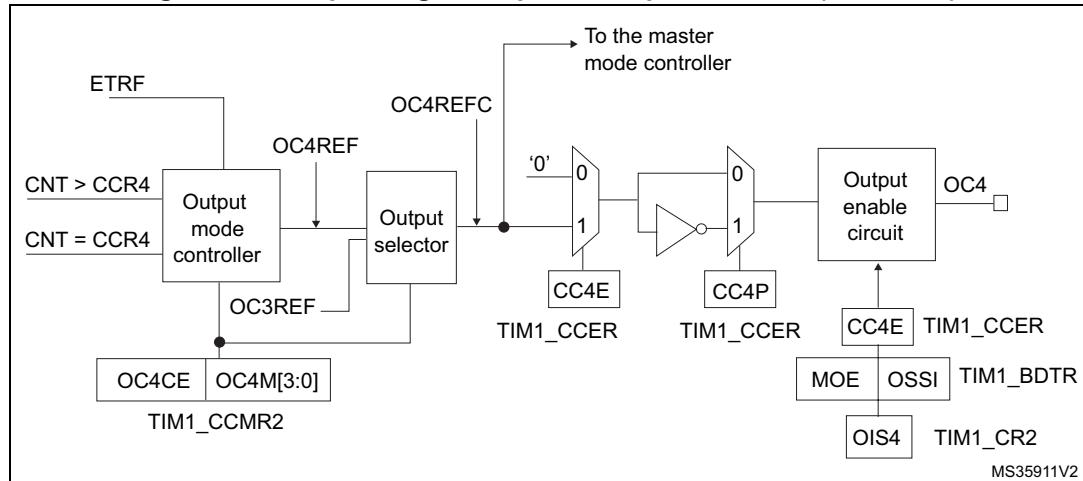
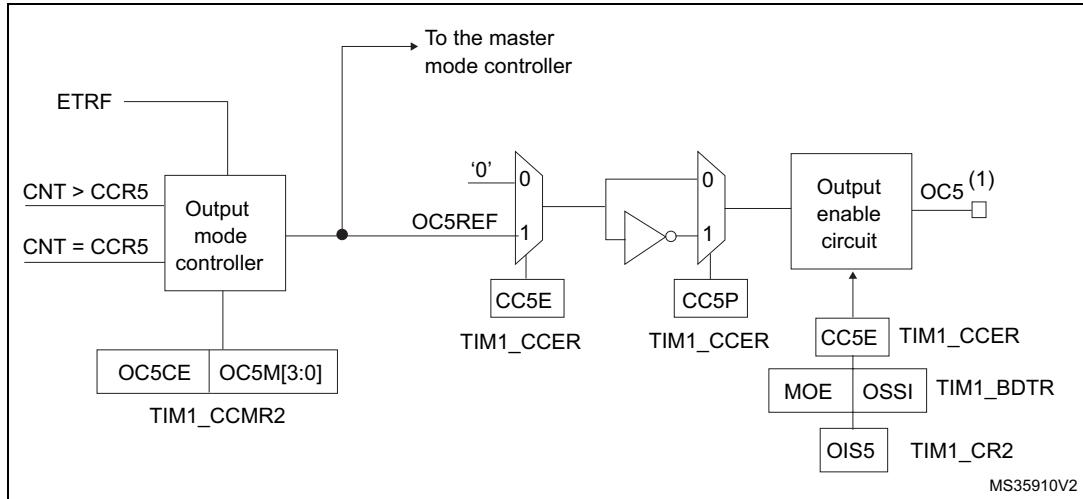


Figure 174. Output stage of capture/compare channel (channel 1, idem ch. 2 and 3)



1. OCxREF, where x is the rank of the complementary channel

**Figure 175. Output stage of capture/compare channel (channel 4)****Figure 176. Output stage of capture/compare channel (channel 5, idem ch. 6)**

1. Not available externally.

The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

### 22.3.7 Input capture mode

In Input capture mode, the Capture/Compare Registers (**TIMx\_CCRx**) are used to latch the value of the counter after a transition detected by the corresponding **ICx** signal. When a capture occurs, the corresponding **CCxIF** flag (**TIMx\_SR** register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the **CCxIF** flag was already high, then the over-capture flag **CCxOF** (**TIMx\_SR** register) is set. **CCxIF** can be

cleared by software by writing it to '0' or by reading the captured data stored in the TIMx\_CCRx register. CCxOF is cleared when written with '0'.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

1. Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx\_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx\_CCR1 register becomes read-only.
2. Program the appropriate input filter duration in relation with the signal connected to the timer (when the input is one of the TIx (ICxF bits in the TIMx\_CCMRx register). Let's imagine that, when toggling, the input signal is not stable during at most 5 internal clock cycles. We must program a filter duration longer than these 5 clock cycles. We can validate a transition on TI1 when 8 consecutive samples with the new level have been detected (sampled at  $f_{DTS}$  frequency). Then write IC1F bits to 0011 in the TIMx\_CCMR1 register.
3. Select the edge of the active transition on the TI1 channel by writing CC1P and CC1NP bits to 0 in the TIMx\_CCER register (rising edge in this case).
4. Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to '00' in the TIMx\_CCMR1 register).
5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx\_CCER register.
6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx\_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx\_DIER register.

When an input capture occurs:

- The TIMx\_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

*Note:*

*IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx\_EGR register.*

### 22.3.8 PWM input mode

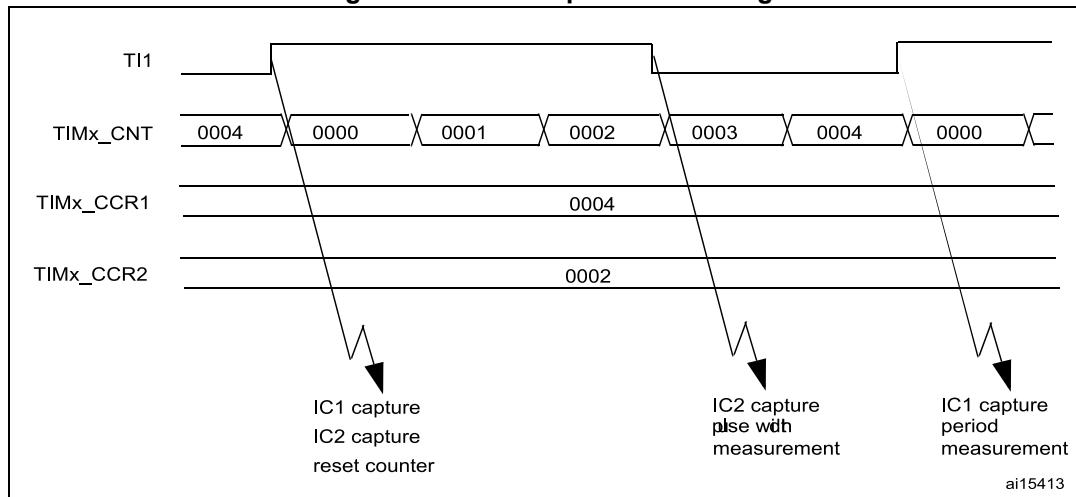
This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, the user can measure the period (in TIMx\_CCR1 register) and the duty cycle (in TIMx\_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK\_INT frequency and prescaler value):

1. Select the active input for TIMx\_CCR1: write the CC1S bits to 01 in the TIMx\_CCMR1 register (TI1 selected).
2. Select the active polarity for TI1FP1 (used both for capture in TIMx\_CCR1 and counter clear): write the CC1P and CC1NP bits to '0' (active on rising edge).
3. Select the active input for TIMx\_CCR2: write the CC2S bits to 10 in the TIMx\_CCMR1 register (TI1 selected).
4. Select the active polarity for TI1FP2 (used for capture in TIMx\_CCR2): write the CC2P and CC2NP bits to CC2P/CC2NP='10' (active on falling edge).
5. Select the valid trigger input: write the TS bits to 101 in the TIMx\_SMCR register (TI1FP1 selected).
6. Configure the slave mode controller in reset mode: write the SMS bits to 0100 in the TIMx\_SMCR register.
7. Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx\_CCER register.

**Figure 177. PWM input mode timing**



### 22.3.9 Forced output mode

In output mode (CCxS bits = 00 in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx/OCxN) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, user just needs to write 0101 in the OCxM bits in the corresponding TIMx\_CCMRx register. Thus OCXREF is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

For example: CCxP=0 (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to 0100 in the TIMx\_CCMRx register.

Anyway, the comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the output compare mode section below.

### 22.3.10 Output compare mode

This function is used to control an output waveform or indicate when a period of time has elapsed. Channels 1 to 4 can be output, while Channel 5 and 6 are only available inside the device (for instance, for compound waveform generation or for ADC triggering).

When a match is found between the capture/compare register and the counter, the output compare function:

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCXM=0000), be set active (OCXM=0001), be set inactive (OCXM=0010) or can toggle (OCXM=0011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx\_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx\_DIER register, CCDS bit in the TIMx\_CR2 register for the DMA request selection).

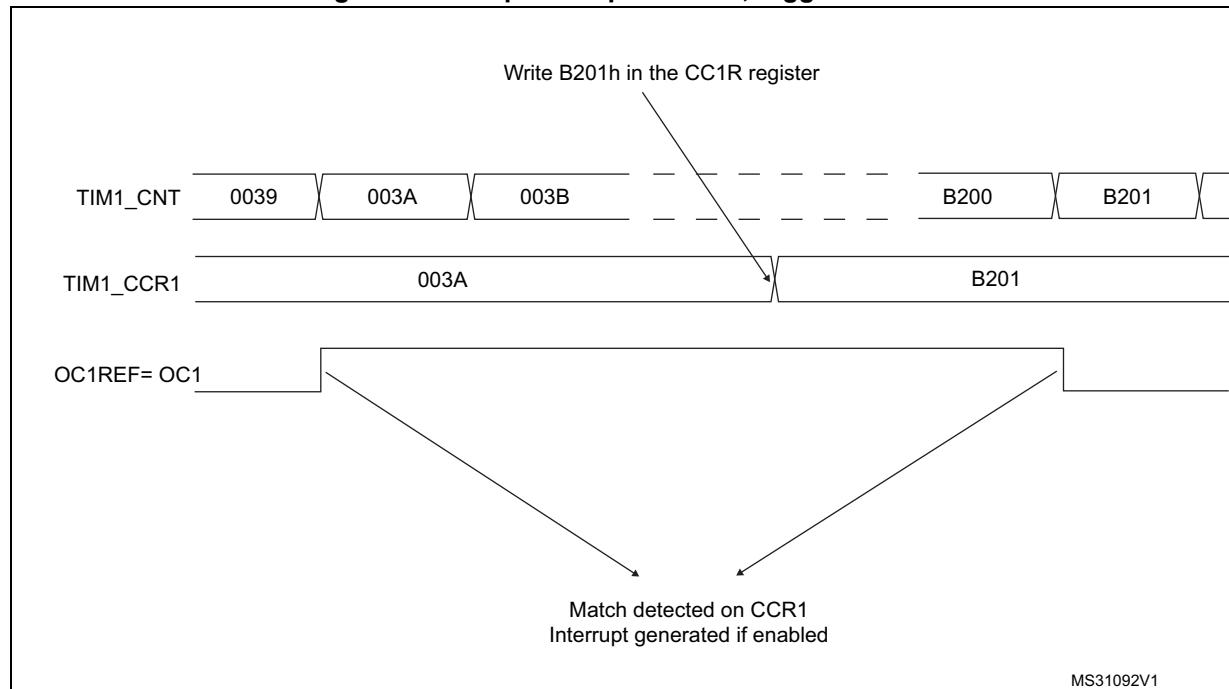
The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One Pulse mode).

#### Procedure

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx\_ARR and TIMx\_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
  - Write OCxM = 0011 to toggle OCx output pin when CNT matches CCRx
  - Write OCxPE = 0 to disable preload register
  - Write CCxP = 0 to select active high polarity
  - Write CCxE = 1 to enable the output
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

The TIMx\_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE='0', else TIMx\_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 178](#).

**Figure 178. Output compare mode, toggle on OC1**

### 22.3.11 PWM mode

Pulse Width Modulation mode allows a signal to be generated with a frequency determined by the value of the TIMx\_ARR register and a duty cycle determined by the value of the TIMx\_CC Rx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '0110' (PWM mode 1) or '0111' (PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register. The corresponding preload register must be enabled by setting the OCxPE bit in the TIMx\_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx\_CR1 register.

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

OCx polarity is software programmable using the CCxP bit in the TIMx\_CCER register. It can be programmed as active high or active low. OCx output is enabled by a combination of the CCxE, CCxNE, MOE, OSS1 and OSSR bits (TIMx\_CCER and TIMx\_BDTR registers). Refer to the TIMx\_CCER register description for more details.

In PWM mode (1 or 2), TIMx\_CNT and TIMx\_CC Rx are always compared to determine whether TIMx\_CC Rx  $\leq$  TIMx\_CNT or TIMx\_CNT  $\leq$  TIMx\_CC Rx (depending on the direction of the counter).

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx\_CR1 register.

### PWM edge-aligned mode

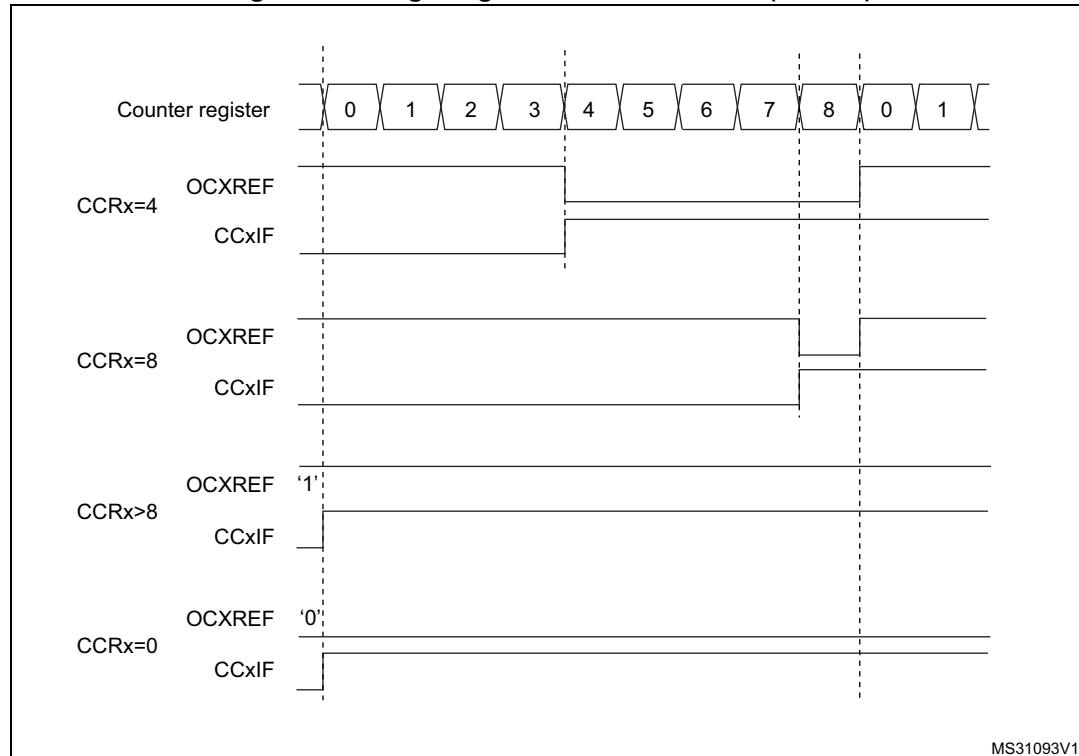
- Upcounting configuration

Upcounting is active when the DIR bit in the TIMx\_CR1 register is low. Refer to the [Upcounting mode on page 639](#).

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 179](#) shows some edge-aligned PWM waveforms in an example where  $\text{TIMx\_ARR}=8$ .

**Figure 179. Edge-aligned PWM waveforms (ARR=8)**



- Downcounting configuration

Downcounting is active when DIR bit in TIMx\_CR1 register is high. Refer to the [Downcounting mode on page 643](#)

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

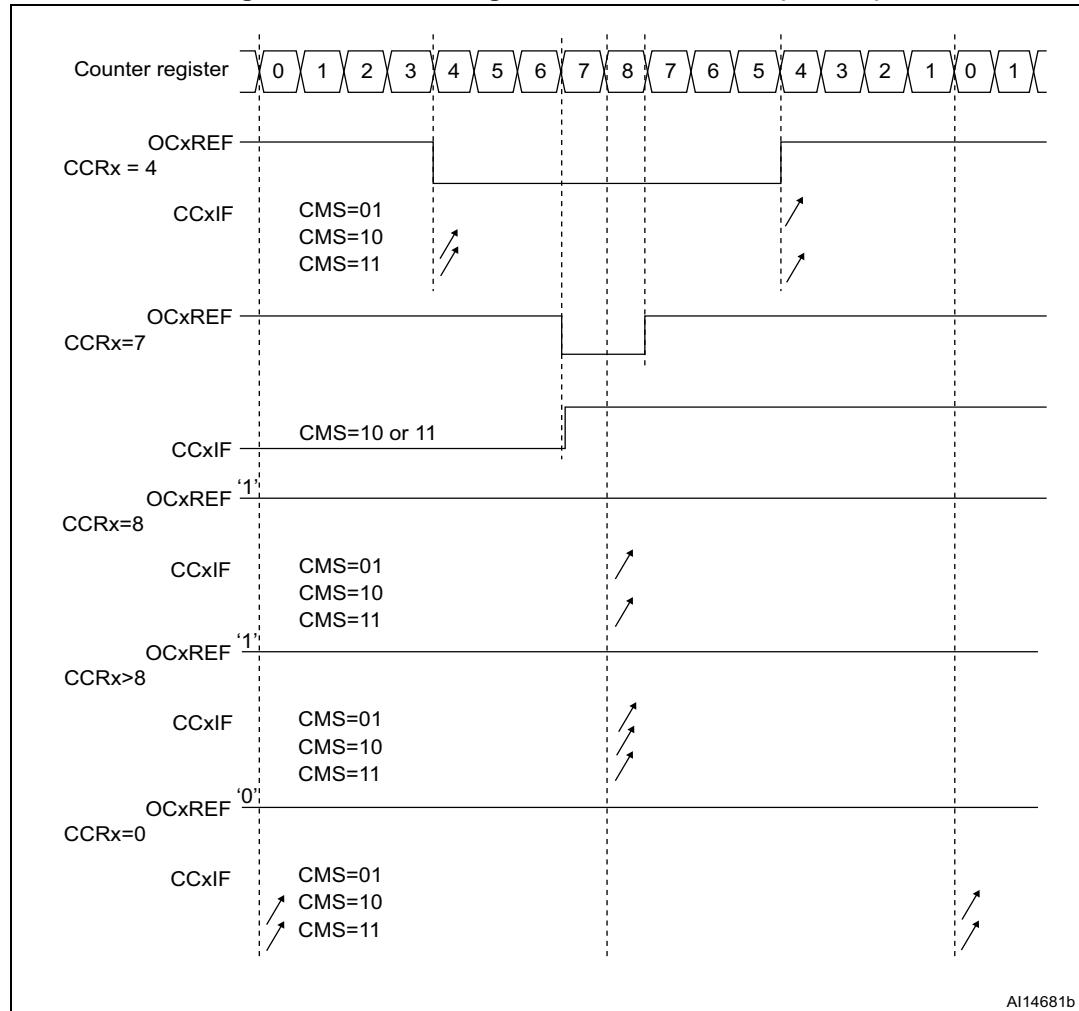
Center-aligned mode is active when the CMS bits in TIMx\_CR1 register are different from '00' (all the remaining configurations having the same effect on the OCxRef/OCx signals). The compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the 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 the [Center-aligned mode \(up/down counting\) on page 646](#).

[Figure 180](#) 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 180. Center-aligned PWM waveforms (ARR=8)**



AI14681b

#### Hints on using center-aligned mode

- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter counts up or down depending on the value written in the DIR bit

in the TIMx\_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.

- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
  - The direction is not updated if a value greater than the auto-reload value is written in the counter (TIMx\_CNT>TIMx\_ARR). For example, if the counter was counting up, it continues to count up.
  - The direction is updated if 0 or the TIMx\_ARR value is written in the counter but no Update Event UEV is generated.
- The safest way to use center-aligned mode is to generate an update by software (setting the UG bit in the TIMx\_EGR register) just before starting the counter and not to write the counter while it is running.

### 22.3.12 Asymmetric PWM mode

Asymmetric mode allows two center-aligned PWM signals to be generated with a programmable phase shift. While the frequency is determined by the value of the TIMx\_ARR register, the duty cycle and the phase-shift are determined by a pair of TIMx\_CCRx register. One register controls the PWM during up-counting, the second during down counting, so that PWM is adjusted every half PWM cycle:

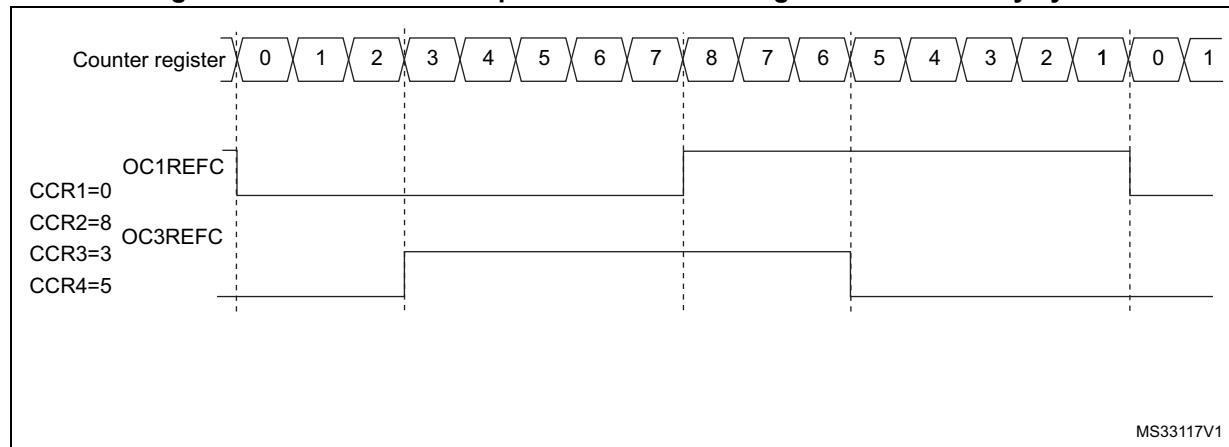
- OC1REFC (or OC2REFC) is controlled by TIMx\_CCR1 and TIMx\_CCR2
- OC3REFC (or OC4REFC) is controlled by TIMx\_CCR3 and TIMx\_CCR4

Asymmetric PWM mode can be selected independently on two channel (one OCx output per pair of CCR registers) by writing '1110' (Asymmetric PWM mode 1) or '1111' (Asymmetric PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register.

*Note:* The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.

When a given channel is used as asymmetric PWM channel, its complementary channel can also be used. For instance, if an OC1REFC signal is generated on channel 1 (Asymmetric PWM mode 1), it is possible to output either the OC2REF signal on channel 2, or an OC2REFC signal resulting from asymmetric PWM mode 1.

*Figure 181* represents an example of signals that can be generated using Asymmetric PWM mode (channels 1 to 4 are configured in Asymmetric PWM mode 1). Together with the deadtime generator, this allows a full-bridge phase-shifted DC to DC converter to be controlled.

**Figure 181. Generation of 2 phase-shifted PWM signals with 50% duty cycle**

### 22.3.13 Combined PWM mode

Combined PWM mode allows two edge or center-aligned PWM signals to be generated with programmable delay and phase shift between respective pulses. While the frequency is determined by the value of the TIMx\_ARR register, the duty cycle and delay are determined by the two TIMx\_CCRx registers. The resulting signals, OCxREFC, are made of an OR or AND logical combination of two reference PWMs:

- OC1REFC (or OC2REFC) is controlled by TIMx\_CCR1 and TIMx\_CCR2
- OC3REFC (or OC4REFC) is controlled by TIMx\_CCR3 and TIMx\_CCR4

Combined PWM mode can be selected independently on two channels (one OCx output per pair of CCR registers) by writing ‘1100’ (Combined PWM mode 1) or ‘1101’ (Combined PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register.

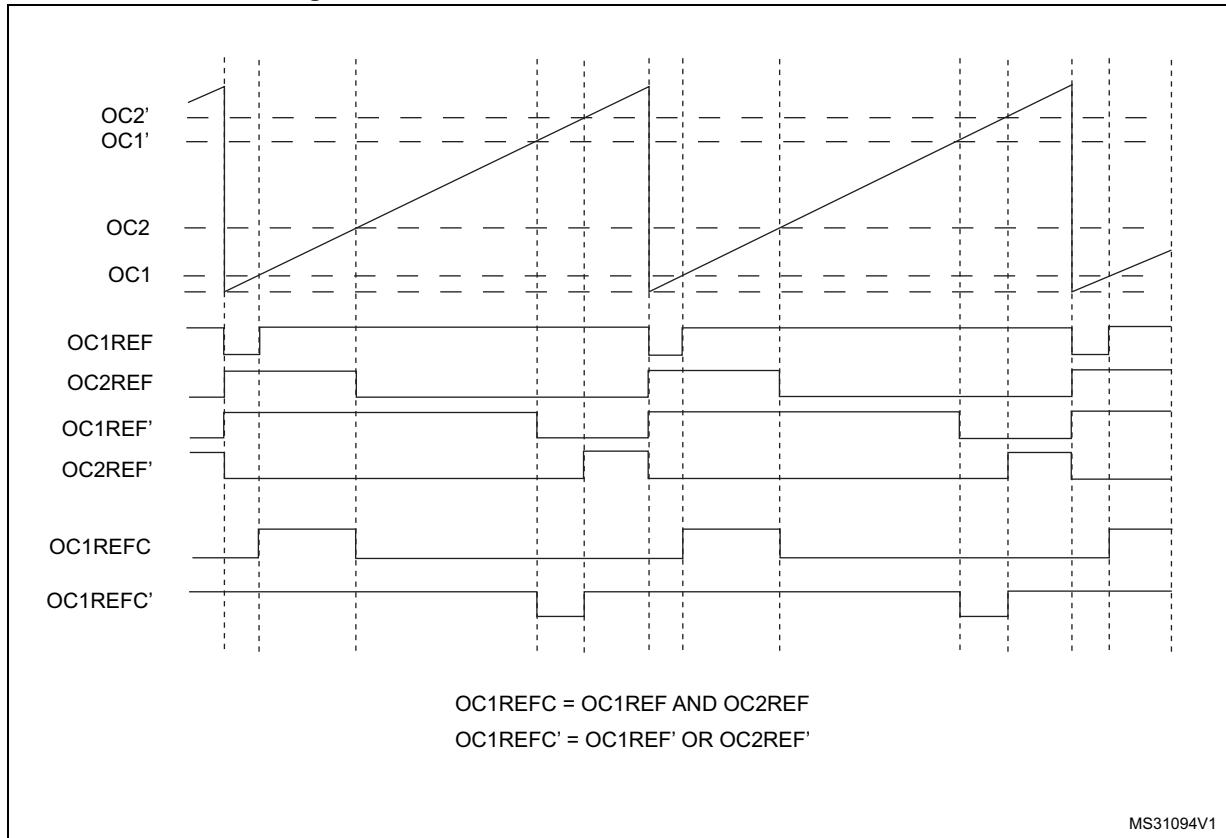
When a given channel is used as combined PWM channel, its complementary channel must be configured in the opposite PWM mode (for instance, one in Combined PWM mode 1 and the other in Combined PWM mode 2).

*Note:*

*The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.*

*Figure 182* represents an example of signals that can be generated using Asymmetric PWM mode, obtained with the following configuration:

- Channel 1 is configured in Combined PWM mode 2,
- Channel 2 is configured in PWM mode 1,
- Channel 3 is configured in Combined PWM mode 2,
- Channel 4 is configured in PWM mode 1.

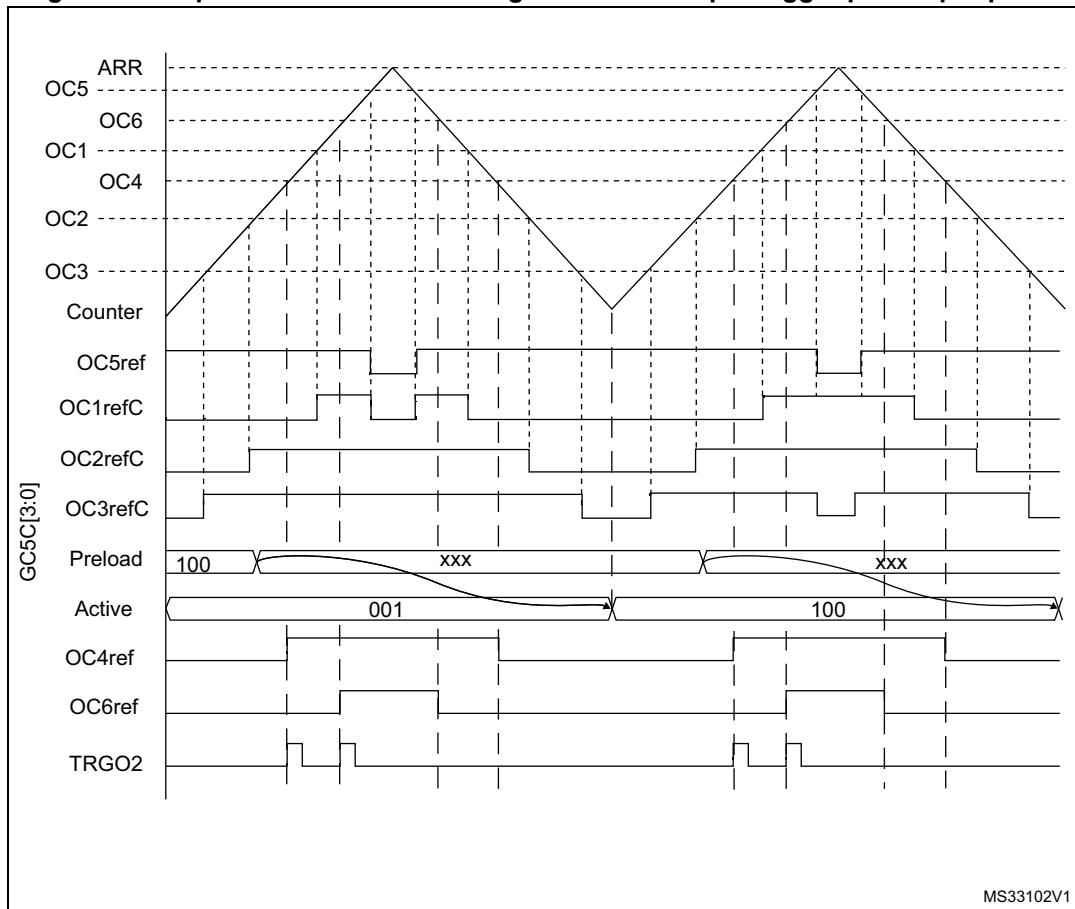
**Figure 182. Combined PWM mode on channel 1 and 3**

### 22.3.14 Combined 3-phase PWM mode

Combined 3-phase PWM mode allows one to three center-aligned PWM signals to be generated with a single programmable signal ANDed in the middle of the pulses. The OC5REF signal is used to define the resulting combined signal. The 3-bits GC5C[3:1] in the TIMx\_CCR5 allow selection on which reference signal the OC5REF is combined. The resulting signals, OCxREFC, are made of an AND logical combination of two reference PWMs:

- If GC5C1 is set, OC1REFC is controlled by TIMx\_CCR1 and TIMx\_CCR5
- If GC5C2 is set, OC2REFC is controlled by TIMx\_CCR2 and TIMx\_CCR5
- If GC5C3 is set, OC3REFC is controlled by TIMx\_CCR3 and TIMx\_CCR5

Combined 3-phase PWM mode can be selected independently on channels 1 to 3 by setting at least one of the 3-bits GC5C[3:1].

**Figure 183. 3-phase combined PWM signals with multiple trigger pulses per period**

The TRGO2 waveform shows how the ADC can be synchronized on given 3-phase PWM signals. Refer to [Section 22.3.26: ADC synchronization](#) for more details.

### 22.3.15 Complementary outputs and dead-time insertion

The advanced-control timers (TIM1/TIM8) can output two complementary signals and manage the switching-off and the switching-on instants of the outputs.

This time is generally known as dead-time and it has to be adjusted depending on the devices that are connected to the outputs and their characteristics (intrinsic delays of level-shifters, delays due to power switches...)

The polarity of the outputs (main output OC<sub>x</sub> or complementary OC<sub>xN</sub>) can be selected independently for each output. This is done by writing to the CC<sub>xP</sub> and CC<sub>xNP</sub> bits in the TIM<sub>x</sub>\_CCER register.

The complementary signals OC<sub>x</sub> and OC<sub>xN</sub> are activated by a combination of several control bits: the CC<sub>xE</sub> and CC<sub>xNE</sub> bits in the TIM<sub>x</sub>\_CCER register and the MOE, OIS<sub>x</sub>, OIS<sub>xN</sub>, OSS<sub>I</sub> and OSS<sub>R</sub> bits in the TIM<sub>x</sub>\_BDTR and TIM<sub>x</sub>\_CR2 registers. Refer to

[Table 150: Output control bits for complementary OC<sub>x</sub> and OC<sub>xN</sub> channels with break feature on page 714](#) for more details. In particular, the dead-time is activated when switching to the idle state (MOE falling down to 0).

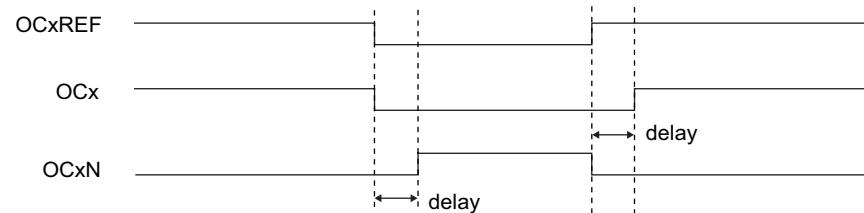
Dead-time insertion is enabled by setting both CCxE and CCxNE bits, and the MOE bit if the break circuit is present. There is one 10-bit dead-time generator for each channel. From a reference waveform OCxREF, it generates 2 outputs OCx and OCxN. If OCx and OCxN are active high:

- The OCx output signal is the same as the reference signal except for the rising edge, which is delayed relative to the reference rising edge.
- The OCxN output signal is the opposite of the reference signal except for the rising edge, which is delayed relative to the reference falling edge.

If the delay is greater than the width of the active output (OCx or OCxN) then the corresponding pulse is not generated.

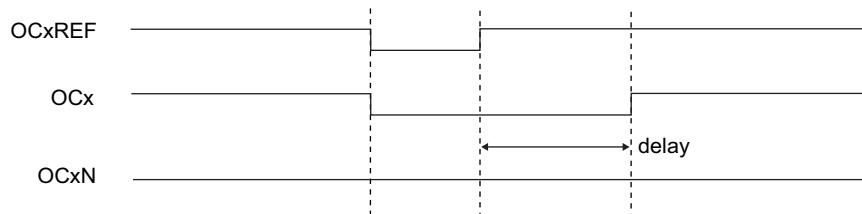
The following figures show the relationships between the output signals of the dead-time generator and the reference signal OCxREF. (we suppose CCxP=0, CCxNP=0, MOE=1, CCxE=1 and CCxNE=1 in these examples)

**Figure 184. Complementary output with dead-time insertion**

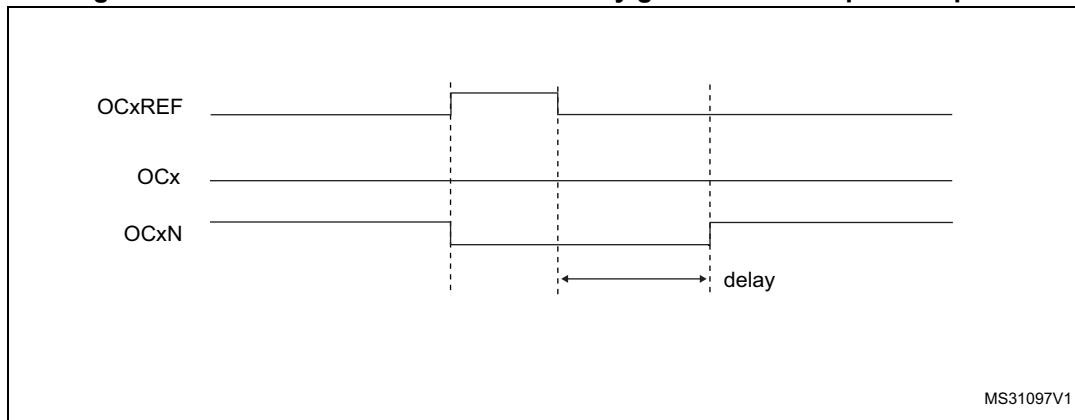


MS31095V1

**Figure 185. Dead-time waveforms with delay greater than the negative pulse**



MS31096V1

**Figure 186. 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 22.4.20: TIMx break and dead-time register \(TIMx\\_BDTR\)\(x = 1, 8\)](#) for delay calculation.

#### Re-directing OCxREF to OCx or OCxN

In output mode (forced, output compare or PWM), OCxREF can be re-directed to the OCx output or to OCxN output by configuring the CCxE and CCxNE bits in the TIMx\_CCER register.

This allows a specific waveform to be sent (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.

#### 22.3.16 Using the break function

The purpose of the break function is to protect power switches driven by PWM signals generated with the TIM1 and TIM8 timers. The two break inputs are usually connected to fault outputs of power stages and 3-phase inverters. When activated, the break circuitry shuts down the PWM outputs and forces them to a predefined safe state. A number of internal MCU events can also be selected to trigger an output shut-down.

The break features two channels. A break channel which gathers both system-level fault (clock failure,...) and application fault (from input pins), and can force the outputs to a predefined level (either active or inactive) after a deadtime duration. A break2 channel which only includes application faults and is able to force the outputs to an inactive state.

The output enable signal and output levels during break are depending on several control bits:

- the MOE bit in TIMx\_BDTR register allows the outputs to be enabled/disabled by software and is reset in case of break or break2 event.
- the OSS1 bit in the TIMx\_BDTR register defines whether the timer controls the output in inactive state or releases the control to the GPIO controller (typically to have it in Hi-Z mode)
- the OISx and OISxN bits in the TIMx\_CR2 register which are setting the output shut-down level, either active or inactive. The OCx and OCxN outputs cannot be set both to active level at a given time, whatever the OISx and OISxN values.  
Refer to [Table 150: Output control bits for complementary OCx and OCxN channels with break feature on page 714](#) for more details.

When exiting from reset, the break circuit is disabled and the MOE bit is low. The break functions can be enabled by setting the BKE and BK2E bits in the TIMx\_BDTR register. The break input polarities can be selected by configuring the BKP and BK2P bits in the same register. BKE/BK2E and BKP/BK2P can be modified at the same time. When the BKE/BK2E and BKP/BK2P bits are written, a delay of 1 APB clock cycle is applied before the writing is effective. Consequently, it is necessary to wait 1 APB clock period to correctly read back the bit after the write operation.

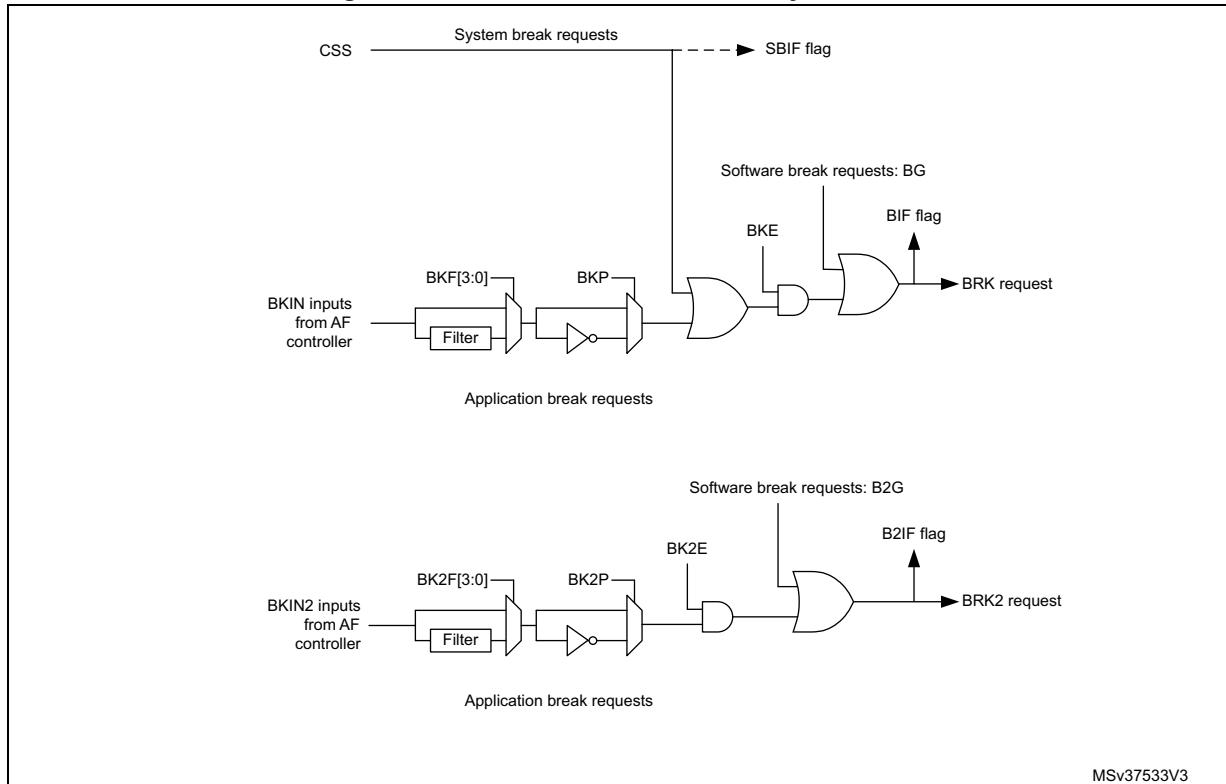
Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIMx\_BDTR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if MOE is set to 1 whereas it was low, a delay must be inserted (dummy instruction) before reading it correctly. This is because the write acts on the asynchronous signal whereas the read reflects the synchronous signal.

The break (BRK) event can be generated by two sources of events ORed together:

- An external source connected to one of the BKIN pin (as per selection done in the GPIO alternate function registers)
- An internal source: clock failure event generated by the CSS detector

The break2 (BRK2) can be generated by an external source connected to one of the BKIN2 pin (as per selection done in the GPIO alternate function registers).

Break events can also be generated by software using BG and B2G bits in the TIMx\_EGR register. The software break generation using BG and B2G is active whatever the BKE and BK2E enable bits values.

**Figure 187. Break and Break2 circuitry overview**

MSv37533V3

**Note:** An asynchronous (clockless) operation is only guaranteed when the programmable filter is disabled. If it is enabled, a fail safe clock mode (for example by using the internal PLL and/or the CSS) must be used to guarantee that break events are handled.

When one of the breaks occurs (selected level on one of the break inputs):

- The MOE bit is cleared asynchronously, putting the outputs in inactive state, idle state or even releasing the control to the GPIO controller (selected by the OSS1 bit). This feature is enabled even if the MCU oscillator is off.
- Each output channel is driven with the level programmed in the OISx bit in the TIMx\_CR2 register as soon as MOE=0. If OSS1=0, the timer releases the output control (taken over by the GPIO controller), otherwise the enable output remains high.
- When complementary outputs are used:
  - The outputs are first put in inactive state (depending on the polarity). This is done asynchronously so that it works even if no clock is provided to the timer.
  - If the timer clock is still present, then the dead-time generator is reactivated in order to drive the outputs with the level programmed in the OISx and OISxN bits after a dead-time. Even in this case, OCx and OCxN cannot be driven to their

active level together. Note that because of the resynchronization on MOE, the dead-time duration is slightly longer than usual (around 2 ck\_tim clock cycles).

- If OSS1=0, the timer releases the output control (taken over by the GPIO controller which forces a Hi-Z state), otherwise the enable outputs remain or become high as soon as one of the CCxE or CCxNE bits is high.
- The break status flag (SBIF, BIF and B2IF bits in the TIMx\_SR register) is set. An interrupt is generated if the BIE bit in the TIMx\_DIER register is set.
- If the AOE bit in the TIMx\_BDTR register is set, the MOE bit is automatically set again at the next update event (UEV). As an example, this can be used to perform a regulation. Otherwise, MOE remains low until the application sets it to '1' again. In this case, it can be used for security and the break input can be connected to an alarm from power drivers, thermal sensors or any security components.

**Note:** *If the MOE is reset by the CPU while the AOE bit is set, the outputs are in idle state and forced to inactive level or Hi-Z depending on OSS1 value.*

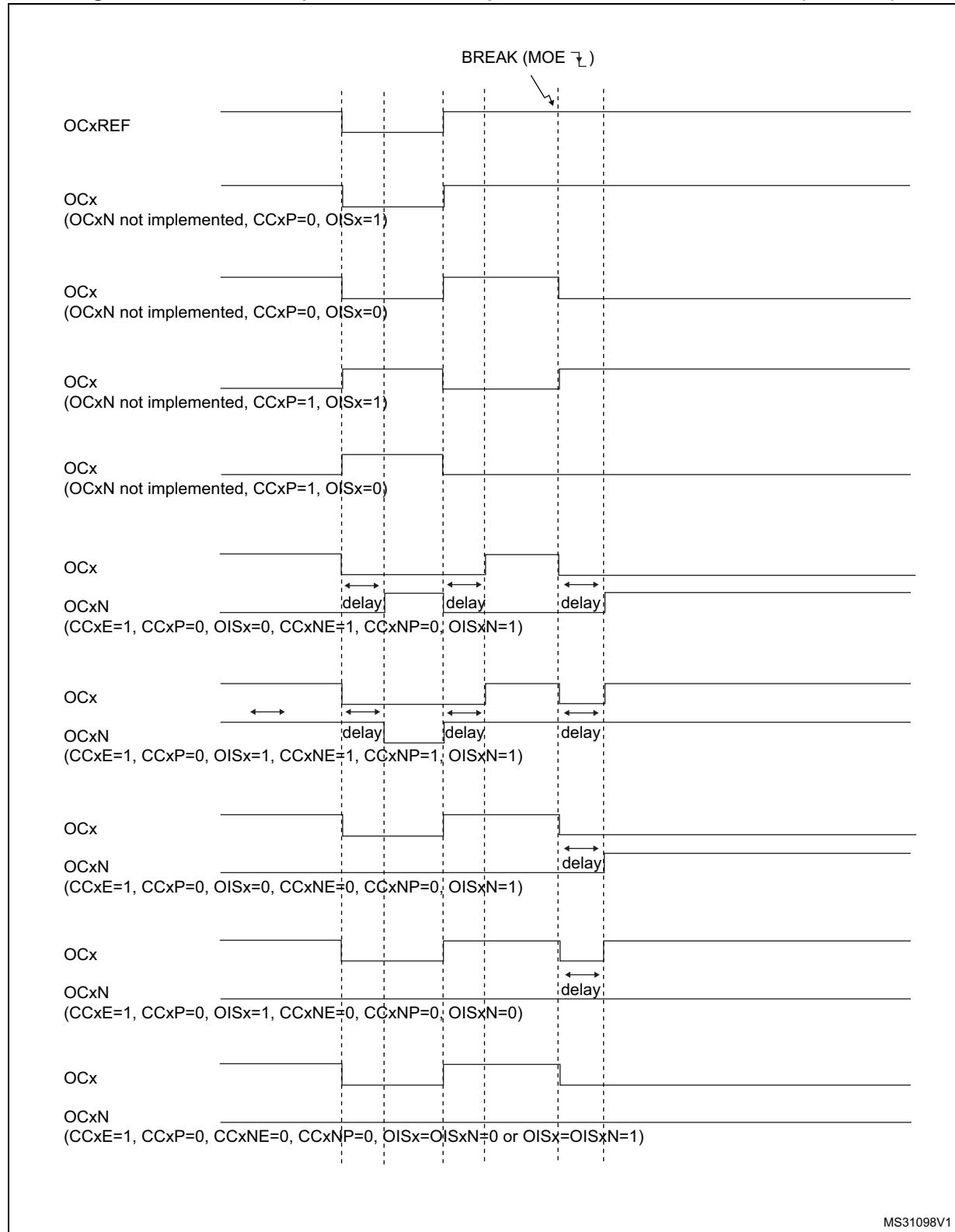
*If both the MOE and AOE bits are reset by the CPU, the outputs are in disabled state and driven with the level programmed in the OISx bit in the TIMx\_CR2 register.*

**Note:** *The break inputs are active on level. Thus, the MOE cannot be set while the break input is active (neither automatically nor by software). In the meantime, the status flag BIF and B2IF cannot be cleared.*

In addition to the break input and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows the configuration of several parameters to be freezed (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). The application can choose from 3 levels of protection selected by the LOCK bits in the TIMx\_BDTR register. Refer to [Section 22.4.20: TIMx break and dead-time register \(TIMx\\_BDTR\)\(x = 1, 8\)](#). The LOCK bits can be written only once after an MCU reset.

*Figure 188* shows an example of behavior of the outputs in response to a break.

Figure 188. Various output behavior in response to a break event on BRK (OSSI = 1)



MS31098V1

The two break inputs have different behaviors on timer outputs:

- The BRK input can either disable (inactive state) or force the PWM outputs to a predefined safe state.
- BRK2 can only disable (inactive state) the PWM outputs.

The BRK has a higher priority than BRK2 input, as described in [Table 147](#).

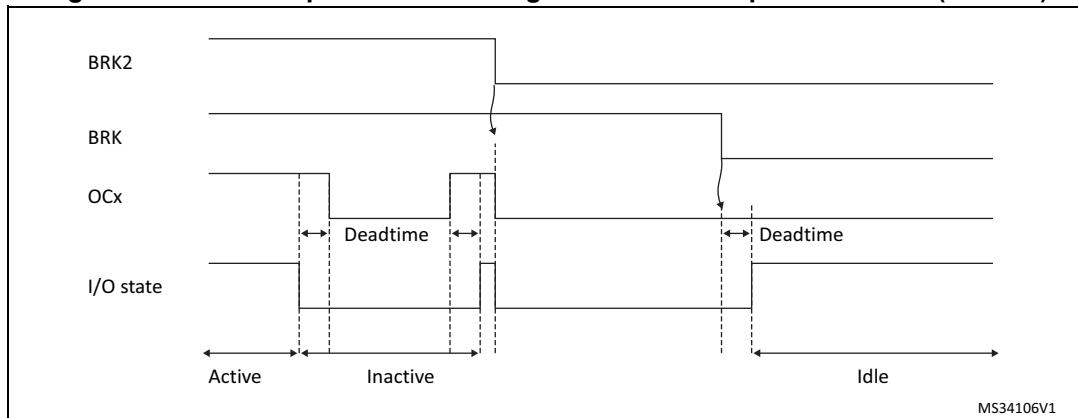
*Note:* *BRK2 must only be used with OSSR = OSSI = 1.*

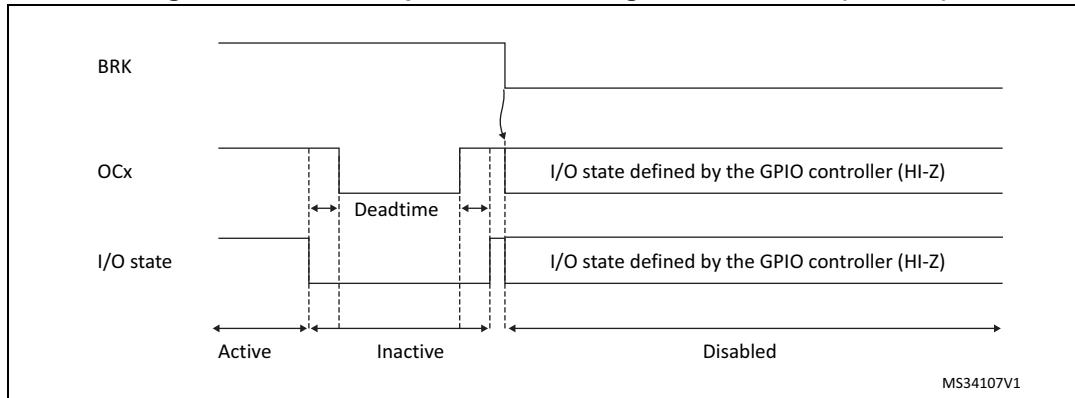
**Table 147. Behavior of timer outputs versus BRK/BRK2 inputs**

BRK	BRK2	Timer outputs state	Typical use case	
			OCxN output (low side switches)	OCx output (high side switches)
Active	X	<ul style="list-style-type: none"> <li>– Inactive then forced output state (after a deadtime)</li> <li>– Outputs disabled if OSSI = 0 (control taken over by GPIO logic)</li> </ul>	ON after deadtime insertion	OFF
Inactive	Active	Inactive	OFF	OFF

[Figure 189](#) gives an example of OCx and OCxN output behavior in case of active signals on BRK and BRK2 inputs. In this case, both outputs have active high polarities (CCxP = CCxNP = 0 in TIMx\_CCER register).

**Figure 189. PWM output state following BRK and BRK2 pins assertion (OSSI=1)**



**Figure 190. PWM output state following BRK assertion (OSSI=0)**

### 22.3.17 Clearing the OCxREF signal on an external event

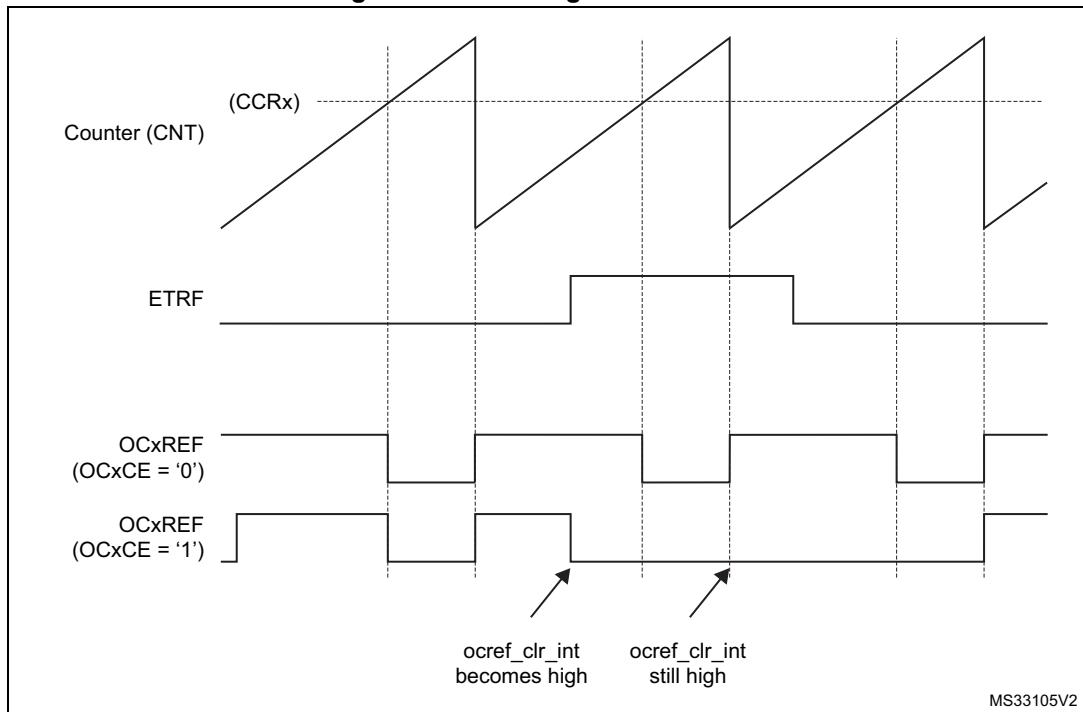
The OCxREF signal for a given channel can be driven low by applying a high level to the ETRF input (OCxCE enable bit of the corresponding TIMx\_CCMRx register set to '1'). The OCxREF signal remains low until the next transition to the active state, on the following PWM cycle.

When ETRF is chosen, ETR must be configured as follows:

1. The External Trigger Prescaler should be kept off: bits ETPS[1:0] of the TIMx\_SMCR register set to '00'.
2. The external clock mode 2 must be disabled: bit ECE of the TIMx\_SMCR register set to '0'.
3. The External Trigger Polarity (ETP) and the External Trigger Filter (ETF) can be configured according to the user needs.

*Figure 191* shows the behavior of the OCxREF signal when the ETRF Input becomes High, for both values of the enable bit OCxCE. In this example, the timer TIMx is programmed in PWM mode.

Figure 191. Clearing TIMx OCxREF



Note:

*In case of a PWM with a 100% duty cycle (if CCR<sub>x</sub>>ARR), then OCxREF is enabled again at the next counter overflow.*

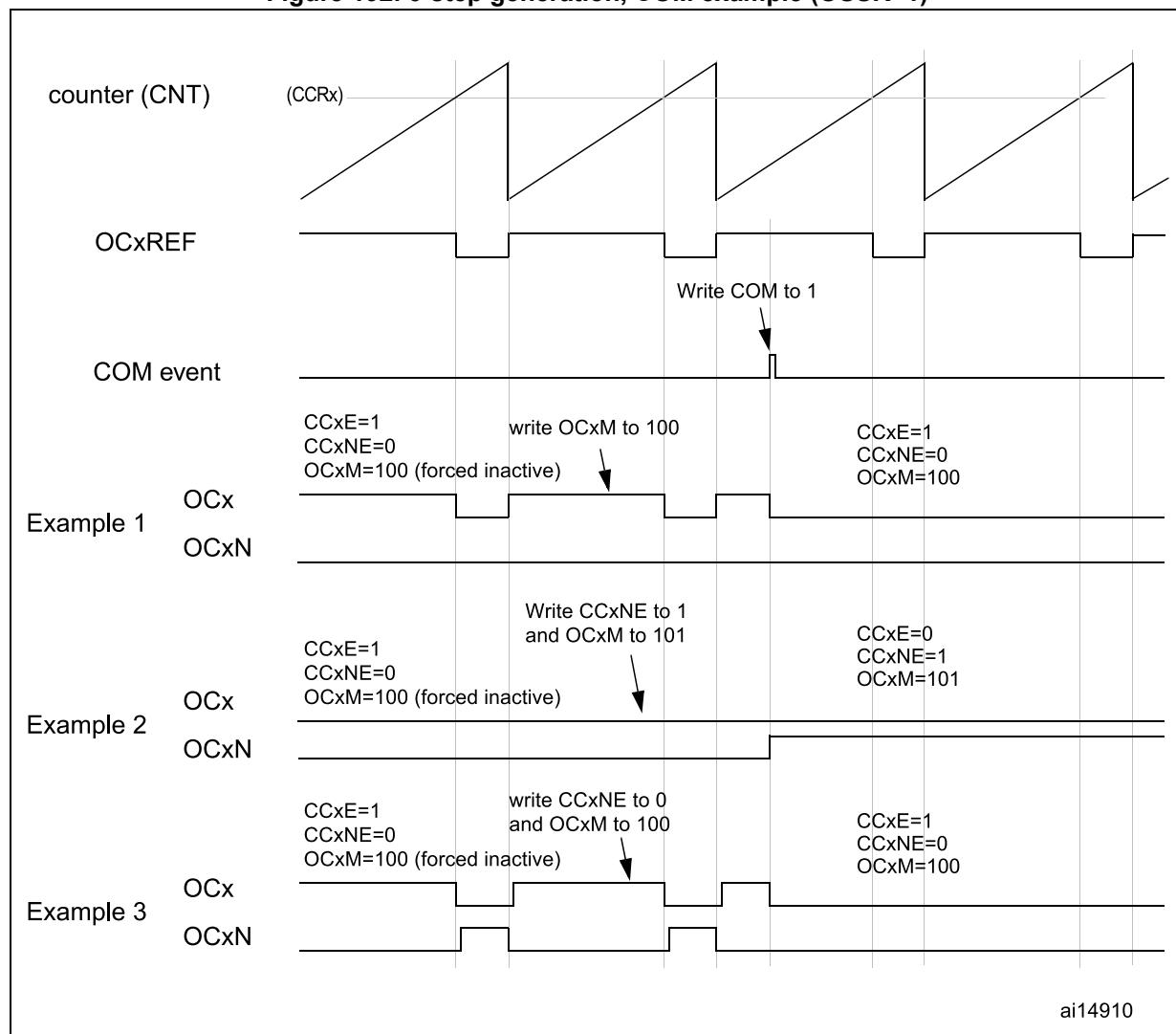
### 22.3.18 6-step PWM generation

When complementary outputs are used on a channel, preload bits are available on the OCxM, CCxE and CCxNE bits. The preload bits are transferred to the shadow bits at the COM commutation event. Thus one can program in advance the configuration for the next step and change the configuration of all the channels at the same time. COM can be generated by software by setting the COM bit in the TIMx\_EGR register or by hardware (on TRGI rising edge).

A flag is set when the COM event occurs (COMIF bit in the TIMx\_SR register), which can generate an interrupt (if the COMIE bit is set in the TIMx\_DIER register) or a DMA request (if the COMDE bit is set in the TIMx\_DIER register).

The [Figure 192](#) describes the behavior of the OCx and OCxN outputs when a COM event occurs, in 3 different examples of programmed configurations.

**Figure 192. 6-step generation, COM example (OSSR=1)**



### 22.3.19 One-pulse mode

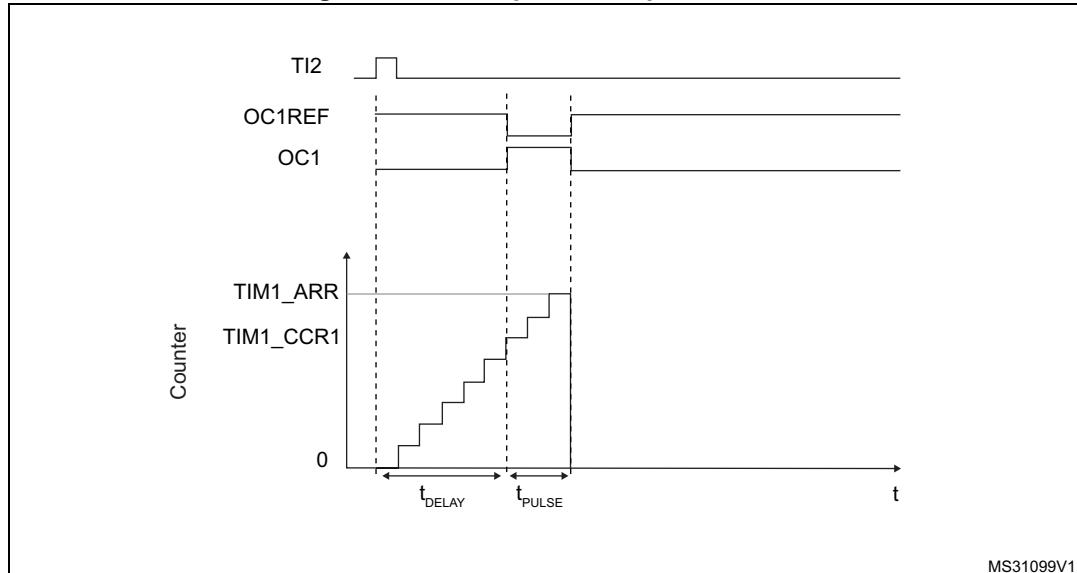
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. One-pulse mode is selected by setting the OPM bit in the TIMx\_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

- In upcounting: CNT < CCRx  $\leq$  ARR (in particular, 0 < CCRx)
- In downcounting: CNT > CCRx

**Figure 193. Example of one pulse mode.**



For example one may want to generate a positive pulse on OC1 with a length of  $t_{PULSE}$  and after a delay of  $t_{DELAY}$  as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

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 one want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this PWM mode 2 must be enabled by writing OC1M=111 in the TIMx\_CCMR1 register. Optionally the preload registers can be enabled by writing OC1PE='1' in the TIMx\_CCMR1 register and ARPE in the TIMx\_CR1 register. In this case one has to write the compare value in the TIMx\_CCR1 register, the auto-reload value in the TIMx\_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

In our example, the DIR and CMS bits in the TIMx\_CR1 register should be low.

Since only 1 pulse (Single mode) is needed, a 1 must be written in the OPM bit in the TIMx\_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0). When OPM bit in the TIMx\_CR1 register is set to '0', so the Repetitive Mode is selected.

Particular case: OCx fast enable:

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay  $t_{DELAY}$  min we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set in the TIMx\_CCMRx register. Then OCxRef (and OCx) are forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

### 22.3.20 Retriggerable one pulse mode

This mode allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length, but with the following differences with Non-retriggerable one pulse mode described in [Section 22.3.19](#):

- The pulse starts as soon as the trigger occurs (no programmable delay)
- The pulse is extended if a new trigger occurs before the previous one is completed

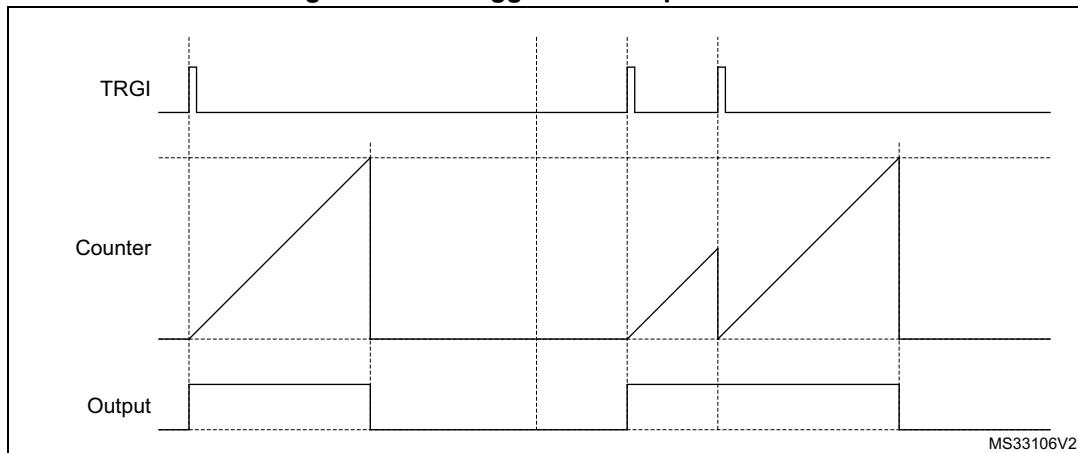
The timer must be in Slave mode, with the bits SMS[3:0] = '1000' (Combined Reset + trigger mode) in the TIMx\_SMCR register, and the OCxM[3:0] bits set to '1000' or '1001' for retriggerable OPM mode 1 or 2.

If the timer is configured in Up-counting mode, the corresponding CCRx must be set to 0 (the ARR register sets the pulse length). If the timer is configured in Down-counting mode, CCRx must be above or equal to ARR.

**Note:** The OCxM[3:0] and SMS[3:0] bit fields are split into two parts for compatibility reasons, the most significant bit are not contiguous with the 3 least significant ones.

*This mode must not be used with center-aligned PWM modes. It is mandatory to have CMS[1:0] = 00 in TIMx\_CR1.*

Figure 194. Retriggerable one pulse mode



### 22.3.21 Encoder interface mode

To select Encoder Interface mode write SMS='001' in the TIMx\_SMCR register if the counter is counting on TI2 edges only, SMS='010' if it is counting on TI1 edges only and SMS='011' if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx\_CCER register. When needed, the input filter can be programmed as well. CC1NP and CC2NP must be kept low.

The two inputs TI1 and TI2 are used to interface to a quadrature encoder. Refer to [Table 148](#). The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx\_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx\_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx\_ARR register (0 to ARR or ARR down to 0 depending on the direction). So the TIMx\_ARR must be configured before starting. In the same way, the capture, compare, repetition counter, trigger output features continue to work as normal. Encoder mode and External clock mode 2 are not compatible and must not be selected together.

*Note:*

*The prescaler must be set to zero when encoder mode is enabled*

In this mode, the counter is modified automatically following the speed and the direction of the quadrature encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 do not switch at the same time.

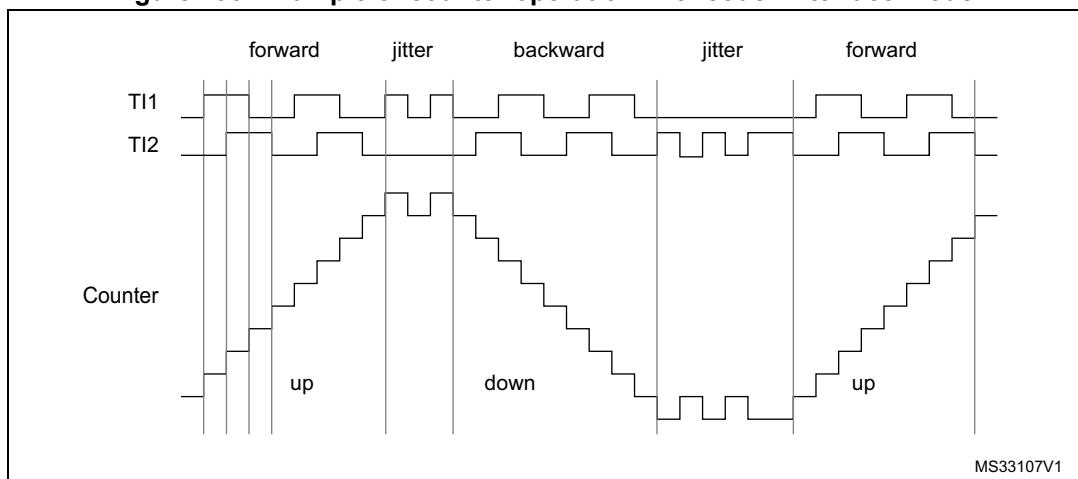
**Table 148. Counting direction versus encoder signals**

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

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

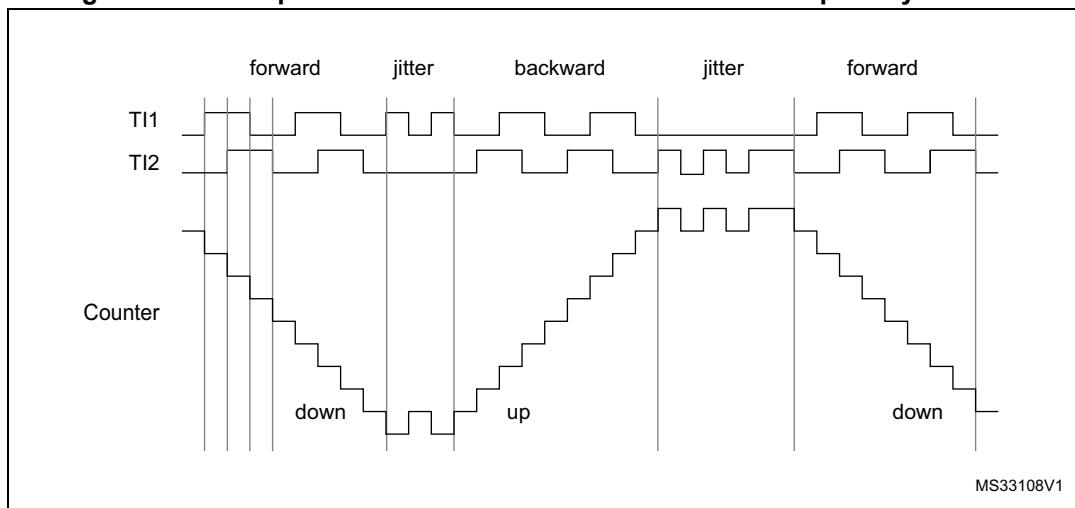
The [Figure 195](#) gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S='01' (TIMx\_CCMR1 register, TI1FP1 mapped on TI1).
- CC2S='01' (TIMx\_CCMR1 register, TI2FP2 mapped on TI2)
- CC1P='0' and CC1NP='0' (TIMx\_CCER register, TI1FP1 non-inverted, TI1FP1=TI1).
- CC2P='0' and CC2NP='0' (TIMx\_CCER register, TI1FP2 non-inverted, TI1FP2=TI2).
- SMS='011' (TIMx\_SMCR register, both inputs are active on both rising and falling edges).
- CEN='1' (TIMx\_CR1 register, Counter enabled).

**Figure 195. Example of counter operation in encoder interface mode.**

*Figure 196* gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P='1').

**Figure 196. Example of encoder interface mode with TI1FP1 polarity inverted.**



The timer, when configured in Encoder Interface mode provides information on the sensor's current position. Dynamic information can be obtained (speed, acceleration, deceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. This can be done by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer). When available, it is also possible to read its value through a DMA request.

The IUFREMAP bit in the TIMx\_CR1 register forces a continuous copy of the update interrupt flag (UIF) into the timer counter register's bit 31 (TIMxCNT[31]). This allows both the counter value and a potential roll-over condition signaled by the UIFCPY flag to be read in an atomic way. It eases the calculation of angular speed by avoiding race conditions caused, for instance, by a processing shared between a background task (counter reading) and an interrupt (update interrupt).

There is no latency between the UIF and UIFCPY flag assertions.

In 32-bit timer implementations, when the IUFREMAP bit is set, bit 31 of the counter is overwritten by the UIFCPY flag upon read access (the counter's most significant bit is only accessible in write mode).

### 22.3.22 UIF bit remapping

The IUFREMAP bit in the TIMx\_CR1 register forces a continuous copy of the Update Interrupt Flag UIF into the timer counter register's bit 31 (TIMxCNT[31]). This allows both the counter value and a potential roll-over condition signaled by the UIFCPY flag to be read in an atomic way. In particular cases, it can ease the calculations by avoiding race conditions, caused for instance by a processing shared between a background task (counter reading) and an interrupt (Update Interrupt).

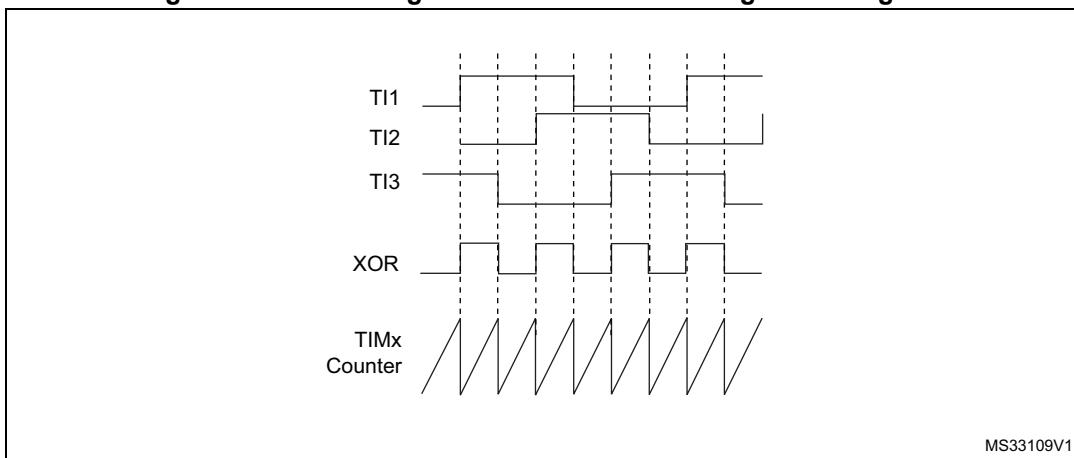
There is no latency between the UIF and UIFCPY flags assertion.

### 22.3.23 Timer input XOR function

The TI1S bit in the TIMx\_CR2 register, allows the input filter of channel 1 to be connected to the output of an XOR gate, combining the three input pins TIMx\_CH1, TIMx\_CH2 and TIMx\_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture. It is convenient to measure the interval between edges on two input signals, as per [Figure 197](#) below.

**Figure 197. Measuring time interval between edges on 3 signals**



### 22.3.24 Interfacing with Hall sensors

This is done using the advanced-control timers (TIM1 or TIM8) to generate PWM signals to drive the motor and another timer TIMx (TIM2, TIM3, TIM4) referred to as “interfacing timer” in [Figure 198](#). The “interfacing timer” captures the 3 timer input pins (CC1, CC2, CC3) connected through a XOR to the TI1 input channel (selected by setting the TI1S bit in the TIMx\_CR2 register).

The slave mode controller is configured in reset mode; the slave input is TI1F\_ED. Thus, each time one of the 3 inputs toggles, the counter restarts counting from 0. This creates a time base triggered by any change on the Hall inputs.

On the “interfacing timer”, capture/compare channel 1 is configured in capture mode, capture signal is TRC (See [Figure 172: Capture/compare channel \(example: channel 1 input stage\) on page 657](#)). 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.

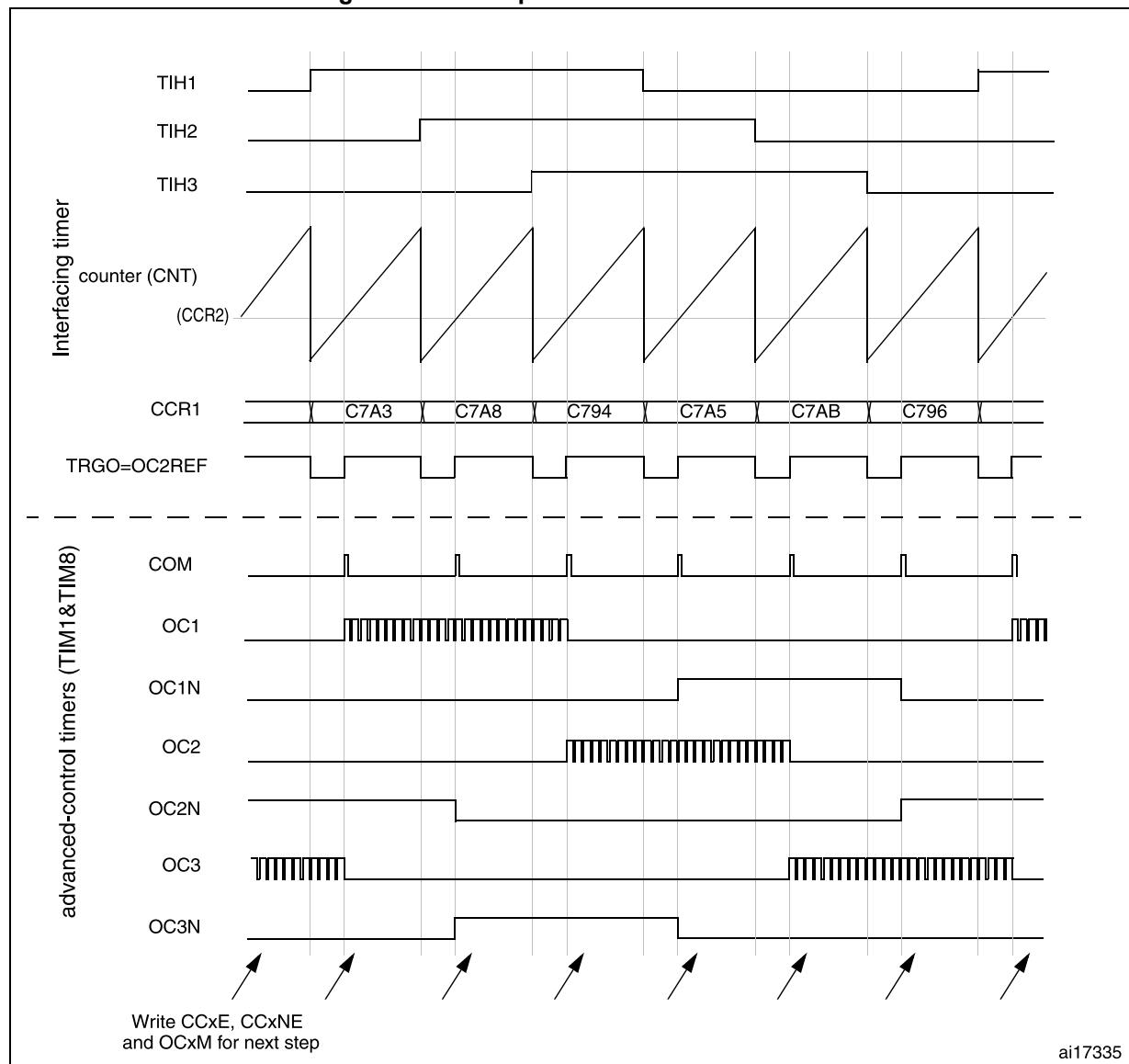
Example: one wants to change the PWM configuration of the advanced-control timer TIM1 after a programmed delay each time a change occurs on the Hall inputs connected to one of the TIMx timers.

- Configure 3 timer inputs ORed to the TI1 input channel by writing the TI1S bit in the TIMx\_CR2 register to '1',
- Program the time base: write the TIMx\_ARR to the max value (the counter must be cleared by the TI1 change. Set the prescaler to get a maximum counter period longer than the time between 2 changes on the sensors,
- Program the channel 1 in capture mode (TRC selected): write the CC1S bits in the TIMx\_CCMR1 register to '11'. The digital filter can also be programmed if needed,
- Program the channel 2 in PWM 2 mode with the desired delay: write the OC2M bits to '111' and the CC2S bits to '00' in the TIMx\_CCMR1 register,
- Select OC2REF as trigger output on TRGO: write the MMS bits in the TIMx\_CR2 register to '101',

In the advanced-control timer TIM1, the right ITR input must be selected as trigger input, the timer is programmed to generate PWM signals, the capture/compare control signals are preloaded (CCPC=1 in the TIMx\_CR2 register) and the COM event is controlled by the trigger input (CCUS=1 in the TIMx\_CR2 register). The PWM control bits (CCxE, OCxM) are written after a COM event for the next step (this can be done in an interrupt subroutine generated by the rising edge of OC2REF).

The [Figure 198](#) describes this example.

Figure 198. Example of Hall sensor interface



### 22.3.25 Timer synchronization

The TIMx timers are linked together internally for timer synchronization or chaining. Refer to [Section 23.3.19: Timer synchronization](#) for details. They can be synchronized in several modes: Reset mode, Gated mode, and Trigger mode.

#### Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

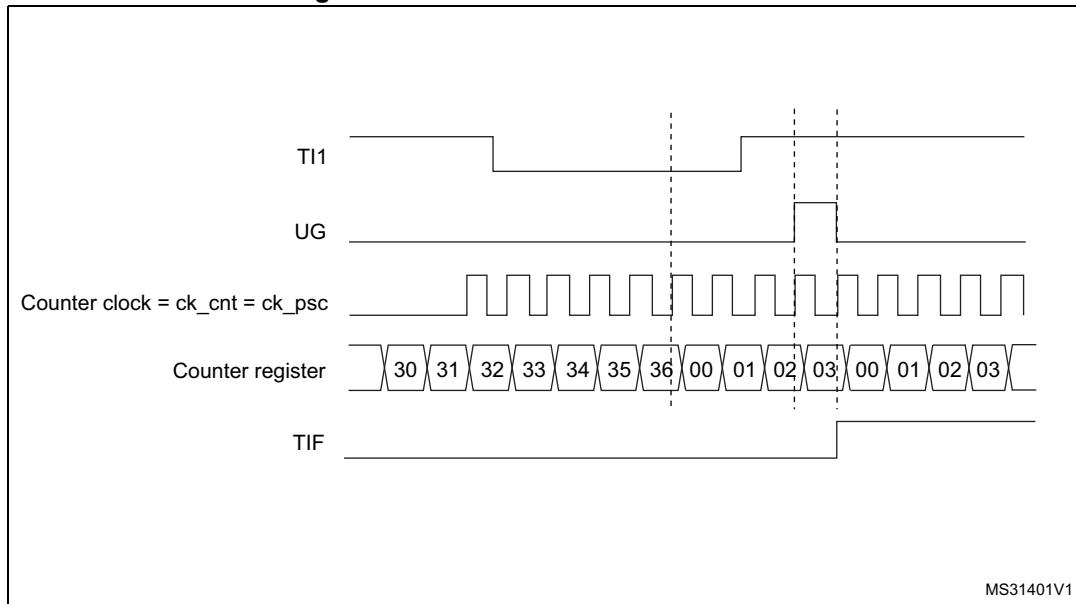
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx\_CCMR1 register. Write CC1P=0 and CC1NP='0' in TIMx\_CCER register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Start the counter by writing CEN=1 in the TIMx\_CR1 register.

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 199. Control circuit in reset mode**



### Slave mode: Gated mode

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

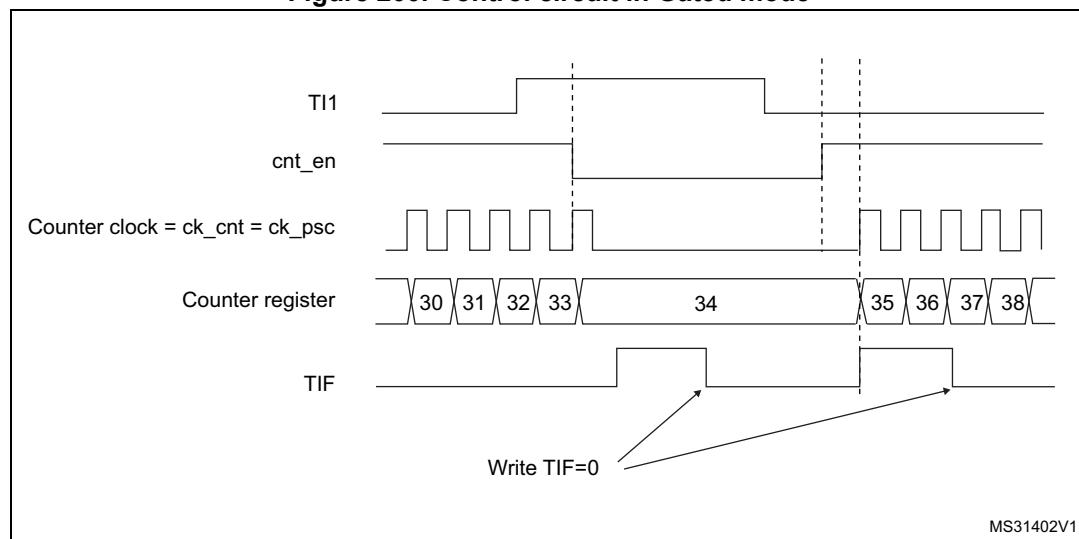
In the following example, the upcounter counts only when TI1 input is low:

- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S=01 in TIMx\_CCMR1 register. Write CC1P=1 and CC1NP='0' in TIMx\_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx\_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

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 200. Control circuit in Gated mode**



### Slave mode: Trigger mode

The counter can start in response to an event on a selected input.

In the following example, the upcounter starts in response to a rising edge on TI2 input:

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we do not need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC2S bits are configured to select the input capture source only, CC2S=01 in TIMx\_CCMR1

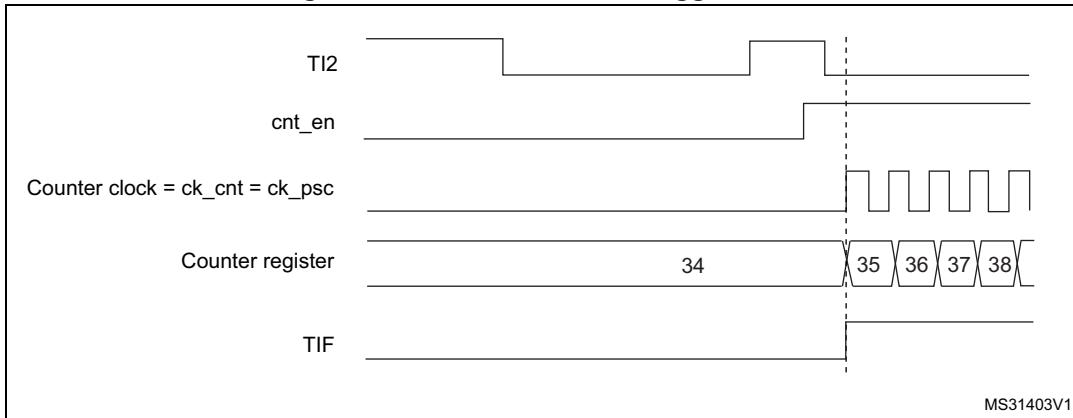
register. Write CC2P=1 and CC2NP=0 in TIMx\_CCER register to validate the polarity (and detect low level only).

- Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx\_SMCR register.

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 201. Control circuit in trigger mode**



### Slave mode: Combined reset + trigger mode

In this case, a rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers, and starts the counter.

This mode is used for one-pulse mode.

### Slave mode: external clock mode 2 + trigger mode

The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input (in reset mode, gated mode or trigger mode). It is recommended not to select ETR as TRGI through the TS bits of TIMx\_SMCR register.

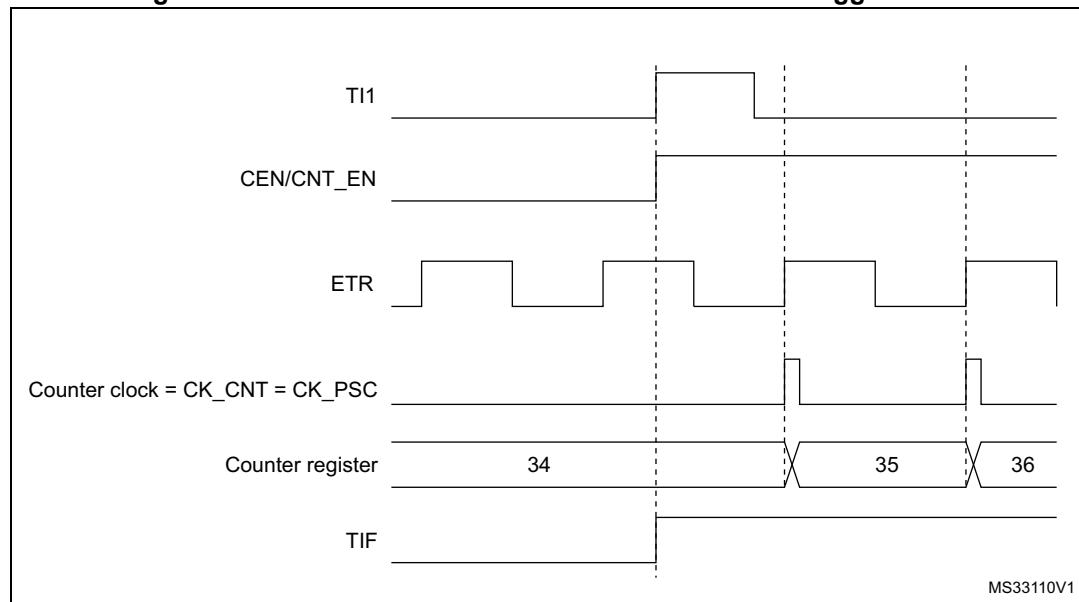
In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

1. Configure the external trigger input circuit by programming the TIMx\_SMCR register as follows:
  - ETF = 0000: no filter
  - ETPS = 00: prescaler disabled
  - ETP = 0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.
2. Configure the channel 1 as follows, to detect rising edges on TI:
  - IC1F = 0000: no filter.
  - The capture prescaler is not used for triggering and does not need to be configured.
  - CC1S = 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 202. Control circuit in external clock mode 2 + trigger mode**



Note:

*The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.*

### 22.3.26 ADC synchronization

The timer can generate an ADC triggering event with various internal signals, such as reset, enable or compare events. It is also possible to generate a pulse issued by internal edge detectors, such as:

- Rising and falling edges of OC4ref
- Rising edge on OC5ref or falling edge on OC6ref

The triggers are issued on the TRGO2 internal line which is redirected to the ADC. There is a total of 16 possible events, which can be selected using the MMS2[3:0] bits in the TIMx\_CR2 register.

An example of an application for 3-phase motor drives is given in [Figure 183 on page 669](#).

**Note:** *The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.*

**Note:** *The clock of the ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the timer.*

### 22.3.27 DMA burst mode

The TIMx timers have the capability to generate multiple DMA requests upon a single event. The main purpose is to be able to re-program part of the timer multiple times without software overhead, but it can also be used to read several registers in a row, at regular intervals.

The DMA controller destination is unique and must point to the virtual register TIMx\_DMAR. On a given timer event, the timer launches a sequence of DMA requests (burst). Each write into the TIMx\_DMAR register is actually redirected to one of the timer registers.

The DBL[4:0] bits in the TIMx\_DCR register set the DMA burst length. The timer recognizes a burst transfer when a read or a write access is done to the TIMx\_DMAR address), i.e. the number of transfers (either in half-words or in bytes).

The DBA[4:0] bits in the TIMx\_DCR registers define the DMA base address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register:

Example:

00000: TIMx\_CR1

00001: TIMx\_CR2

00010: TIMx\_SMCR

As an example, the timer DMA burst feature is used to update the contents of the CCRx registers ( $x = 2, 3, 4$ ) upon an update event, with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
  - DMA channel peripheral address is the DMAR register address
  - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
  - Number of data to transfer = 3 (See note below).
  - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:  
DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let's take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.

*Note:* A null value can be written to the reserved registers.

### 22.3.28 Debug mode

When the microcontroller enters debug mode (Cortex®-M7 core halted), the TIMx counter either continues to work normally or stops, depending on DBG\_TIMx\_STOP configuration bit in DBG module.

For safety purposes, when the counter is stopped, the outputs are disabled (as if the MOE bit was reset). The outputs can either be forced to an inactive state (OSSI bit = 1), or have their control taken over by the GPIO controller (OSSI bit = 0), typically to force a Hi-Z.

For more details, refer to section Debug support (DBG).

## 22.4 TIM1/TIM8 registers

Refer to for a list of abbreviations used in register descriptions.

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

### 22.4.1 TIMx control register 1 (TIMx\_CR1)(x = 1, 8)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 11 **UIFREMAP**: UIF status bit remapping

0: No remapping. UIF status bit is not copied to TIMx\_CNT register bit 31.

1: Remapping enabled. UIF status bit is copied to TIMx\_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and the dead-time and sampling clock ( $t_{DTS}$ ) used by the dead-time generators and the digital filters (ETR, TIx):

00:  $t_{DTS} = t_{CK\_INT}$

01:  $t_{DTS} = 2*t_{CK\_INT}$

10:  $t_{DTS} = 4*t_{CK\_INT}$

11: Reserved, do not program this value

*Note:*  $t_{DTS} = 1/f_{DTS}$ ,  $t_{CK\_INT} = 1/f_{CK\_INT}$ .

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:* Switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1) is not allowed

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.*

## 22.4.2 TIMx control register 2 (TIMx\_CR2)(x = 1, 8)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MMS2[3:0]				Res.	OIS6	Res.	OIS5	
								rw	rw	rw	rw		rw		rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	TI1S	MMS[2:0]				CCDS	CCUS	Res.	CCPC
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	

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

Bits 23:20 **MMS2[3:0]**: Master mode selection 2

These bits allow the information to be sent to ADC for synchronization (TRGO2) to be selected. The combination is as follows:

0000: **Reset** - the UG bit from the TIMx\_EGR register is used as trigger output (TRGO2). If the reset is generated by the trigger input (slave mode controller configured in reset mode), the signal on TRGO2 is delayed compared to the actual reset.

0001: **Enable** - the Counter Enable signal CNT\_EN is used as trigger output (TRGO2). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic AND between the CEN control bit and the trigger input when configured in Gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO2, except if the Master/Slave mode is selected (see the MSM bit description in TIMx\_SMCR register).

0010: **Update** - the update event is selected as trigger output (TRGO2). For instance, a master timer can then be used as a prescaler for a slave timer.

0011: **Compare pulse** - the trigger output sends a positive pulse when the CC1IF flag is to be set (even if it was already high), as soon as a capture or compare match occurs (TRGO2).

0100: **Compare** - OC1REFC signal is used as trigger output (TRGO2)

0101: **Compare** - OC2REFC signal is used as trigger output (TRGO2)

0110: **Compare** - OC3REFC signal is used as trigger output (TRGO2)

0111: **Compare** - OC4REFC signal is used as trigger output (TRGO2)

1000: **Compare** - OC5REFC signal is used as trigger output (TRGO2)

1001: **Compare** - OC6REFC signal is used as trigger output (TRGO2)

1010: **Compare Pulse** - OC4REFC rising or falling edges generate pulses on TRGO2

1011: **Compare Pulse** - OC6REFC rising or falling edges generate pulses on TRGO2

1100: **Compare Pulse** - OC4REFC or OC6REFC rising edges generate pulses on TRGO2

1101: **Compare Pulse** - OC4REFC rising or OC6REFC falling edges generate pulses on TRGO2

1110: **Compare Pulse** - OC5REFC or OC6REFC rising edges generate pulses on TRGO2

1111: **Compare Pulse** - OC5REFC rising or OC6REFC falling edges generate pulses on TRGO2

*Note: The clock of the slave timer or ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

Bit 19 Reserved, must be kept at reset value.

Bit 18 **OIS6**: Output Idle state 6 (OC6 output)

Refer to OIS1 bit

Bit 17 Reserved, must be kept at reset value.

Bit 16 **OIS5**: Output Idle state 5 (OC5 output)

Refer to OIS1 bit

Bit 15 Reserved, must be kept at reset value.

Bit 14 **OIS4**: Output Idle state 4 (OC4 output)

Refer to OIS1 bit

Bit 13 **OIS3N**: Output Idle state 3 (OC3N output)

Refer to OIS1N bit

Bit 12 **OIS3**: Output Idle state 3 (OC3 output)

Refer to OIS1 bit

Bit 11 **OIS2N**: Output Idle state 2 (OC2N output)

Refer to OIS1N bit

Bit 10 **OIS2**: Output Idle state 2 (OC2 output)

Refer to OIS1 bit

Bit 9 **OIS1N**: Output Idle state 1 (OC1N output)

0: OC1N=0 after a dead-time when MOE=0

1: OC1N=1 after a dead-time when MOE=0

*Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 8 **OIS1**: Output Idle state 1 (OC1 output)

0: OC1=0 (after a dead-time if OC1N is implemented) when MOE=0

1: OC1=1 (after a dead-time if OC1N is implemented) when MOE=0

*Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 7 **TI1S**: TI1 selection

0: The TIMx\_CH1 pin is connected to TI1 input

1: The TIMx\_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

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

These bits allow selected 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 AND 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** - OC1REFC signal is used as trigger output (TRGO)

101: **Compare** - OC2REFC signal is used as trigger output (TRGO)

110: **Compare** - OC3REFC signal is used as trigger output (TRGO)

111: **Compare** - OC4REFC signal is used as trigger output (TRGO)

*Note: The clock of the slave timer or ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

Bit 3 **CCDS**: Capture/compare DMA selection

0: CCx DMA request sent when CCx event occurs

1: CCx DMA requests sent when update event occurs

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

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

1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI

*Note: This bit acts only on channels that have a complementary output.*

Bit 1 Reserved, must be kept at reset value.

Bit 0 **CCPC**: Capture/compare preloaded control

0: CCxE, CCxNE and OCxM bits are not preloaded

1: CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs (COMG bit set or rising edge detected on TRGI, depending on the CCUS bit).

*Note: This bit acts only on channels that have a complementary output.*

### 22.4.3 TIMx slave mode control register (TIMx\_SMCR)(x = 1, 8)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SMS[3]
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			Res.	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw

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

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: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).*

*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).*

*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  $f_{CK\_INT}$  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[2:0]**: Trigger selection

This bit-field selects the trigger input to be used to synchronize the counter.

- 000: Internal Trigger 0 (ITR0)
- 001: Internal Trigger 1 (ITR1)
- 010: Internal Trigger 2 (ITR2)
- 011: Internal Trigger 3 (ITR3)
- 100: T11 Edge Detector (T11F\_ED)
- 101: Filtered Timer Input 1 (TI1FP1)
- 110: Filtered Timer Input 2 (TI2FP2)
- 111: External Trigger input (ETRF)

See [Table 149: TIMx internal trigger connection on page 700](#) for more details on ITRx meaning for each Timer.

*Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

*Note: The other bit is at position 16 in the same register*

Bit 3 Reserved, must be kept at reset value.

Bits 16, 2, 1, 0 **SMS[3:0]**: Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (refer to ETP bit in TIMx\_SMCR for tim\_etr\_in and CCxP/CCxNP bits in TIMx\_CCER register for tim\_ti1fp1 and tim\_ti2fp2).

0000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

0001: Encoder mode 1 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

0010: Encoder mode 2 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

0011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

0100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

0101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

0110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

0111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

1000: Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter.

Codes above 1000: Reserved.

*Note: The gated mode must not be used if TI1F\_ED is selected as the trigger input (TS=100). Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.*

*Note: The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.*

**Table 149. 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

#### 22.4.4 TIMx DMA/interrupt enable register (TIMx\_DIER)(x = 1, 8)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE

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

### 22.4.5 TIMx status register (TIMx\_SR)(x = 1, 8)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CC6IF	CC5IF
														rc_w0	rc_w0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	SBIF	CC4OF	CC3OF	CC2OF	CC1OF	B2IF	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
		rc_w0													

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **CC6IF**: Compare 6 interrupt flag

Refer to CC1IF description (Note: Channel 6 can only be configured as output)

Bit 16 **CC5IF**: Compare 5 interrupt flag

Refer to CC1IF description (Note: Channel 5 can only be configured as output)

Bits 15:14 Reserved, must be kept at reset value.

Bit 13 **SBIF**: System Break interrupt flag

This flag is set by hardware as soon as the system break input goes active. It can be cleared by software if the system break input is not active.

This flag must be reset to re-start PWM operation.

0: No break event occurred.

1: An active level has been detected on the system break input. An interrupt is generated if BIE=1 in the TIMx\_DIER register.

Bit 12 **CC4OF**: Capture/Compare 4 overcapture flag

Refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 overcapture flag

Refer to CC1OF description

Bit 10 **CC2OF**: Capture/Compare 2 overcapture flag

Refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bit 8 **B2IF**: Break 2 interrupt flag

This flag is set by hardware as soon as the break 2 input goes active. It can be cleared by software if the break 2 input is not active.

0: No break event occurred.

1: An active level has been detected on the break 2 input. An interrupt is generated if BIE=1 in the TIMx\_DIER register.

**Bit 7 **BIF**: Break interrupt flag**

This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active.

0: No break event occurred.

1: An active level has been detected on the break input. An interrupt is generated if BIE=1 in the TIMx\_DIER register.

**Bit 6 **TIF**: Trigger interrupt flag**

This flag is set by hardware on the TRG 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**

This flag is set by hardware. It is cleared by software (input capture or output compare mode) or by reading the TIMx\_CCR1 register (input capture mode only).

0: No compare match / No input capture occurred

1: A compare match or an input capture occurred.

**If channel CC1 is configured as output:** this flag is set when the content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register. When the content of TIMx\_CCR1 is greater than the content of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in up-counting and up/down-counting modes) or underflow (in down-counting mode). There are 3 possible options for flag setting in center-aligned mode, refer to the CMS bits in the TIMx\_CR1 register for the full description.

**If channel CC1 is configured as input:** this bit is set when counter value has been captured in TIMx\_CCR1 register (an edge has been detected on IC1, as per the edge sensitivity defined with the CC1P and CC1NP bits setting, in TIMx\_CCER).

**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 22.4.3: TIMx slave mode control register \(TIMx\\_SMCR\)\(x = 1, 8\)](#)), if URS=0 and UDIS=0 in the TIMx\_CR1 register.

## 22.4.6 TIMx event generation register (TIMx\_EGR)(x = 1, 8)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	B2G	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG						
							w	w	w	w	w	w	w	w	w

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **B2G:** Break 2 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A break 2 event is generated. MOE bit is cleared and B2IF flag is set. Related interrupt can occur if enabled.

Bit 7 **BG:** Break generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.

Bit 6 **TG:** Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIMx\_SR register. Related interrupt or DMA transfer can occur if enabled.

Bit 5 **COMG:** Capture/Compare control update generation

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

0: No action

1: When CCPC bit is set, it allows CCxE, CCxNE and OCxM bits to be updated.

*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. The prescaler internal counter is also cleared (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).

## 22.4.7 TIMx capture/compare mode register 1 (TIMx\_CCMR1)(x = 1, 8)

Address offset: 0x18

Reset value: 0x0000 0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function for input capture and for output compare modes. It is possible to combine both modes independently (e.g. channel 1 in input capture mode and channel 2 in output compare mode).

**Input capture 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															
IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:12 **IC2F[3:0]**: Input capture 2 filter

Refer to IC1F[3:0] description.

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Refer to IC1PSC[1:0] description.

**Bits 9:8 CC2S[1:0]: Capture/Compare 2 selection**

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx\_CCER).*

**Bits 7:4 IC1F[3:0]: Input capture 1 filter**

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

0000: No filter, sampling is done at  $f_{DTS}$

0001:  $f_{SAMPLING} = f_{CK\_INT}$ , N=2

0010:  $f_{SAMPLING} = f_{CK\_INT}$ , N=4

0011:  $f_{SAMPLING} = f_{CK\_INT}$ , N=8

0100:  $f_{SAMPLING} = f_{DTS}/2$ , N=6

0101:  $f_{SAMPLING} = f_{DTS}/2$ , N=8

0110:  $f_{SAMPLING} = f_{DTS}/4$ , N=6

0111:  $f_{SAMPLING} = f_{DTS}/4$ , N=8

1000:  $f_{SAMPLING} = f_{DTS}/8$ , N=6

1001:  $f_{SAMPLING} = f_{DTS}/8$ , N=8

1010:  $f_{SAMPLING} = f_{DTS}/16$ , N=5

1011:  $f_{SAMPLING} = f_{DTS}/16$ , N=6

1100:  $f_{SAMPLING} = f_{DTS}/16$ , N=8

1101:  $f_{SAMPLING} = f_{DTS}/32$ , N=5

1110:  $f_{SAMPLING} = f_{DTS}/32$ , N=6

1111:  $f_{SAMPLING} = f_{DTS}/32$ , N=8

**Bits 3:2 IC1PSC[1:0]: Input capture 1 prescaler**

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1). The prescaler is reset as soon as CC1E='0' (TIMx\_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input

01: capture is done once every 2 events

10: capture is done once every 4 events

11: capture is done once every 8 events

**Bits 1:0 CC1S[1:0]: Capture/Compare 1 Selection**

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx\_CCER).*

## 22.4.8 TIMx capture/compare mode register 1 [alternate] (TIMx\_CCMR1)(x = 1, 8)

Address offset: 0x18

Reset value: 0x0000 0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the

corresponding CCxS bits. All the other bits of this register have a different function for input capture and for output compare modes. It is possible to combine both modes independently (e.g. channel 1 in input capture mode and channel 2 in output compare mode).

#### Output compare mode:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M[3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M[3]
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2 CE	OC2M[2:0]			OC2 PE	OC2 FE	CC2S[1:0]		OC1 CE	OC1M[2:0]			OC1 PE	OC1 FE	CC1S[1:0]	
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 23:17 Reserved, must be kept at reset value.

Bit 15 **OC2CE**: Output Compare 2 clear enable

Refer to OC1CE description.

Bits 24, 14:12 **OC2M[3:0]**: Output Compare 2 mode

Refer to OC1M[3:0] description.

Bit 11 **OC2PE**: Output Compare 2 preload enable

Refer to OC1PE description.

Bit 10 **OC2FE**: Output Compare 2 fast enable

Refer to OC1FE description.

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx\_CCER).

Bit 7 **OC1CE**: Output Compare 1 clear enable

0: OC1Ref is not affected by the ETRF input

1: OC1Ref is cleared as soon as a High level is detected on ETRF input

Bits 16, 6:4 **OC1M[3:0]**: Output Compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

0000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs. This mode can be used when the timer serves as a software timebase. When the frozen mode is enabled during timer operation, the output keeps the state (active or inactive) it had before entering the frozen state.

0001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

0010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

0011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.

0100: Force inactive level - OC1REF is forced low.

0101: Force active level - OC1REF is forced high.

0110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF='1').

0111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

1000: Retriggerable OPM mode 1 - In up-counting mode, the channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes active again at the next update. In down-counting mode, the channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes inactive again at the next update.

1001: Retriggerable OPM mode 2 - In up-counting mode, the channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 2 and the channels becomes inactive again at the next update. In down-counting mode, the channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes active again at the next update.

1010: Reserved,

1011: Reserved,

1100: Combined PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC is the logical OR between OC1REF and OC2REF.

1101: Combined PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC is the logical AND between OC1REF and OC2REF.

1110: Asymmetric PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC outputs OC1REF when the counter is counting up, OC2REF when it is counting down.

1111: Asymmetric PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC outputs OC1REF when the counter is counting up, OC2REF when it is counting down.

*Note:* These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).

*Note:* In PWM mode, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode.

*Note:* On channels having a complementary output, this bit field is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the OC1M active bits take the new value from the preloaded bits only when a COM event is generated.

*Note:* The OC1M[3] bit is not contiguous, located in bit 16.

Bit 3 **OC1PE**: Output Compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).*

Bit 2 **OC1FE**: Output Compare 1 fast enable

This bit decreases the latency between a trigger event and a transition on the timer output. It must be used in one-pulse mode (OPM bit set in TIMx\_CR1 register), to have the output pulse starting as soon as possible after the starting trigger.

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[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx\_CCER).*

## 22.4.9 TIMx capture/compare mode register 2 (TIMx\_CCMR2)(x = 1, 8)

Address offset: 0x1C

Reset value: 0x0000 0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function for input capture and for output compare modes. It is possible to combine both modes independently (e.g. channel 1 in input capture mode and channel 2 in output compare mode).

### Input capture 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
IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]		IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
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 15:12 **IC4F[3:0]**: Input capture 4 filter

Refer to IC1F[3:0] description.

Bits 11:10 **IC4PSC[1:0]**: Input capture 4 prescaler  
 Refer to IC1PSC[1:0] description.

Bits 9:8 **CC4S[1:0]**: 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[3:0]**: Input capture 3 filter

Refer to IC1F[3:0] description.

Bits 3:2 **IC3PSC[1:0]**: Input capture 3 prescaler

Refer to IC1PSC[1:0] description.

Bits 1:0 **CC3S[1:0]**: 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).*

## 22.4.10 TIMx capture/compare mode register 2 [alternate] (TIMx\_CCMR2)(x = 1, 8)

Address offset: 0x1C

Reset value: 0x0000 0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function for input capture and for output compare modes. It is possible to combine both modes independently (e.g. channel 1 in input capture mode and channel 2 in output compare mode).

### Output compare mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC4M[3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC3M[3]
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4 CE	OC4M[2:0]			OC4 PE	OC4 FE	CC4S[1:0]		OC3 CE	OC3M[2:0]			OC3 PE	OC3 FE	CC3S[1:0]	
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 23:17 Reserved, must be kept at reset value.

Bit 15 **OC4CE**: Output compare 4 clear enable

Refer to OC1CE description.

Bits 24, 14:12 **OC4M[3:0]**: Output compare 4 mode  
 Refer to OC3M[3:0] description.

Bit 11 **OC4PE**: Output compare 4 preload enable  
 Refer to OC1PE description.

Bit 10 **OC4FE**: Output compare 4 fast enable  
 Refer to OC1FE description.

Bits 9:8 **CC4S[1:0]**: 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  
 Refer to OC1CE description.

Bits 16, 6:4 **OC3M[3:0]**: Output compare 3 mode  
 Refer to OC1M[3:0] description.

Bit 3 **OC3PE**: Output compare 3 preload enable  
 Refer to OC1PE description.

Bit 2 **OC3FE**: Output compare 3 fast enable  
 Refer to OC1FE description.

Bits 1:0 **CC3S[1:0]**: 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).*

## 22.4.11 TIMx capture/compare enable register (TIMx\_CCER)(x = 1, 8)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CC6P	CC6E	Res.	Res.	CC5P	CC5E
										rw	rw			rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

- Bit 21 **CC6P**: Capture/Compare 6 output polarity  
Refer to CC1P description
- Bit 20 **CC6E**: Capture/Compare 6 output enable  
Refer to CC1E description
- Bits 19:18 Reserved, must be kept at reset value.
- Bit 17 **CC5P**: Capture/Compare 5 output polarity  
Refer to CC1P description
- Bit 16 **CC5E**: Capture/Compare 5 output enable  
Refer to CC1E description
- Bit 15 **CC4NP**: Capture/Compare 4 complementary output polarity  
Refer to CC1NP description
- Bit 14 Reserved, must be kept at reset value.
- Bit 13 **CC4P**: Capture/Compare 4 output polarity  
Refer to CC1P description
- Bit 12 **CC4E**: Capture/Compare 4 output enable  
Refer to CC1E description
- Bit 11 **CC3NP**: Capture/Compare 3 complementary output polarity  
Refer to CC1NP description
- Bit 10 **CC3NE**: Capture/Compare 3 complementary output enable  
Refer to CC1NE description
- Bit 9 **CC3P**: Capture/Compare 3 output polarity  
Refer to CC1P description
- Bit 8 **CC3E**: Capture/Compare 3 output enable  
Refer to CC1E description
- Bit 7 **CC2NP**: Capture/Compare 2 complementary output polarity  
Refer to CC1NP description
- Bit 6 **CC2NE**: Capture/Compare 2 complementary output enable  
Refer to CC1NE description
- Bit 5 **CC2P**: Capture/Compare 2 output polarity  
Refer to CC1P description
- Bit 4 **CC2E**: Capture/Compare 2 output enable  
Refer to CC1E description

Bit 3 **CC1NP**: Capture/Compare 1 complementary output polarity

**CC1 channel configured as output:**

0: OC1N active high.

1: OC1N active low.

**CC1 channel configured as input:**

This bit is used in conjunction with CC1P to define the polarity of TI1FP1 and TI2FP1. Refer to CC1P description.

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S="00" (channel configured as output).*

*On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1NP active bit takes the new value from the preloaded bit only when a Commutation event is generated.*

Bit 2 **CC1NE**: Capture/Compare 1 complementary output enable

0: Off - OC1N is not active. OC1N level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

1: On - OC1N signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

*On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1NE active bit takes the new value from the preloaded bit only when a Commutation event is generated.*

Bit 1 **CC1P**: Capture/Compare 1 output polarity

0: OC1 active high (output mode) / Edge sensitivity selection (input mode, see below)

1: OC1 active low (output mode) / Edge sensitivity selection (input mode, see below)

When CC1 channel is configured as input, both CC1NP/CC1P bits select the active polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

CC1NP=0, CC1P=0: 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).

CC1NP=0, CC1P=1: 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).

CC1NP=1, CC1P=1: 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.

CC1NP=1, CC1P=0: The configuration is reserved, it must not be used.

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

*On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1P active bit takes the new value from the preloaded bit only when a Commutation event is generated.*

Bit 0 **CC1E**: Capture/Compare 1 output enable

0: Capture mode disabled / OC1 is not active (see below)

1: Capture mode enabled / OC1 signal is output on the corresponding output pin

**When CC1 channel is configured as output**, the OC1 level depends on MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits, regardless of the CC1E bits state. Refer to [Table 150](#) for details.

*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 150. 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	X	0	0	Output disabled (not driven by the timer: Hi-Z) OCx=0, OCxN=0	
		0	0	1	Output disabled (not driven by the timer: Hi-Z) OCx=0	OCxREF + Polarity OCxN = OCxREF xor CCxNP
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Output Disabled (not driven by the timer: Hi-Z) OCxN=0
		X	1	1	OCREF + Polarity + dead-time	Complementary to OCREF (not OCREF) + Polarity + dead-time
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP	OCxREF + Polarity OCxN = OCxREF x or CCxNP
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Off-State (output enabled with inactive state) OCxN=CCxNP
0	X	0	X	X	Output disabled (not driven by the timer: Hi-Z).	
		0	0			
		0	1		Off-State (output enabled with inactive state)	
		1	0		Asynchronously: OCx=CCxP, OCxN=CCxNP (if BRK or BRK2 is triggered).	
		1	1		Then (this is valid only if BRK is triggered), if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCx and OCxN both in active state (may cause a short circuit when driving switches in half-bridge configuration). <b>Note:</b> BRK2 can only be used if OSSI = OSSR = 1.	

- When both outputs of a channel are not used (control taken over by GPIO), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

**Note:** The state of the external I/O pins connected to the complementary OCx and OCxN channels depends on the OCx and OCxN channel state and the GPIO registers.

### 22.4.12 TIMx counter (TIMx\_CNT)(x = 1, 8)

Address offset: 0x24

Reset value: 0x0000 0000

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

Bit 31 **UIFCPY**: UIF copy

This bit is a read-only copy of the UIF bit of the TIMx\_ISR register. If the UIFREMAP bit in the TIMxCR1 is reset, bit 31 is reserved and read at 0.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

### 22.4.13 TIMx prescaler (TIMx\_PSC)(x = 1, 8)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency (CK\_CNT) is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

### 22.4.14 TIMx auto-reload register (TIMx\_ARR)(x = 1, 8)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 22.3.1: Time-base unit on page 637](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 22.4.15 TIMx repetition counter register (TIMx\_RCR)(x = 1, 8)

Address offset: 0x30

Reset value: 0x0000

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

Bits 15:0 **REP[15:0]**: Repetition counter value

These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.

Each time the REP\_CNT related downcounter reaches zero, an update event is generated and it restarts counting from REP value. As REP\_CNT is reloaded with REP value only at the repetition update event U\_RC, any write to the TIMx\_RCR register is not taken in account until the next repetition update event.

It means in PWM mode (REP+1) corresponds to:  
the number of PWM periods in edge-aligned mode  
the number of half PWM period in center-aligned mode.

### 22.4.16 TIMx capture/compare register 1 (TIMx\_CCR1)(x = 1, 8)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

If channel CC1 is configured as output: CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

If channel CC1 is configured as input: CR1 is the counter value transferred by the last input capture 1 event (IC1). The TIMx\_CCR1 register is read-only and cannot be programmed.

### 22.4.17 TIMx capture/compare register 2 (TIMx\_CCR2)(x = 1, 8)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]**: Capture/Compare 2 value

**If channel CC2 is configured as output:** CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC2 output.

**If channel CC2 is configured as input:** CCR2 is the counter value transferred by the last input capture 2 event (IC2). The TIMx\_CCR2 register is read-only and cannot be programmed.

### 22.4.18 TIMx capture/compare register 3 (TIMx\_CCR3)(x = 1, 8)

Address offset: 0x3C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR3[15:0]**: Capture/Compare value

**If channel CC3 is configured as output:** CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC3 output.

**If channel CC3 is configured as input:** CCR3 is the counter value transferred by the last input capture 3 event (IC3). The TIMx\_CCR3 register is read-only and cannot be programmed.

### 22.4.19 TIMx capture/compare register 4 (TIMx\_CCR4)(x = 1, 8)

Address offset: 0x40

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR4[15:0]**: Capture/Compare value

**If channel CC4 is configured as output:** CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (bit OC4PE). Else the preload value is copied in the active capture/compare 4 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC4 output.

**If channel CC4 is configured as input:** CCR4 is the counter value transferred by the last input capture 4 event (IC4). The TIMx\_CCR4 register is read-only and cannot be programmed.

### 22.4.20 TIMx break and dead-time register (TIMx\_BDTR)(x = 1, 8)

Address offset: 0x44

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	BK2P	BK2E	BK2F[3:0]							
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		DTG[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw						

**Note:** As the bits BK2P, BK2E, BK2F[3:0], BKF[3:0], AOE, BKP, BKE, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx\_BDTR register.

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

Bit 25 **BK2P:** Break 2 polarity

- 0: Break input BRK2 is active low
- 1: Break input BRK2 is active high

**Note:** This bit cannot be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

**Note:** Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 24 **BK2E**: Break 2 enable

This bit enables the complete break 2 protection (including all sources connected to bk\_acth and BKIN sources, as per [Figure 187: Break and Break2 circuitry overview](#)).

- 0: Break2 function disabled
- 1: Break2 function enabled

*Note: The BKIN2 must only be used with OSSR = OSSI = 1.*

*Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).*

*Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.*

Bits 23:20 **BK2F[3:0]**: Break 2 filter

This bit-field defines the frequency used to sample BRK2 input and the length of the digital filter applied to BRK2. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, BRK2 acts asynchronously
- 0001:  $f_{\text{SAMPLING}}=f_{\text{CK\_INT}}$ , N=2
- 0010:  $f_{\text{SAMPLING}}=f_{\text{CK\_INT}}$ , N=4
- 0011:  $f_{\text{SAMPLING}}=f_{\text{CK\_INT}}$ , N=8
- 0100:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/2$ , N=6
- 0101:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/2$ , N=8
- 0110:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/4$ , N=6
- 0111:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/4$ , N=8
- 1000:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/8$ , N=6
- 1001:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/8$ , N=8
- 1010:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$ , N=5
- 1011:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$ , N=6
- 1100:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$ , N=8
- 1101:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$ , N=5
- 1110:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$ , N=6
- 1111:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$ , N=8

*Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bits 19:16 **BKF[3:0]**: Break filter

This bit-field defines the frequency used to sample BRK input and the length of the digital filter applied to BRK. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, BRK acts asynchronously
- 0001:  $f_{\text{SAMPLING}}=f_{\text{CK\_INT}}$ , N=2
- 0010:  $f_{\text{SAMPLING}}=f_{\text{CK\_INT}}$ , N=4
- 0011:  $f_{\text{SAMPLING}}=f_{\text{CK\_INT}}$ , N=8
- 0100:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/2$ , N=6
- 0101:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/2$ , N=8
- 0110:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/4$ , N=6
- 0111:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/4$ , N=8
- 1000:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/8$ , N=6
- 1001:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/8$ , N=8
- 1010:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$ , N=5
- 1011:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$ , N=6
- 1100:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$ , N=8
- 1101:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$ , N=5
- 1110:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$ , N=6
- 1111:  $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$ , N=8

*Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 15 **MOE**: Main output enable

This bit is cleared asynchronously by hardware as soon as one of the break inputs is active (BRK or BRK2). It is set by software or automatically depending on the AOE bit. It is acting only on the channels which are configured in output.

0: In response to a break 2 event. OC and OCN outputs are disabled

In response to a break event or if MOE is written to 0: OC and OCN outputs are disabled or forced to idle state depending on the OSS1 bit.

1: OC and OCN outputs are enabled if their respective enable bits are set (CCxE, CCxNE in TIMx\_CCER register).

See OC/OCN enable description for more details ([Section 22.4.11: TIMx capture/compare enable register \(TIMx\\_CCER\)\(x = 1, 8\)](#)).

Bit 14 **AOE**: Automatic output enable

0: MOE can be set only by software

1: MOE can be set by software or automatically at the next update event (if none of the break inputs BRK and BRK2 is active)

*Note:* This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

Bit 13 **BKP**: Break polarity

0: Break input BRK is active low

1: Break input BRK is active high

*Note:* This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

*Note:* Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 12 **BKE**: Break enable

This bit enables the complete break protection (including all sources connected to bk\_acth and BRK sources, as per [Figure 187: Break and Break2 circuitry overview](#)).

0: Break function disabled

1: Break function enabled

*Note:* This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

*Note:* Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 11 **OSSR**: Off-state selection for Run mode

This bit is used when MOE=1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer.

See OC/OCN enable description for more details ([Section 22.4.11: TIMx capture/compare enable register \(TIMx\\_CCER\)\(x = 1, 8\)](#)).

0: When inactive, OC/OCN outputs are disabled (the timer releases the output control which is taken over by the GPIO logic, which forces a Hi-Z state).

1: When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1 (the output is still controlled by the timer).

*Note:* This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).

Bit 10 **OSSI**: Off-state selection for Idle mode

This bit is used when MOE=0 due to a break event or by a software write, on channels configured as outputs.

See OC/OCN enable description for more details ([Section 22.4.11: TIMx capture/compare enable register \(TIMx\\_CCER\)\(x = 1, 8\)](#)).

0: When inactive, OC/OCN outputs are disabled (the timer releases the output control which is taken over by the GPIO logic and which imposes a Hi-Z state).

1: When inactive, OC/OCN outputs are first forced with their inactive level then forced to their idle level after the deadtime. The timer maintains its control over the output.

*Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bits 9:8 **LOCK[1:0]**: Lock configuration

These bits offer a write protection against software errors.

00: LOCK OFF - No bit is write protected.

01: LOCK Level 1 = DTG bits in TIMx\_BDTR register, OISx and OISxN bits in TIMx\_CR2 register and BKE/BKP/AOE bits in TIMx\_BDTR register can no longer be written.

10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP/CCxNP bits in TIMx\_CCER register, as long as the related channel is configured in output through the CCxS bits) as well as OSSR and OSSI bits can no longer be written.

11: LOCK Level 3 = LOCK Level 2 + CC Control bits (OCxM and OCxPE bits in TIMx\_CCMRx registers, as long as the related channel is configured in output through the CCxS bits) can no longer be written.

*Note: The LOCK bits can be written only once after the reset. Once the TIMx\_BDTR register has been written, their content is frozen until the next reset.*

Bits 7:0 **DTG[7:0]**: Dead-time generator setup

This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.

DTG[7:5] = 0xx => DT = DTG[7:0] x t<sub>DTG</sub> with t<sub>DTG</sub> = t<sub>DTS</sub>.

DTG[7:5] = 10x => DT = (64 + DTG[5:0]) x t<sub>DTG</sub> with t<sub>DTG</sub> = 2 x t<sub>DTS</sub>.

DTG[7:5] = 110 => DT = (32 + DTG[4:0]) x t<sub>DTG</sub> with t<sub>DTG</sub> = 8 x t<sub>DTS</sub>.

DTG[7:5] = 111 => DT = (32 + DTG[4:0]) x t<sub>DTG</sub> with t<sub>DTG</sub> = 16 x t<sub>DTS</sub>.

Example if t<sub>DTS</sub> = 125 ns (8 MHz), dead-time possible values are:

0 to 15875 ns by 125 ns steps,

16 µs to 31750 ns by 250 ns steps,

32 µs to 63 µs by 1 µs steps,

64 µs to 126 µs by 2 µs steps

*Note: This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

## 22.4.21 TIMx DMA control register (TIMx\_DCR)(x = 1, 8)

Address offset: 0x48

Reset value: 0x0000

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

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:8 **DBL[4:0]**: DMA burst length

This 5-bit vector defines the length of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx\_DMAR address), i.e. the number of transfers. Transfers can be in half-words or in bytes (see example below).

00000: 1 transfer  
00001: 2 transfers  
00010: 3 transfers

...  
10001: 18 transfers

**Example:** Let us consider the following transfer: DBL = 7 bytes & DBA = TIMx\_CR1.

- If DBL = 7 bytes and DBA = TIMx\_CR1 represents the address of the byte to be transferred, the address of the transfer should be given by the following equation:

(TIMx\_CR1 address) + DBA + (DMA index), where DMA index = DBL

In this example, 7 bytes are added to (TIMx\_CR1 address) + DBA, which gives us the address from/to which the data is copied. In this case, the transfer is done to 7 registers starting from the following address: (TIMx\_CR1 address) + DBA

According to the configuration of the DMA Data Size, several cases may occur:

- If the DMA Data Size is configured in half-words, 16-bit data is transferred to each of the 7 registers.
- If the DMA Data Size is configured in bytes, the data is also transferred to 7 registers: the first register contains the first MSB byte, the second register, the first LSB byte and so on. So with the transfer Timer, one also has to specify the size of data transferred by DMA.

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

Bits 4:0 **DBA[4:0]**: DMA base address

This 5-bits vector defines the base-address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register.

**Example:**

00000: TIMx\_CR1,  
00001: TIMx\_CR2,  
00010: TIMx\_SMCR,

...

## 22.4.22 TIMx DMA address for full transfer (TIMx\_DMAR)(x = 1, 8)

Address offset: 0x4C

Reset value: 0x0000 0000

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

Bits 31:0 **DMAB[31:0]**: DMA register for burst accesses

A read or write operation to the DMAR register accesses the register located at the address (TIMx\_CR1 address) + (DBA + DMA index) x 4

where TIMx\_CR1 address is the address of the control register 1, DBA is the DMA base address configured in TIMx\_DCR register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx\_DCR).

### 22.4.23 TIMx capture/compare mode register 3 (TIMx\_CCMR3)(x = 1, 8)

Address offset: 0x54

Reset value: 0x0000 0000

The channels 5 and 6 can only be configured in output.

#### Output compare mode:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC6M[3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC5M[3]
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC6CE	OC6M[2:0]			OC6PE	OC6FE	Res.	Res.	OC5CE	OC5M[2:0]			OC5PE	OC5FE	Res.	Res.
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw		

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

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

Bit 15 **OC6CE**: Output compare 6 clear enable

Refer to OC1CE description.

Bits 24, 14, 13, 12 **OC6M[3:0]**: Output compare 6 mode

Refer to OC1M description.

Bit 11 **OC6PE**: Output compare 6 preload enable

Refer to OC1PE description.

Bit 10 **OC6FE**: Output compare 6 fast enable

Refer to OC1FE description.

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

Bit 7 **OC5CE**: Output compare 5 clear enable

Refer to OC1CE description.

Bits 16, 6, 5, 4 **OC5M[3:0]**: Output compare 5 mode

Refer to OC1M description.

Bit 3 **OC5PE**: Output compare 5 preload enable

Refer to OC1PE description.

Bit 2 **OC5FE**: Output compare 5 fast enable

Refer to OC1FE description.

Bits 1:0 Reserved, must be kept at reset value.

## 22.4.24 TIMx capture/compare register 5 (TIMx\_CCR5)(x = 1, 8)

Address offset: 0x58

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GC5C3	GC5C2	GC5C1	Res.												
rw	rw	rw													
15      14      13      12      11      10      9      8      7      6      5      4      3      2      1      0															
CCR5[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **GC5C3**: Group Channel 5 and Channel 3

Distortion on Channel 3 output:

0: No effect of OC5REF on OC3REFC

1: OC3REFC is the logical AND of OC3REFC and OC5REF

This bit can either have immediate effect or be preloaded and taken into account after an update event (if preload feature is selected in TIMxCCMR2).

*Note: it is also possible to apply this distortion on combined PWM signals.*

Bit 30 **GC5C2**: Group Channel 5 and Channel 2

Distortion on Channel 2 output:

0: No effect of OC5REF on OC2REFC

1: OC2REFC is the logical AND of OC2REFC and OC5REF

This bit can either have immediate effect or be preloaded and taken into account after an update event (if preload feature is selected in TIMxCCMR1).

*Note: it is also possible to apply this distortion on combined PWM signals.*

Bit 29 **GC5C1**: Group Channel 5 and Channel 1

Distortion on Channel 1 output:

0: No effect of OC5REF on OC1REFC5

1: OC1REFC is the logical AND of OC1REFC and OC5REF

This bit can either have immediate effect or be preloaded and taken into account after an update event (if preload feature is selected in TIMxCCMR1).

*Note: it is also possible to apply this distortion on combined PWM signals.*

Bits 28:16 Reserved, must be kept at reset value.

Bits 15:0 **CCR5[15:0]**: Capture/Compare 5 value

CCR5 is the value to be loaded in the actual capture/compare 5 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR3 register (bit OC5PE). Else the preload value is copied in the active capture/compare 5 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC5 output.

### 22.4.25 TIMx capture/compare register 6 (TIMx\_CCR6)(x = 1, 8)

Address offset: 0x5C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR6[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR6[15:0]**: Capture/Compare 6 value

CCR6 is the value to be loaded in the actual capture/compare 6 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR3 register (bit OC6PE). Else the preload value is copied in the active capture/compare 6 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC6 output.

## 22.4.26 TIM1 register map

TIM1 registers are mapped as 16-bit addressable registers as described in the table below:

**Table 151. TIM1 register map and reset values**

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0x00	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																																				
0x04	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																																				
0x08	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																																				
0x0C	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																																				
0x10	<b>TIM1_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.	
		Reset value																																				
0x14	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																																				
0x18	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																																				
	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
		Reset value																																				
0x1C	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
		Reset value																																				
	<b>TIM1_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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
		Reset value																																				
0x20	<b>TIM1_CCER</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
		Reset value																																				

Table 151. TIM1 register map and reset values (continued)

Offset	Register name	Reset value	UIFCP	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
0x24	TIM1_CNT		Res.	CNT[15:0]																																							
	Reset value	0	Res.																																								
0x28	TIM1_PSC		Res.	PSC[15:0]																																							
	Reset value	0	Res.																																								
0x2C	TIM1_ARR		Res.	ARR[15:0]																																							
	Reset value	0	Res.																																								
0x30	TIM1_RCR		Res.	REP[15:0]																																							
	Reset value	0	Res.																																								
0x34	TIM1_CCR1		Res.	CCR1[15:0]																																							
	Reset value	0	Res.																																								
0x38	TIM1_CCR2		Res.	CCR2[15:0]																																							
	Reset value	0	Res.																																								
0x3C	TIM1_CCR3		Res.	CCR3[15:0]																																							
	Reset value	0	Res.																																								
0x40	TIM1_CCR4		Res.	CCR4[15:0]																																							
	Reset value	0	Res.																																								
0x44	TIM1_BDTR	Res.	BK2F[3:0]	BKF[3:0]	DT[7:0]																																						
	Reset value	0	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				
0x48	TIM1_DCR	Res.	0	BK2P	DBL[4:0]	DBA[4:0]																																					
	Reset value	0	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		
0x4C	TIM1_DMAR			DMAB[31:0]																																							
	Reset value	0	Res.																																								
0x54	TIM1_CCMR3 Output Compare mode	Res.	GC5C3	GC5C2	GC5C1	OC6M[2:0]	OC6E	OC6CE	0	0	0	0	0	0	0	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	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		
0x58	TIM1_CCR5	Res.	0	OC6M[3]	OC6CE	0	0	0	0	0	0	0	0	0	0	0	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	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	
0x5C	TIM1_CCR6	Res.	0	OC6M[3]	OC6CE	0	0	0	0	0	0	0	0	0	0	0	0	0	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	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

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

### 22.4.27 TIM8 register map

TIM8 registers are mapped as 16-bit addressable registers as described in the table below:

**Table 152. TIM8 register map and reset values**

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	<b>TIM8_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.	Res.	Res.	Res.	Res.
	Reset value																																		
0x04	<b>TIM8_CR2</b>																																		
	Reset value																																		
0x08	<b>TIM8_SMCR</b>																																		
	Reset value																																		
0x0C	<b>TIM8_DIER</b>																																		
	Reset value																																		
0x10	<b>TIM8_SR</b>																																		
	Reset value																																		
0x14	<b>TIM8_EGR</b>																																		
	Reset value																																		
0x18	<b>TIM8_CCMR1</b> Output Compare mode	0	OC2M[3]	Res.																															
	Reset value																																		
	<b>TIM8_CCMR1</b> Input Capture mode	0	OC4M[3]	Res.																															
0x1C	<b>TIM8_CCMR2</b> Output Compare mode	0	OC4CE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value																																		
	<b>TIM8_CCMR2</b> Input Capture mode	0	IC4F[3:0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	<b>TIM8_CCER</b>	0	CC4NP	0	0	0	0	0	0	0	0	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																																		

**Table 152. TIM8 register map and reset values (continued)**

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 23 General-purpose timers (TIM2/TIM3/TIM4/TIM5)

### 23.1 TIM2/TIM3/TIM4/TIM5 introduction

The general-purpose timers consist of a 16-bit/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.

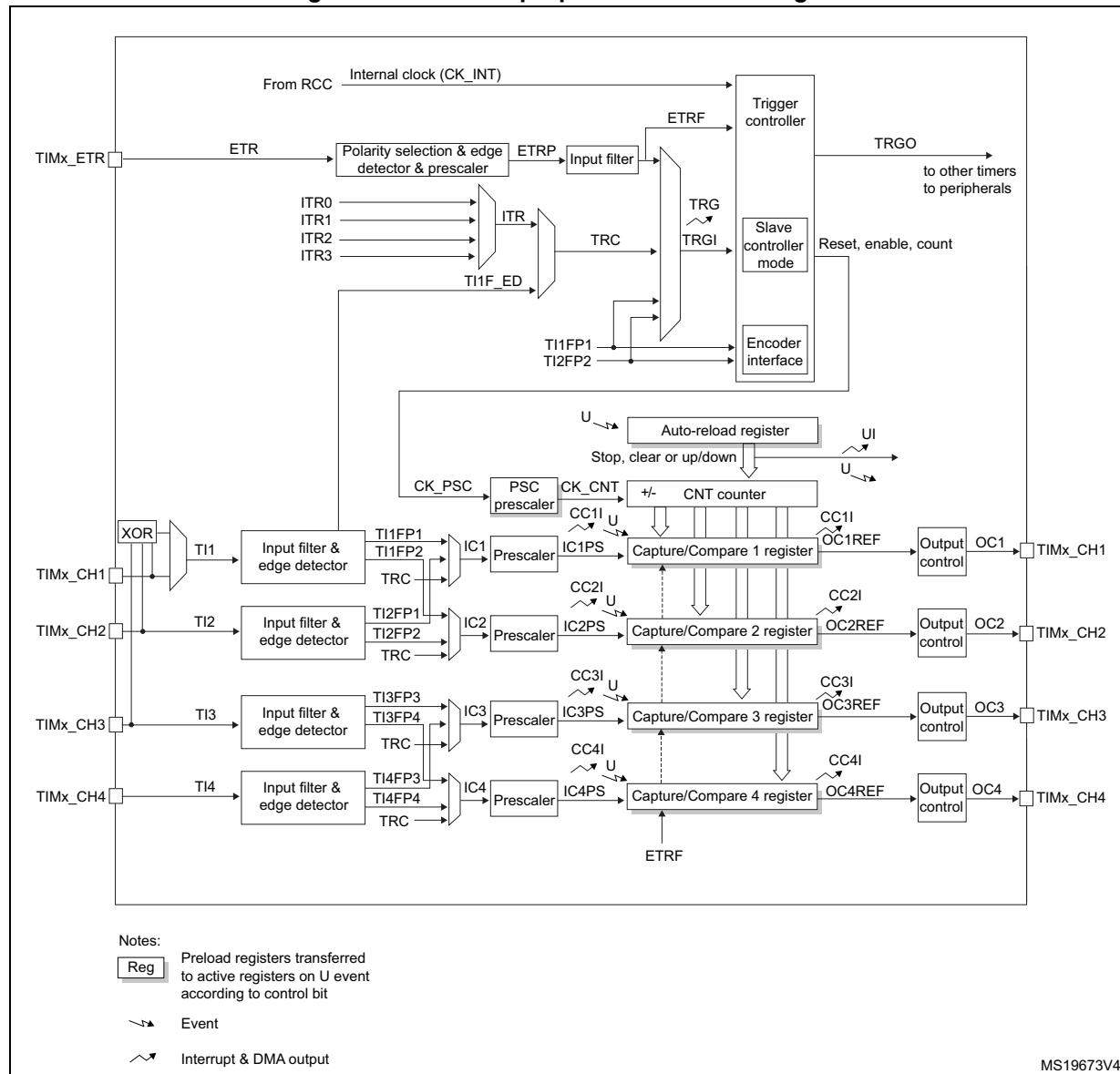
The timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 23.3.19: Timer synchronization](#).

### 23.2 TIM2/TIM3/TIM4/TIM5 main features

General-purpose TIMx timer features include:

- 16-bit (TIM3, TIM4) or 32-bit (TIM2 and TIM5) up, down, up/down auto-reload counter.
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535.
- Up to 4 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (Edge- and Center-aligned modes)
  - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers.
- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
- Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

Figure 203. General-purpose timer block diagram



## 23.3 TIM2/TIM3/TIM4/TIM5 functional description

### 23.3.1 Time-base unit

The main block of the programmable timer is a 16-bit/32-bit counter with its related auto-reload register. The counter can count up, down or both up and down. 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.

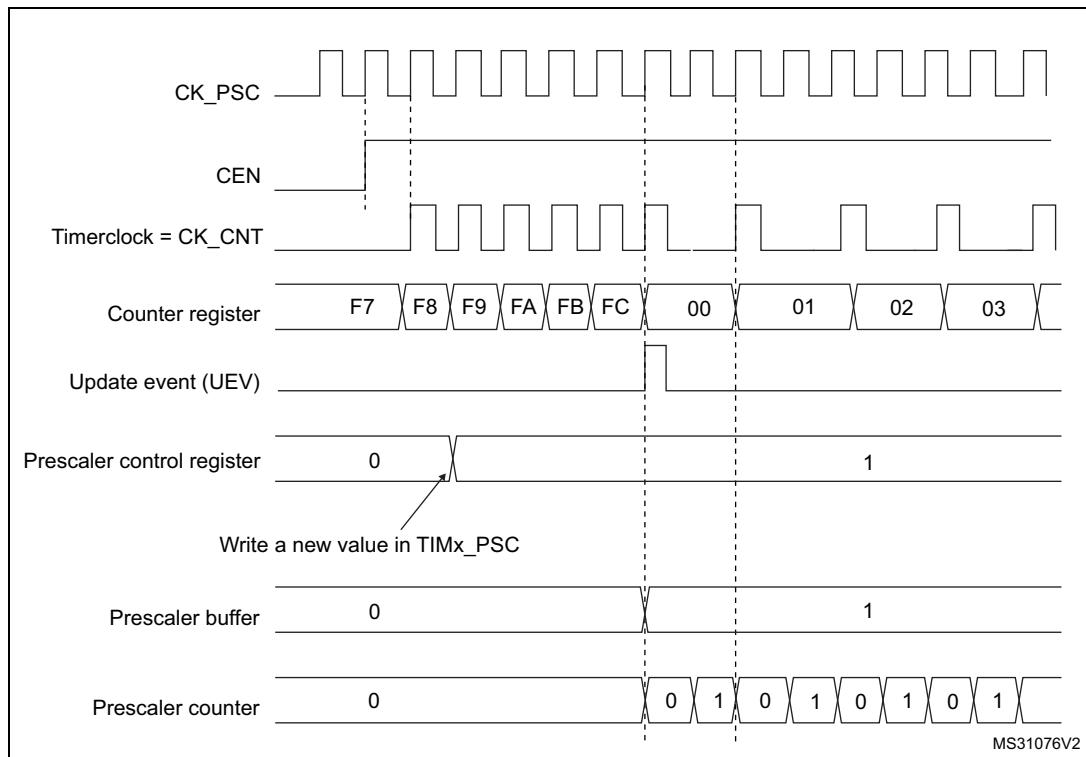
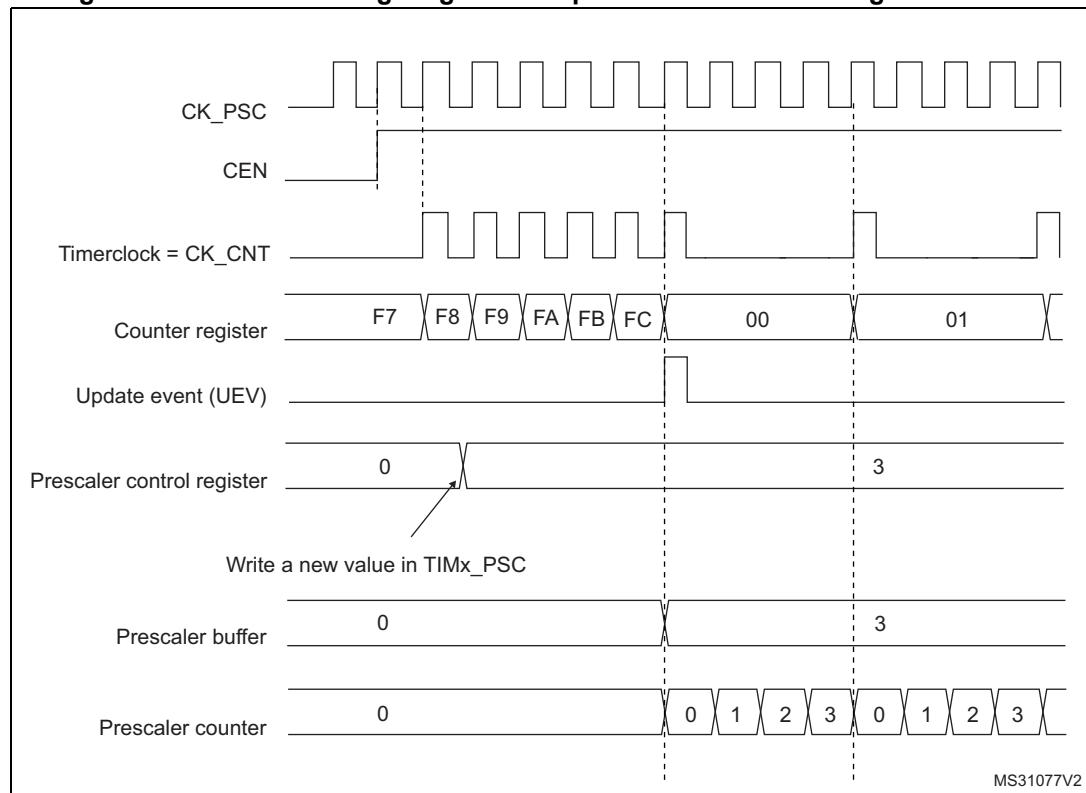
The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

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 204* and *Figure 205* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 204. Counter timing diagram with prescaler division change from 1 to 2****Figure 205. Counter timing diagram with prescaler division change from 1 to 4**

### 23.3.2 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).

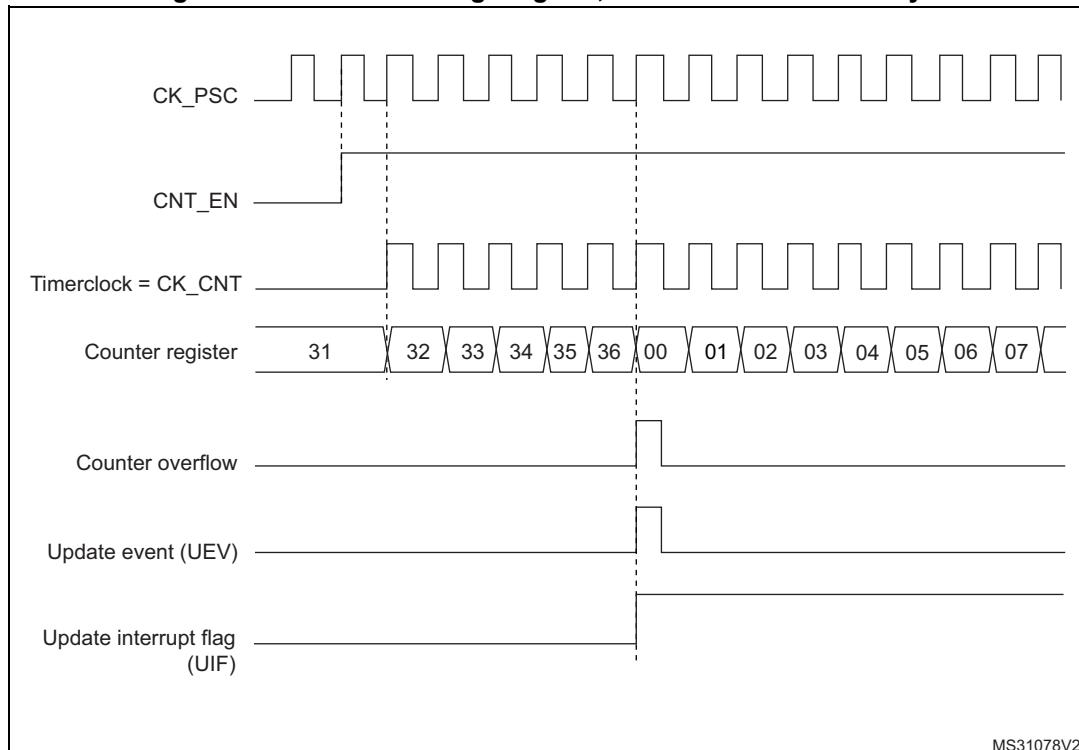
The UEV event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well 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.

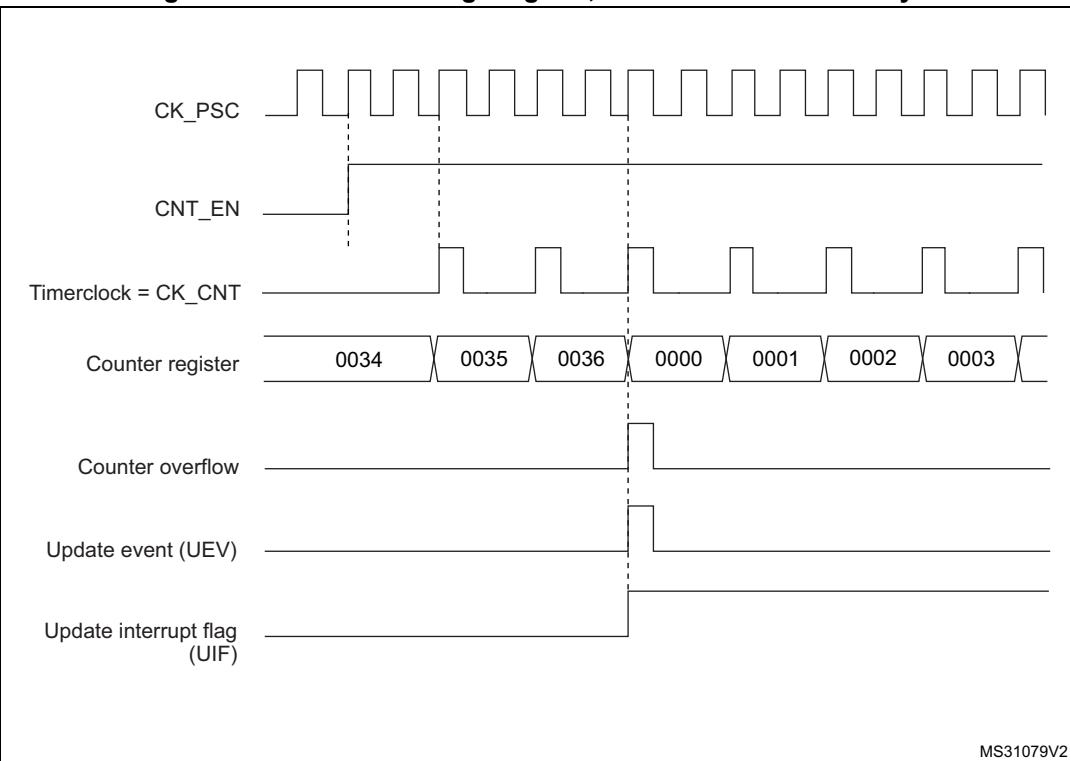
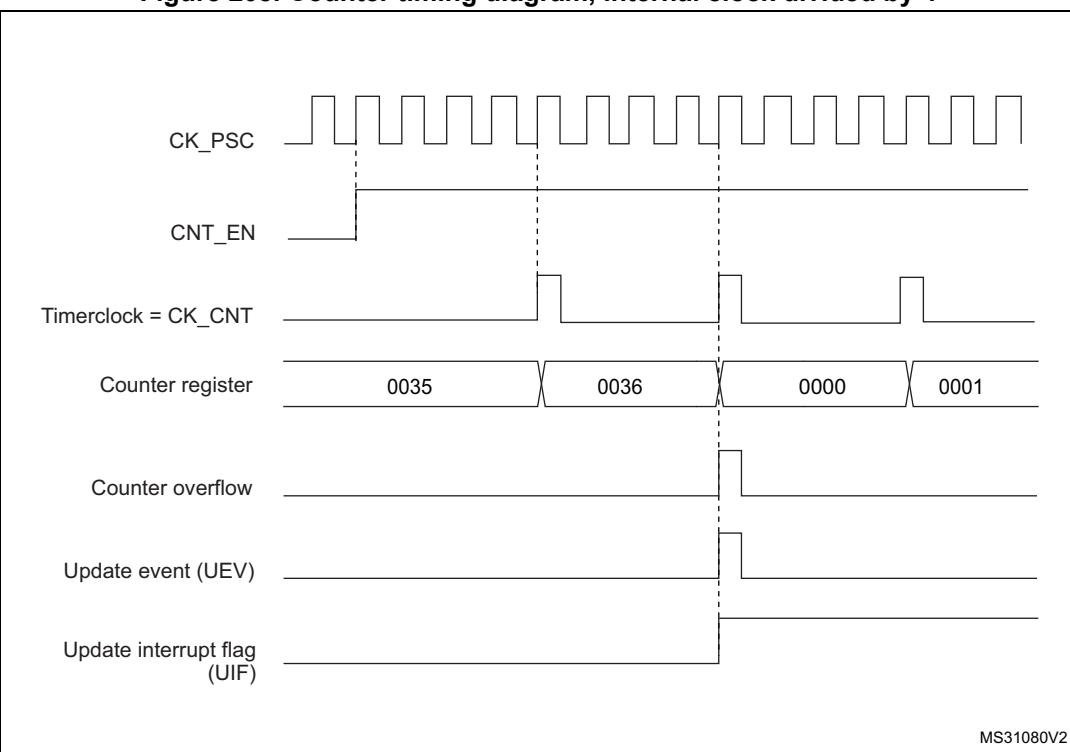
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

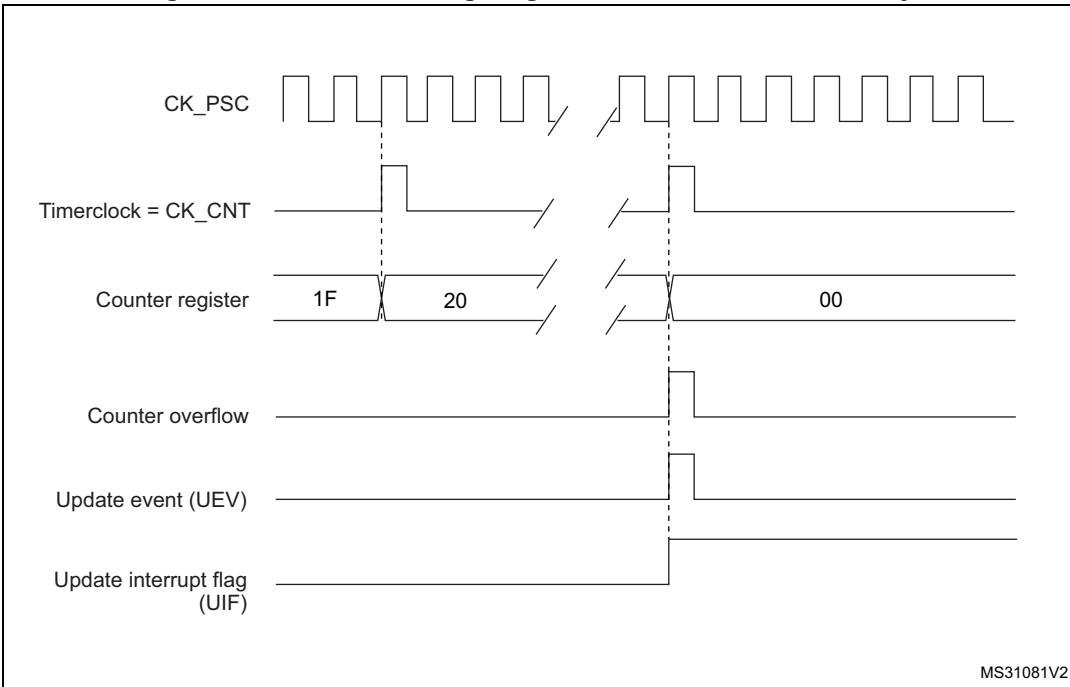
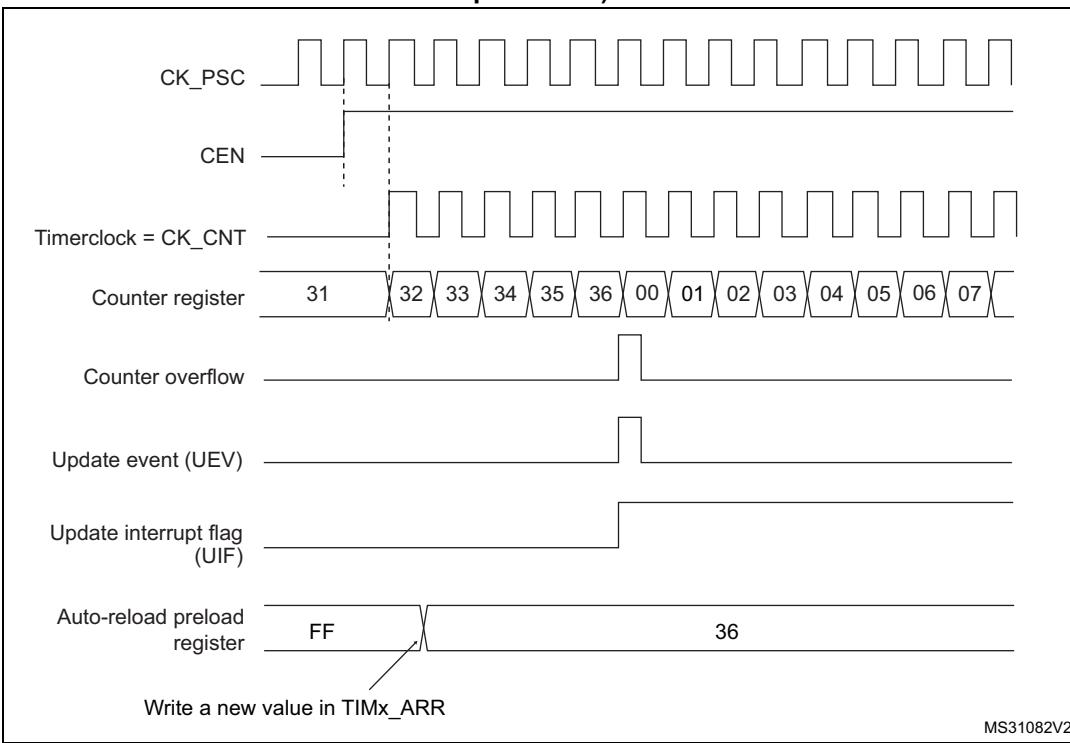
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR)

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

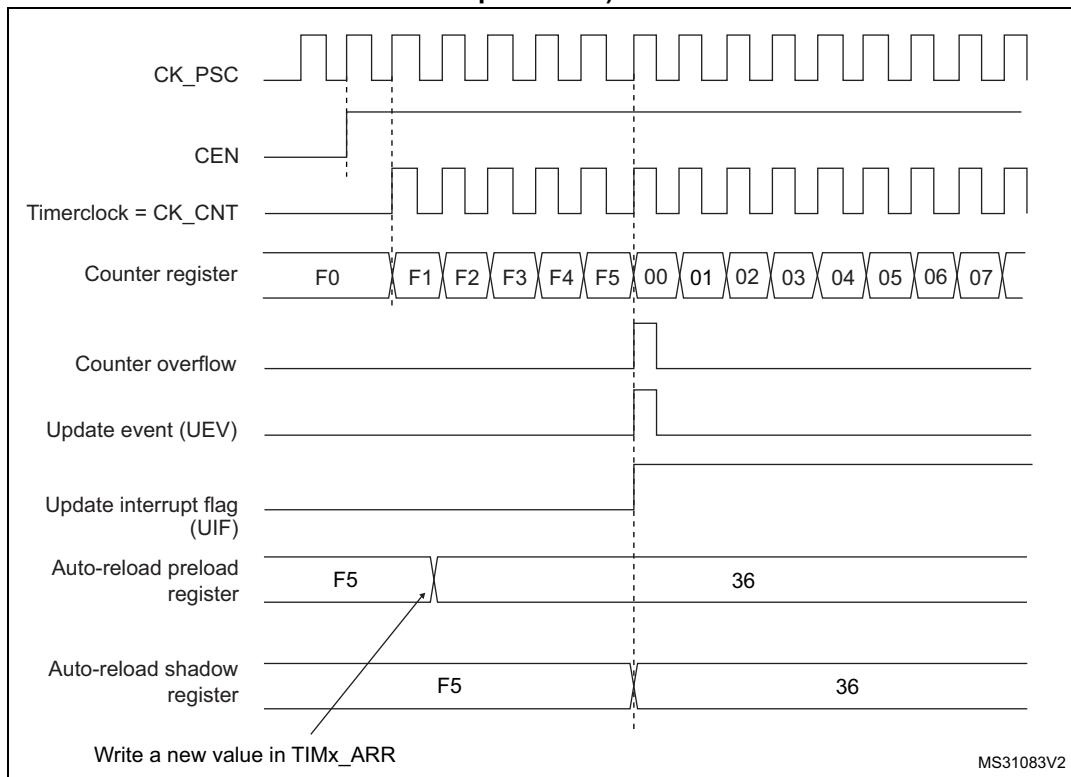
**Figure 206. Counter timing diagram, internal clock divided by 1**



**Figure 207. Counter timing diagram, internal clock divided by 2****Figure 208. Counter timing diagram, internal clock divided by 4**

**Figure 209. Counter timing diagram, internal clock divided by N****Figure 210. Counter timing diagram, Update event when ARPE=0 (TIMx\_ARR not preloaded)**

**Figure 211. 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).

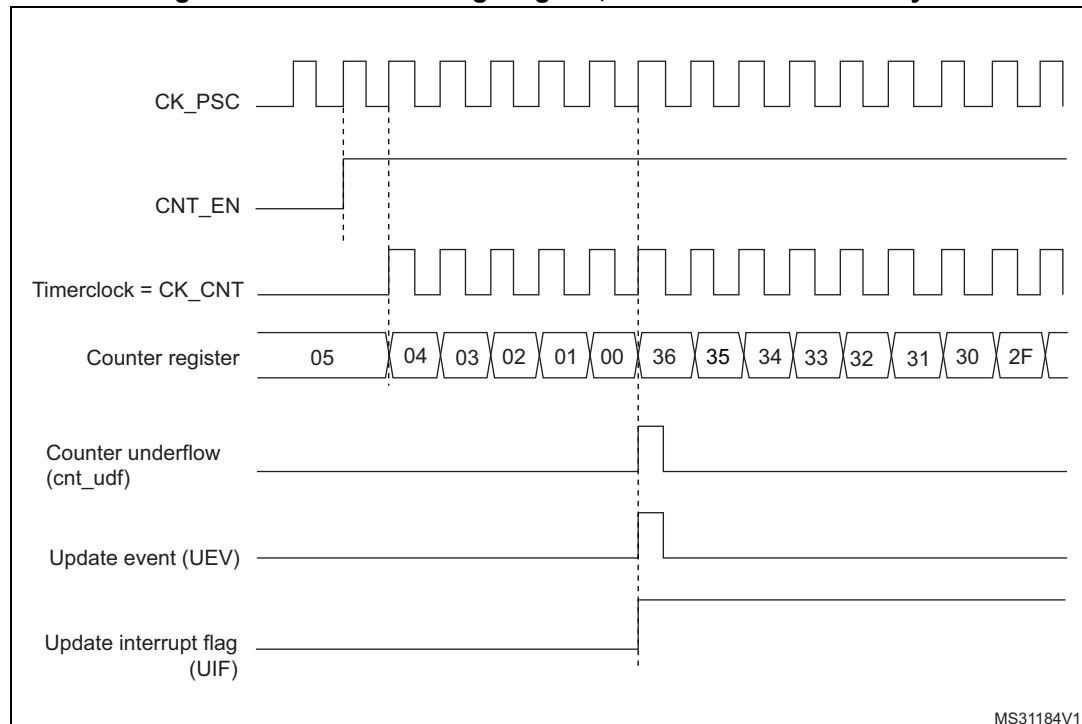
In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on 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):

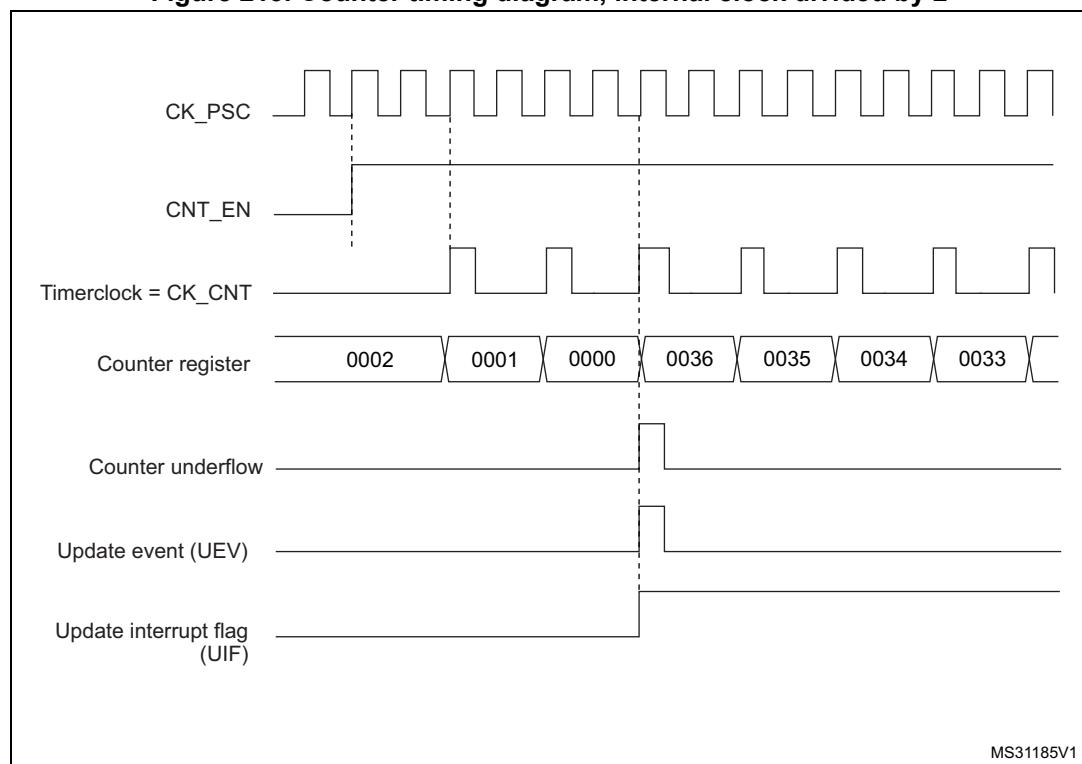
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

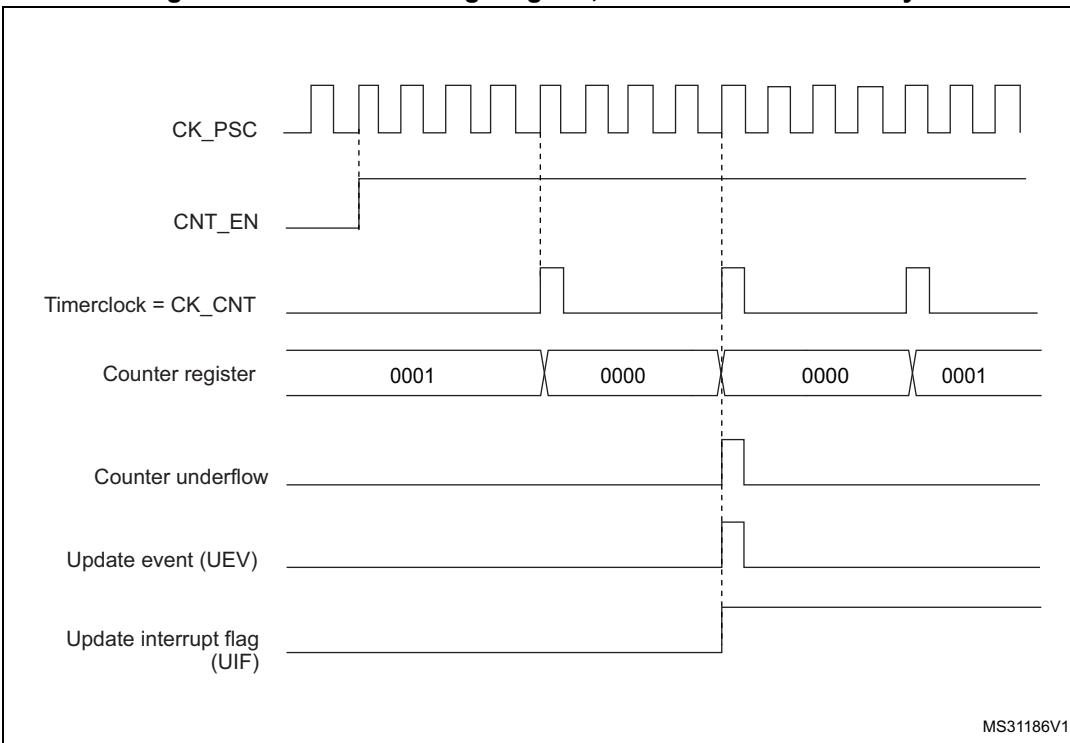
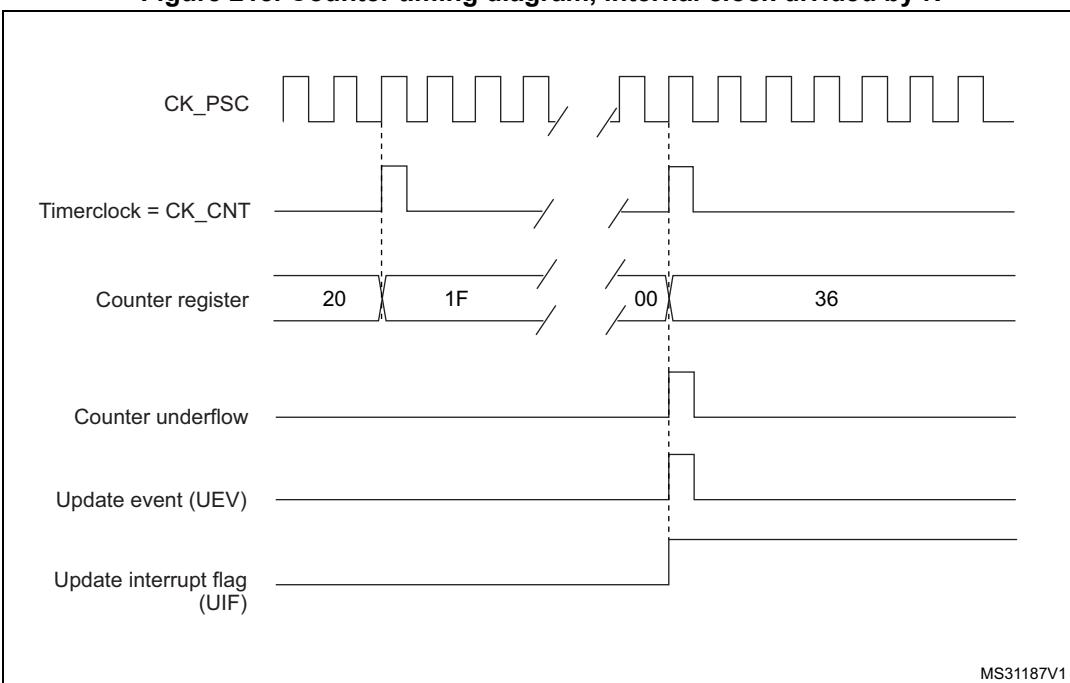
The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

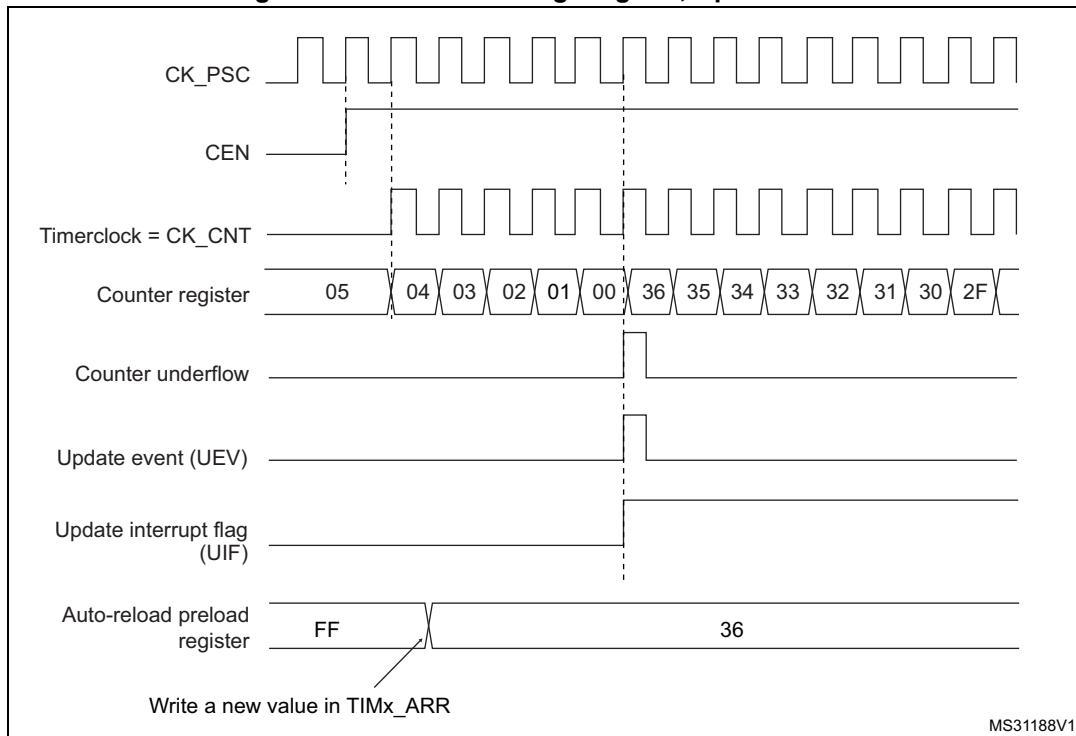
**Figure 212. Counter timing diagram, internal clock divided by 1**



**Figure 213. Counter timing diagram, internal clock divided by 2**



**Figure 214. Counter timing diagram, internal clock divided by 4****Figure 215. Counter timing diagram, internal clock divided by N**

**Figure 216. 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") or 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

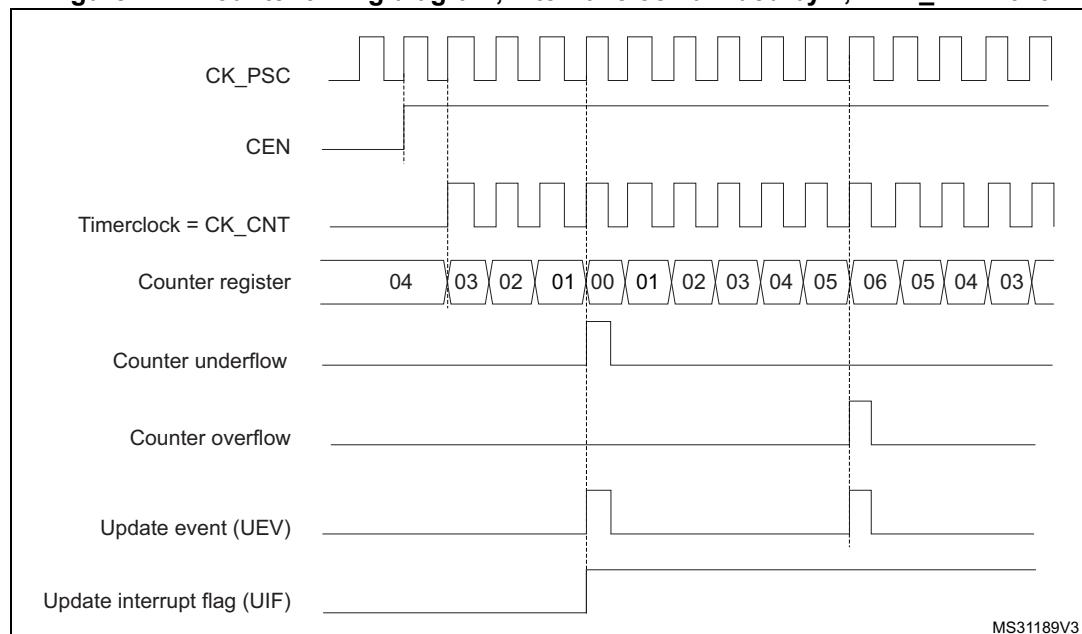
DMA request is sent). This is to avoid generating both update and capture interrupt when clearing the counter on the capture event.

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):

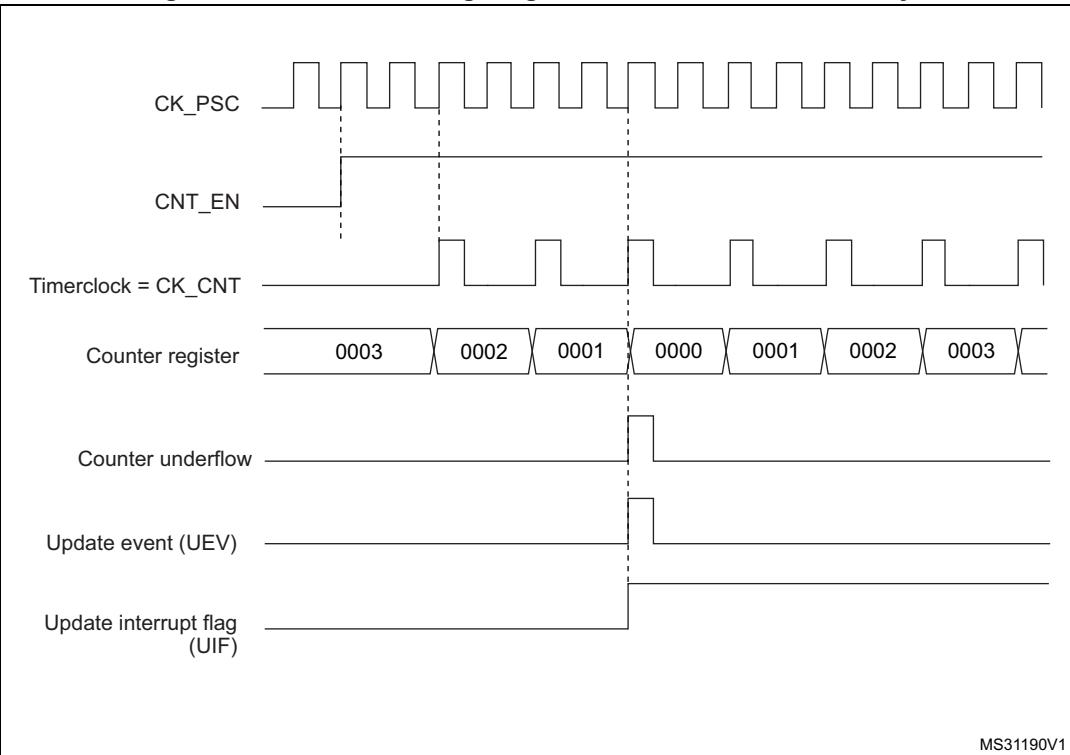
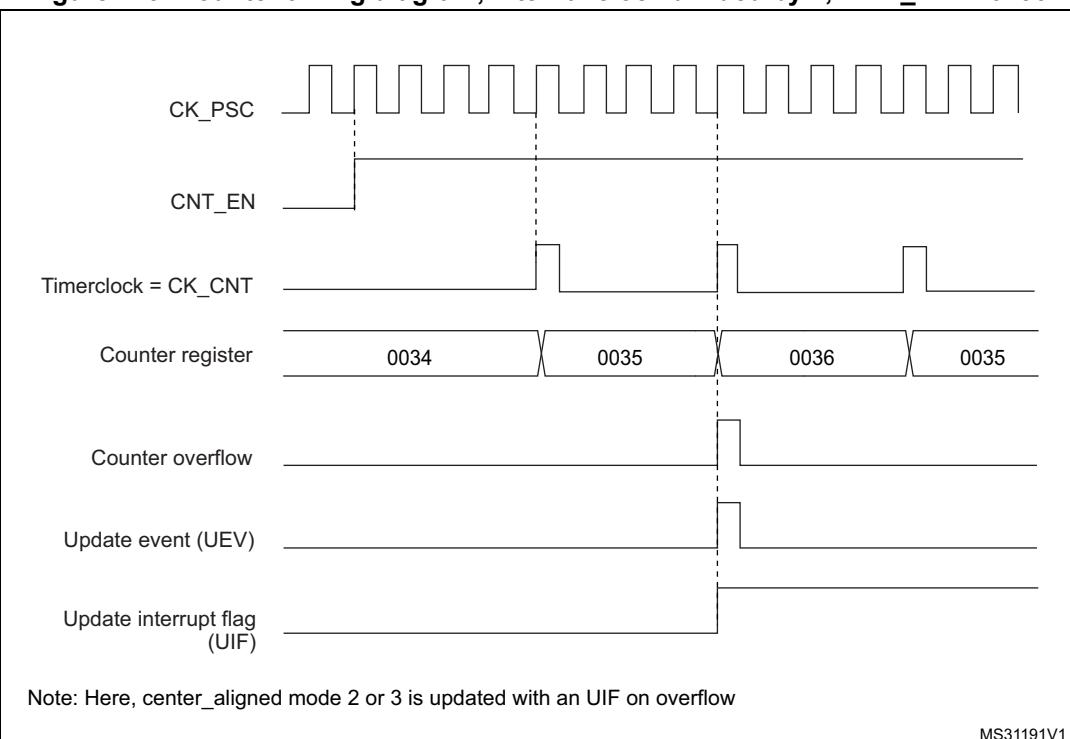
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter 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 217. 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 23.4.1: TIMx control register 1 \(TIMx\\_CR1\)\(x = 2 to 5\) on page 777](#)).

**Figure 218. Counter timing diagram, internal clock divided by 2****Figure 219. 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 220. Counter timing diagram, internal clock divided by N

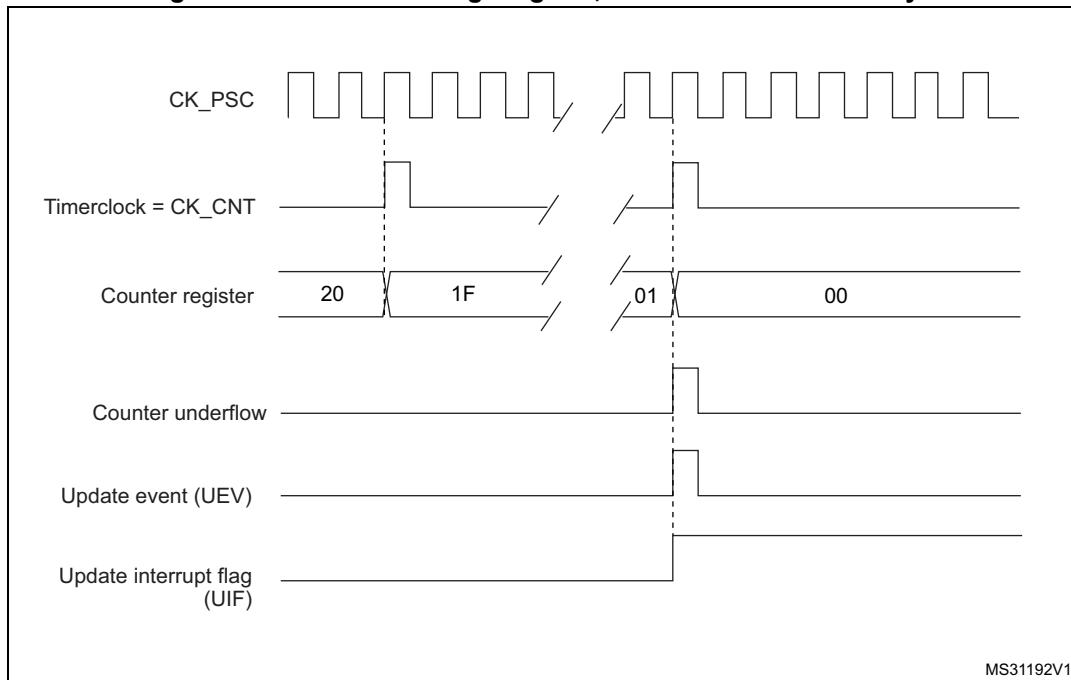
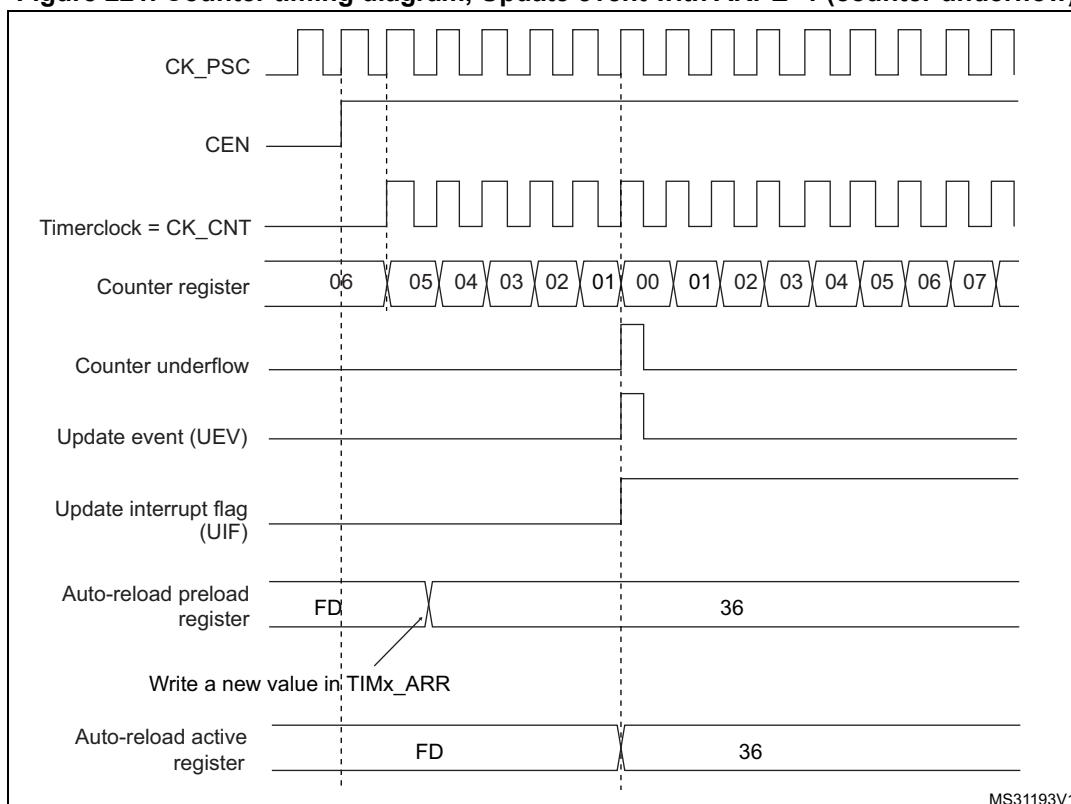
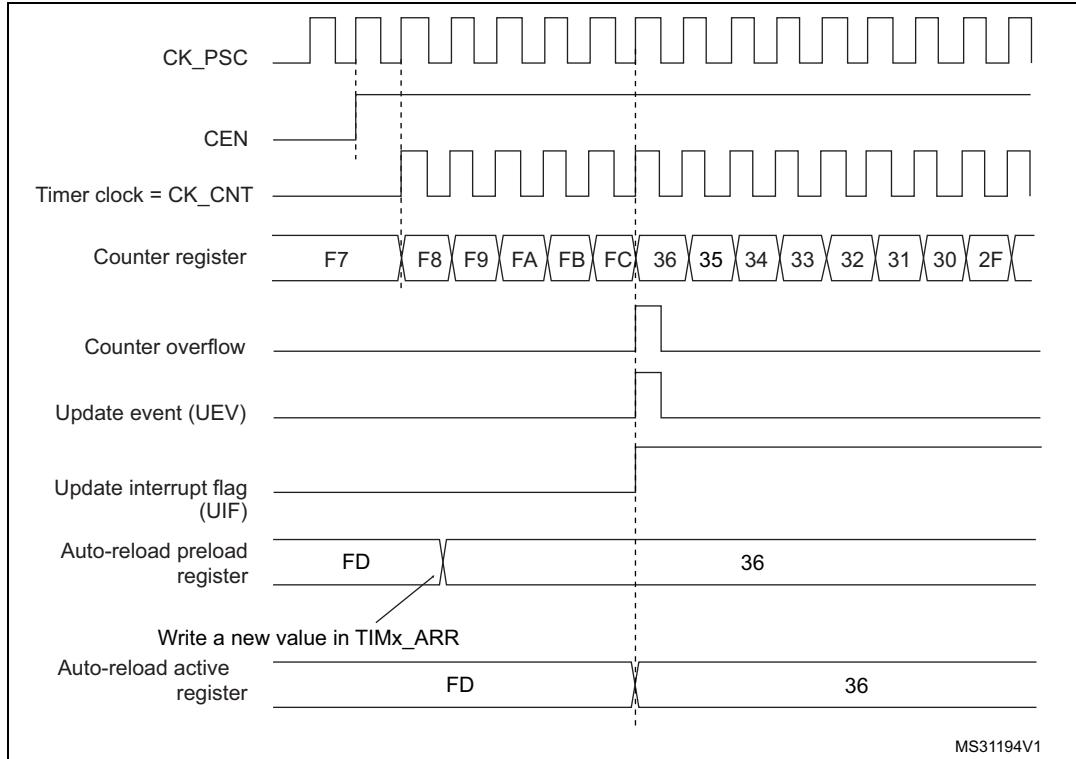


Figure 221. Counter timing diagram, Update event with ARPE=1 (counter underflow)



**Figure 222. Counter timing diagram, Update event with ARPE=1 (counter overflow)**

### 23.3.3 Clock selection

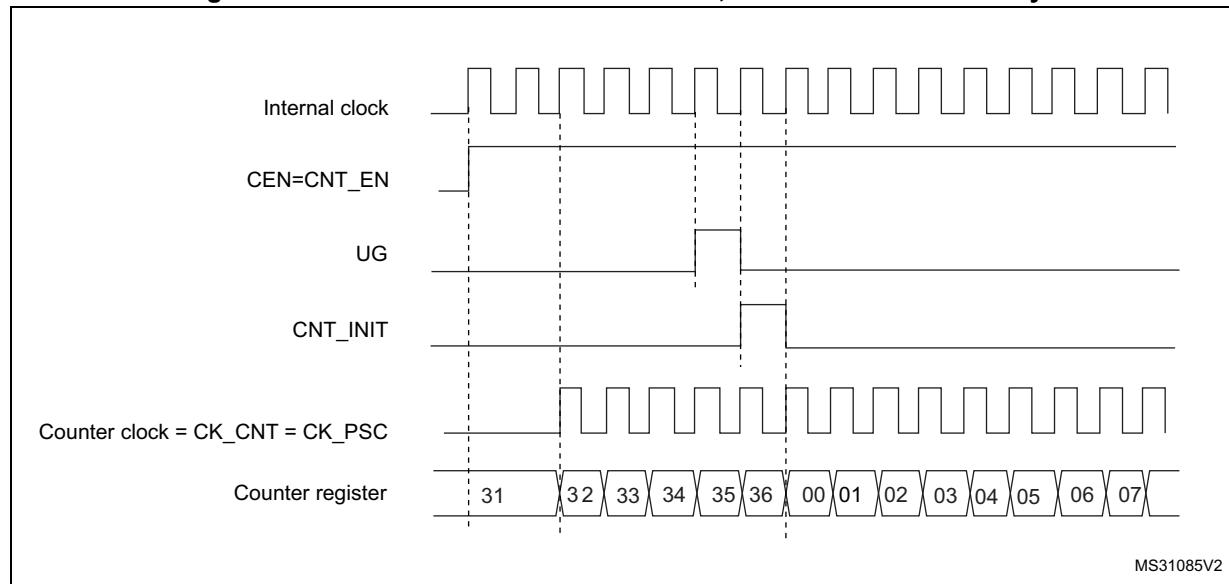
The counter clock can be provided by the following clock sources:

- Internal clock (CK\_INT)
- External clock mode1: external input pin (TI<sub>x</sub>)
- External clock mode2: external trigger input (ETR)
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, Timer X can be configured to act as a prescaler for Timer Y. Refer to : [Using one timer as prescaler for another timer on page 771](#) for more details.

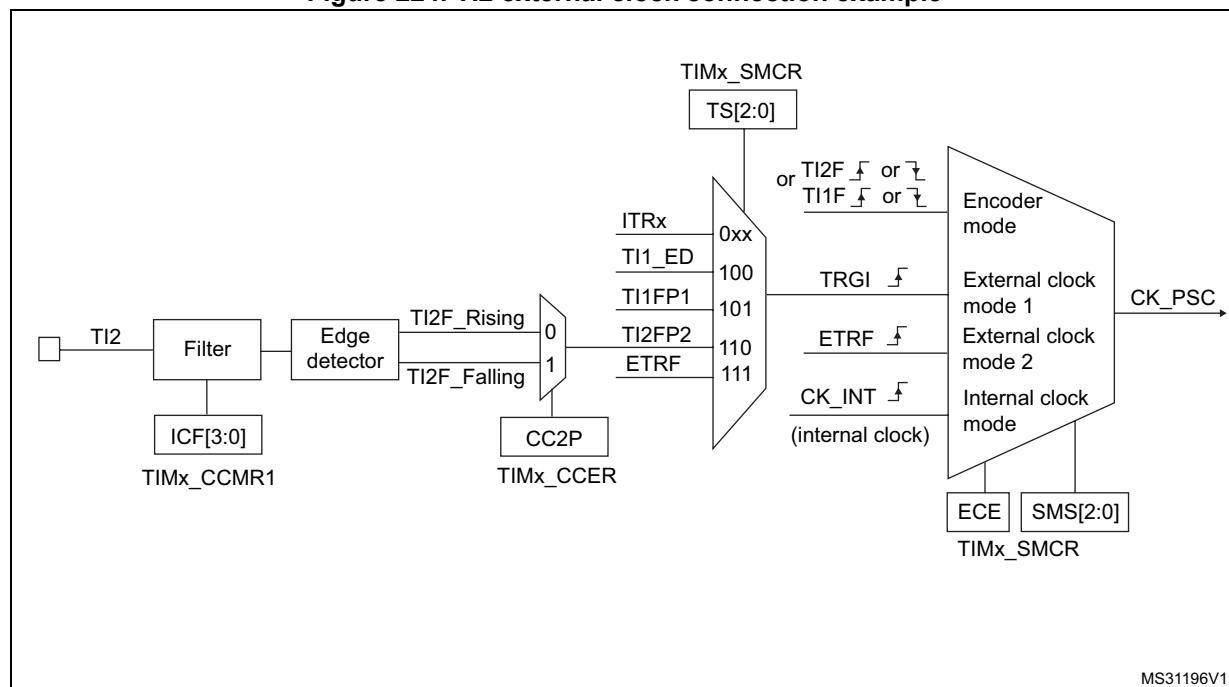
#### Internal clock source (CK\_INT)

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 223](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

**Figure 223. 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 224. 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).

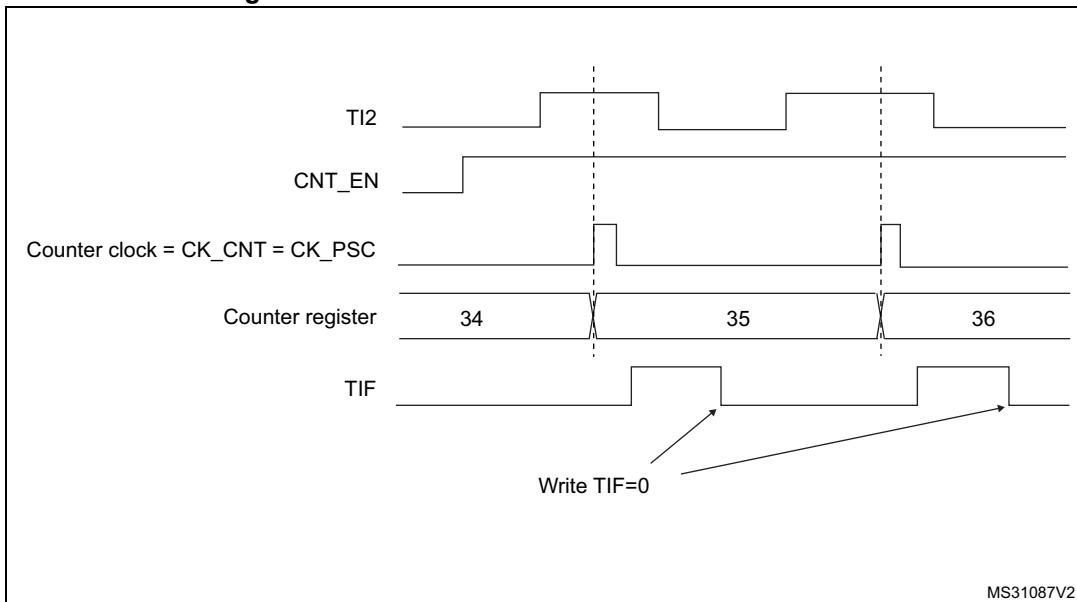
**Note:** *The capture prescaler is not used for triggering, so it does not need to be configured.*

3. Select rising edge polarity by writing CC2P=0 and CC2NP=0 and CC2NP=0 in the TIMx\_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx\_SMCR register.
5. Select TI2 as the input source by writing TS=110 in the TIMx\_SMCR register.
6. Enable the counter by writing CEN=1 in the TIMx\_CR1 register.

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

The delay between the rising edge on TI2 and the actual clock of the counter is due to the resynchronization circuit on TI2 input.

**Figure 225. Control circuit in external clock mode 1**



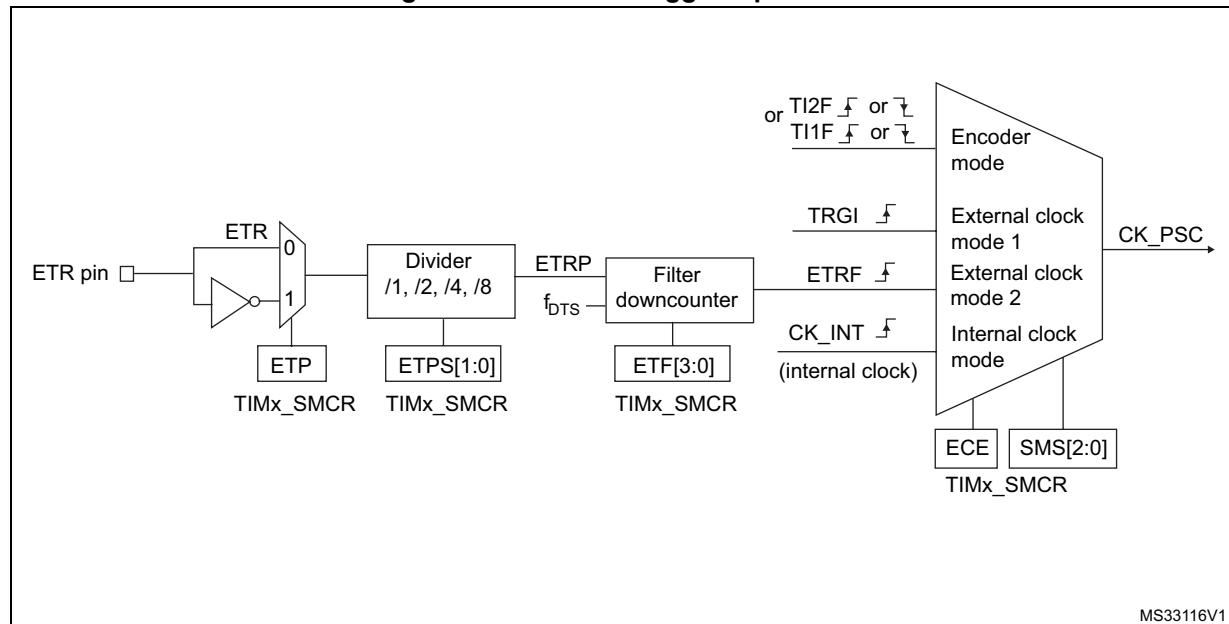
### External clock source mode 2

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 226](#) gives an overview of the external trigger input block.

Figure 226. External trigger input block

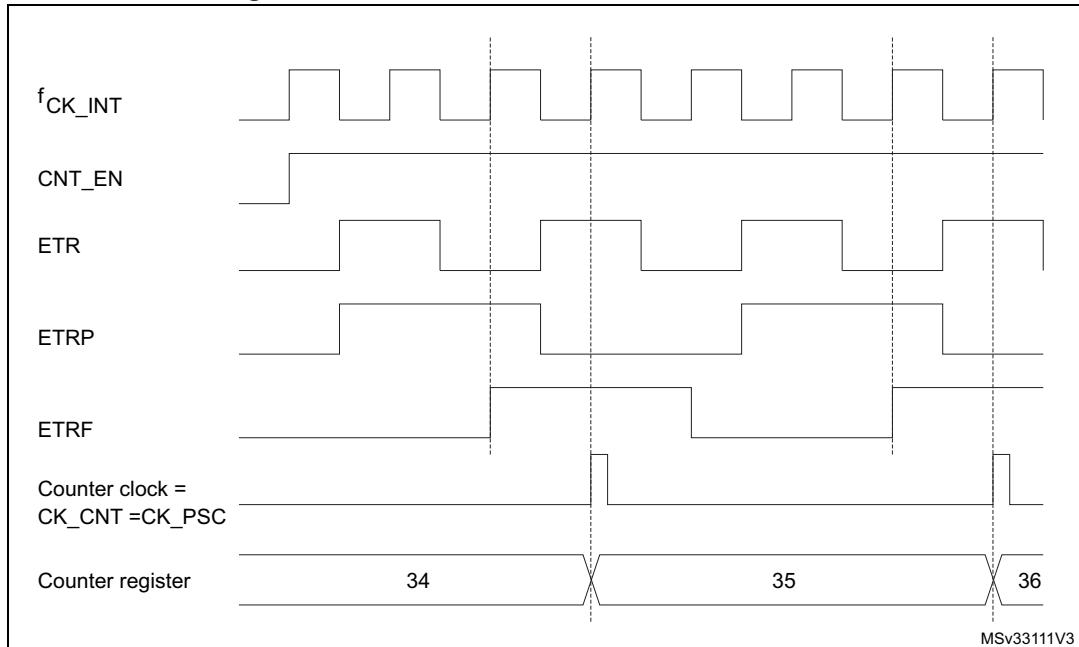


For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

1. As no filter is needed in this example, write ETF[3:0]=0000 in the TIMx\_SMCR register.
2. Set the prescaler by writing ETSPS[1:0]=01 in the TIMx\_SMCR register
3. Select rising edge detection on the ETR pin by writing ETP=0 in the TIMx\_SMCR register
4. Enable external clock mode 2 by writing ECE=1 in the TIMx\_SMCR register.
5. Enable the counter by writing CEN=1 in the TIMx\_CR1 register.

The counter counts once each 2 ETR rising edges.

The delay between the rising edge on ETR and the actual clock of the counter is due to the resynchronization circuit on the ETRP signal. As a consequence, the maximum frequency which can be correctly captured by the counter is at most ¼ of TIMxCLK frequency. When the ETRP signal is faster, the user should apply a division of the external signal by a proper ETSPS prescaler setting.

**Figure 227. Control circuit in external clock mode 2**

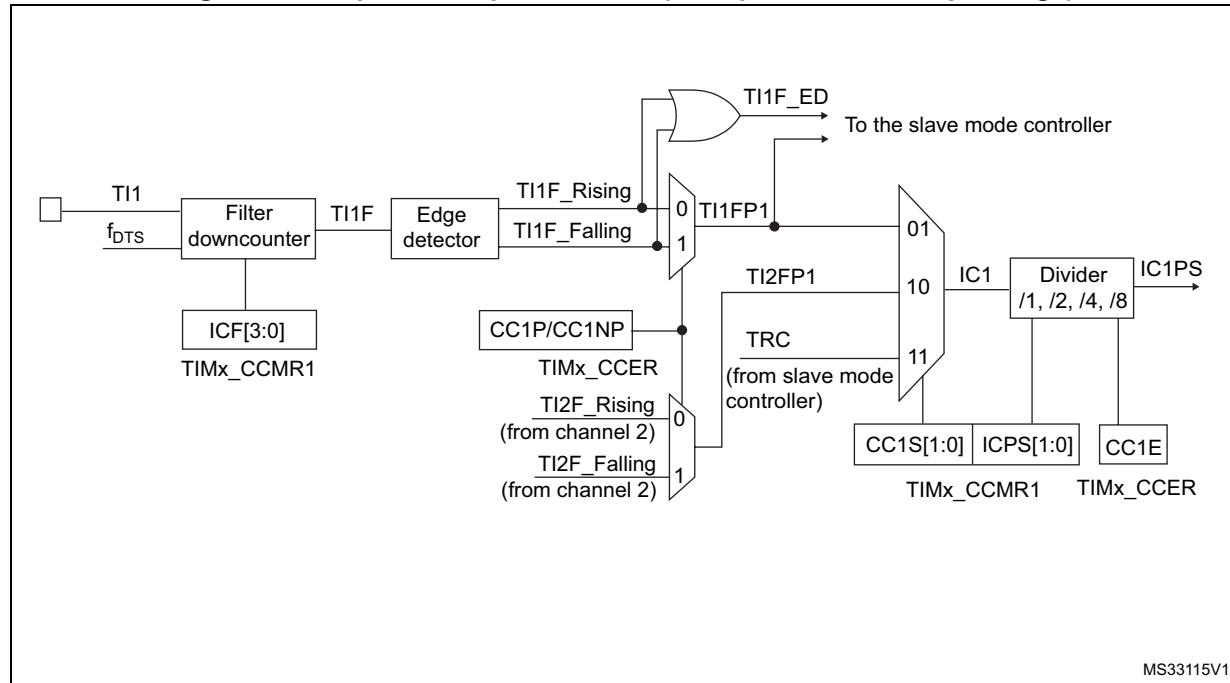
### 23.3.4 Capture/Compare channels

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.

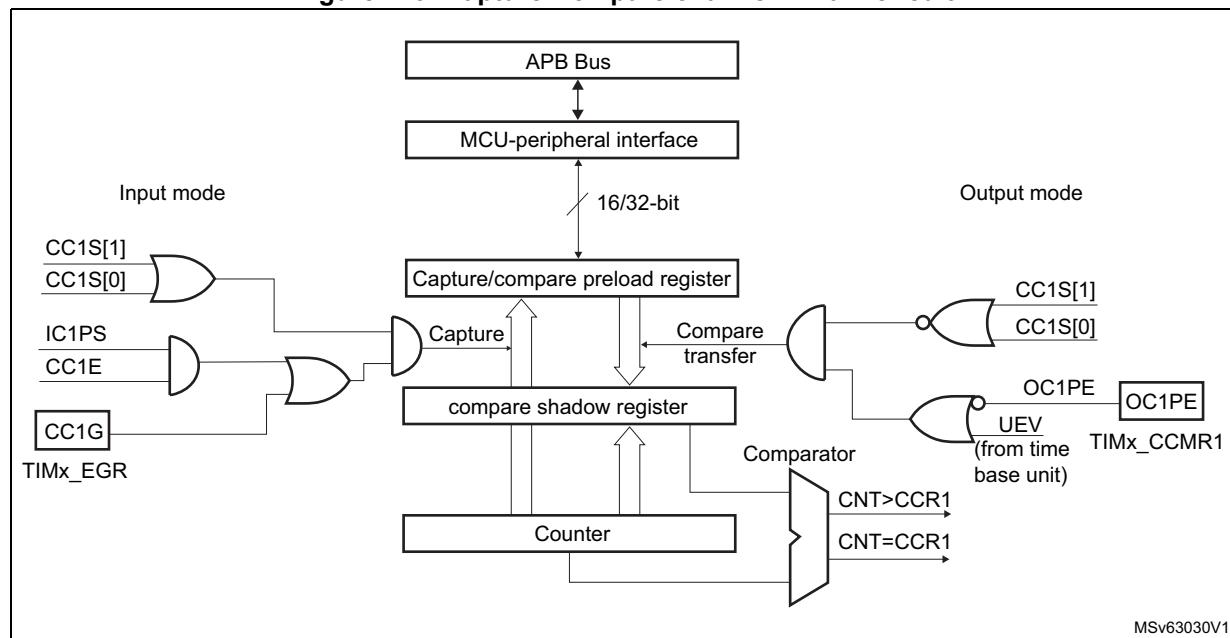
The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

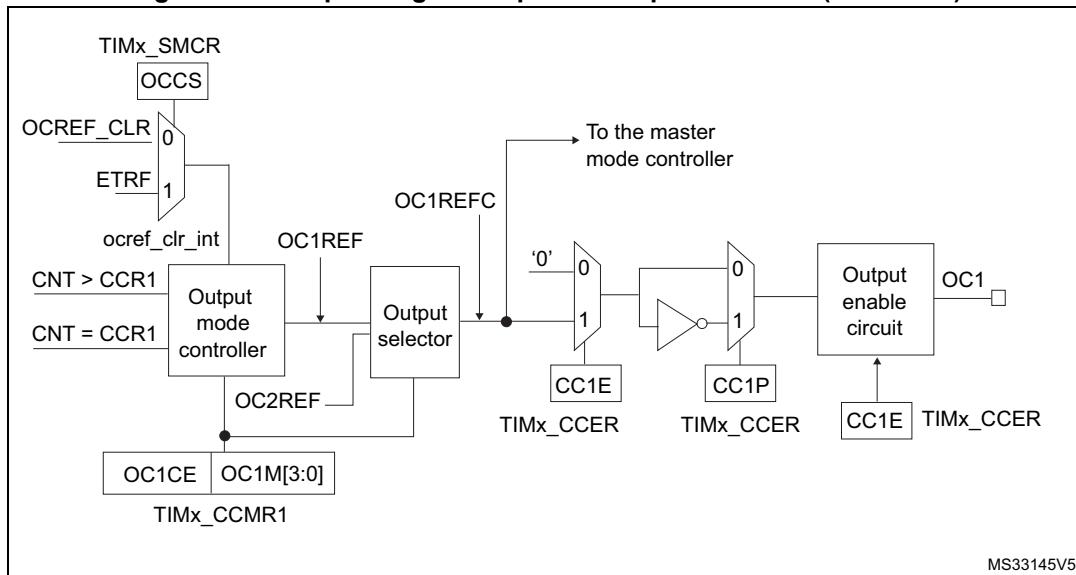
Figure 228. Capture/Compare channel (example: channel 1 input stage)



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 229. Capture/Compare channel 1 main circuit



**Figure 230. 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.

### 23.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCxIF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx\_SR register) is set. CCxIF can be cleared by software by writing it to 0 or by reading the captured data stored in the TIMx\_CCRx register. CCxOF is cleared when it is written with 0.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

1. Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx\_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx\_CCR1 register becomes read-only.
2. Program the appropriate input filter duration in relation with the signal connected to the timer (when the input is one of the TIx (ICxF bits in the TIMx\_CCMRx register). Let's imagine that, when toggling, the input signal is not stable during at most 5 internal clock cycles. We must program a filter duration longer than these 5 clock cycles. We can validate a transition on TI1 when 8 consecutive samples with the new level have been

detected (sampled at  $f_{DTS}$  frequency). Then write IC1F bits to 0011 in the TIMx\_CCMR1 register.

3. Select the edge of the active transition on the TI1 channel by writing the CC1P and CC1NP and CC1NP bits to 000 in the TIMx\_CCER register (rising edge in this case).
4. Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to 00 in the TIMx\_CCMR1 register).
5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx\_CCER register.
6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx\_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx\_DIER register.

When an input capture occurs:

- The TIMx\_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

*Note: IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx\_EGR register.*

### 23.3.6 PWM input mode

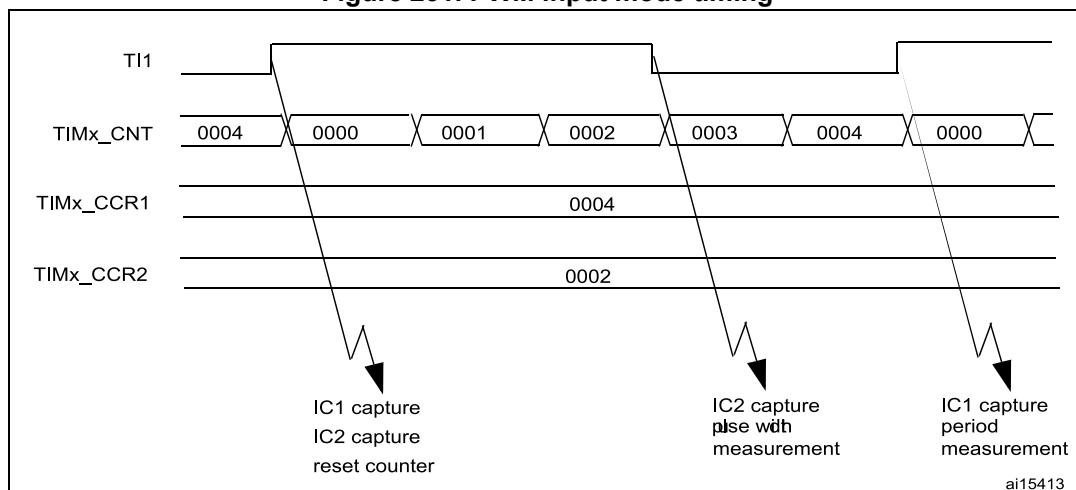
This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, one can measure the period (in TIMx\_CCR1 register) and the duty cycle (in TIMx\_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK\_INT frequency and prescaler value):

1. Select the active input for TIMx\_CCR1: write the CC1S bits to 01 in the TIMx\_CCMR1 register (TI1 selected).
2. Select the active polarity for TI1FP1 (used both for capture in TIMx\_CCR1 and counter clear): write the CC1P to '0' and the CC1NP bit to '0' (active on rising edge).
3. Select the active input for TIMx\_CCR2: write the CC2S bits to 10 in the TIMx\_CCMR1 register (TI1 selected).
4. Select the active polarity for TI1FP2 (used for capture in TIMx\_CCR2): write the CC2P bit to '1' and the CC2NP bit to '0' (active on falling edge).
5. Select the valid trigger input: write the TS bits to 101 in the TIMx\_SMCR register (TI1FP1 selected).
6. Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx\_SMCR register.
7. Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx\_CCER register.

**Figure 231. 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.

### 23.3.7 Forced output mode

In output mode (CCxS bits = 00 in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (ocxref/OCx) to its active level, one just needs to write 101 in the OCxM bits in the corresponding TIMx\_CCMRx register. Thus ocxref is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

e.g.: CCxP=0 (OCx active high) => OCx is forced to high level.

ocxref signal can be forced low by writing the OCxM bits to 100 in the TIMx\_CCMRx register.

Anyway, the comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the Output Compare Mode section.

### 23.3.8 Output compare mode

This function is used to control an output waveform or indicating when a period of time has elapsed.

When a match is found between the capture/compare register and the counter, the output compare function:

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCXM=000), be set active (OCXM=001), be set inactive (OCXM=010) or can toggle (OCXM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx\_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx\_DIER register, CCDS bit in the TIMx\_CR2 register for the DMA request selection).

The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

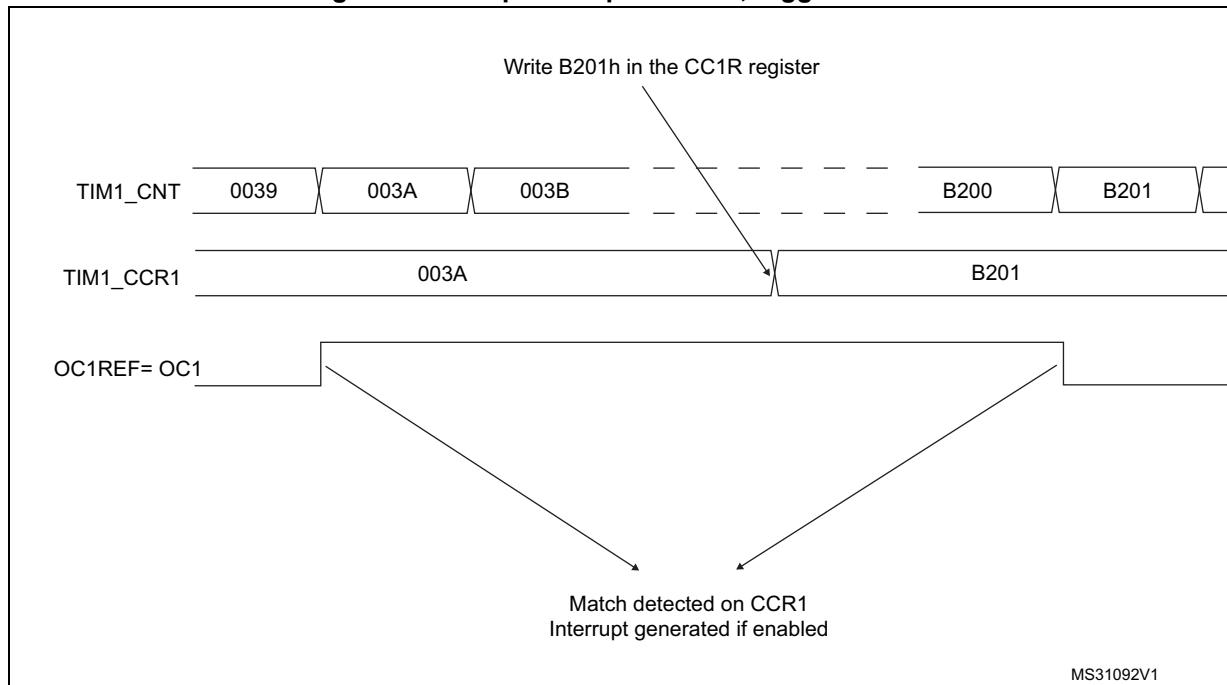
In output compare mode, the update event UEV has no effect on OCxref and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One-pulse mode).

#### Procedure

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx\_ARR and TIMx\_CCRx registers.
3. Set the CCxIE and/or CCxDE bits if an interrupt and/or a DMA request is to be generated.
4. Select the output mode. For example, one must write OCXM=011, OCxPE=0, CCxP=0 and CCxE=1 to toggle OCx output pin when CNT matches CCRx, CCRx preload is not used, OCx is enabled and active high.
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

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 232](#).

Figure 232. Output compare mode, toggle on OC1



### 23.3.9 PWM mode

Pulse width modulation mode permits to generate a signal with a frequency determined by the value of the TIMx\_ARR register and a duty cycle determined by the value of the TIMx\_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or '111 (PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register. The corresponding preload register must be enabled by setting the OCxPE bit in the TIMx\_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx\_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, all registers must be initialized by setting the UG bit in the TIMx\_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx\_CCER register. It can be programmed as active high or active low. OCx output is enabled by the CCxE bit in the TIMx\_CCER register. Refer to the TIMx\_CCERx register description for more details.

In PWM mode (1 or 2), TIMx\_CNT and TIMx\_CCRx are always compared to determine whether TIMx\_CCRx ≤ TIMx\_CNT or TIMx\_CNT ≤ TIMx\_CCRx (depending on the direction of the counter). However, to comply with the OCREF\_CLR functionality (OCREF can be cleared by an external event through the ETR signal until the next PWM period), the OCREF signal is asserted only:

- When the result of the comparison 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.

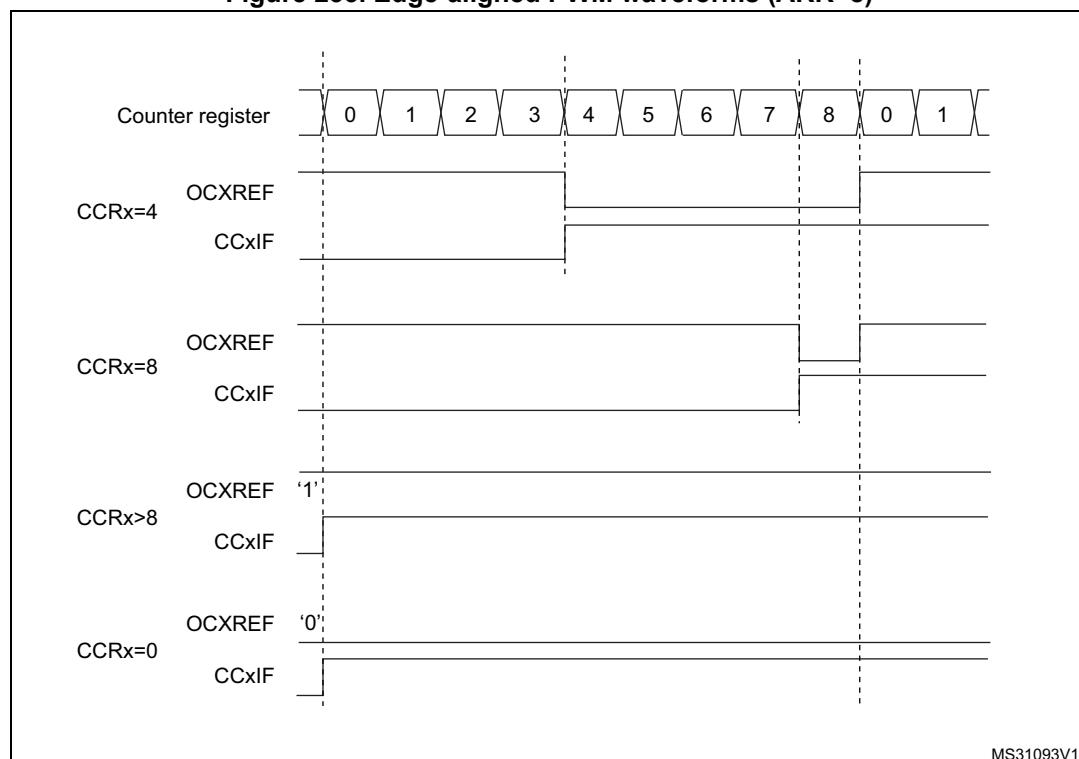
### PWM edge-aligned mode

#### Upcounting configuration

Upcounting is active when the DIR bit in the TIMx\_CR1 register is low. Refer to [Upcounting mode on page 735](#).

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 233](#) shows some edge-aligned PWM waveforms in an example where TIMx\_ARR=8.

**Figure 233. 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 738](#).

In PWM mode 1, the reference signal ocxref is low as long as TIMx\_CNT>TIMx\_CCRx else it becomes high. If the compare value in TIMx\_CCRx is greater than the auto-reload value in TIMx\_ARR, then ocxref is held at 100%. PWM is not possible in this mode.

### PWM center-aligned mode

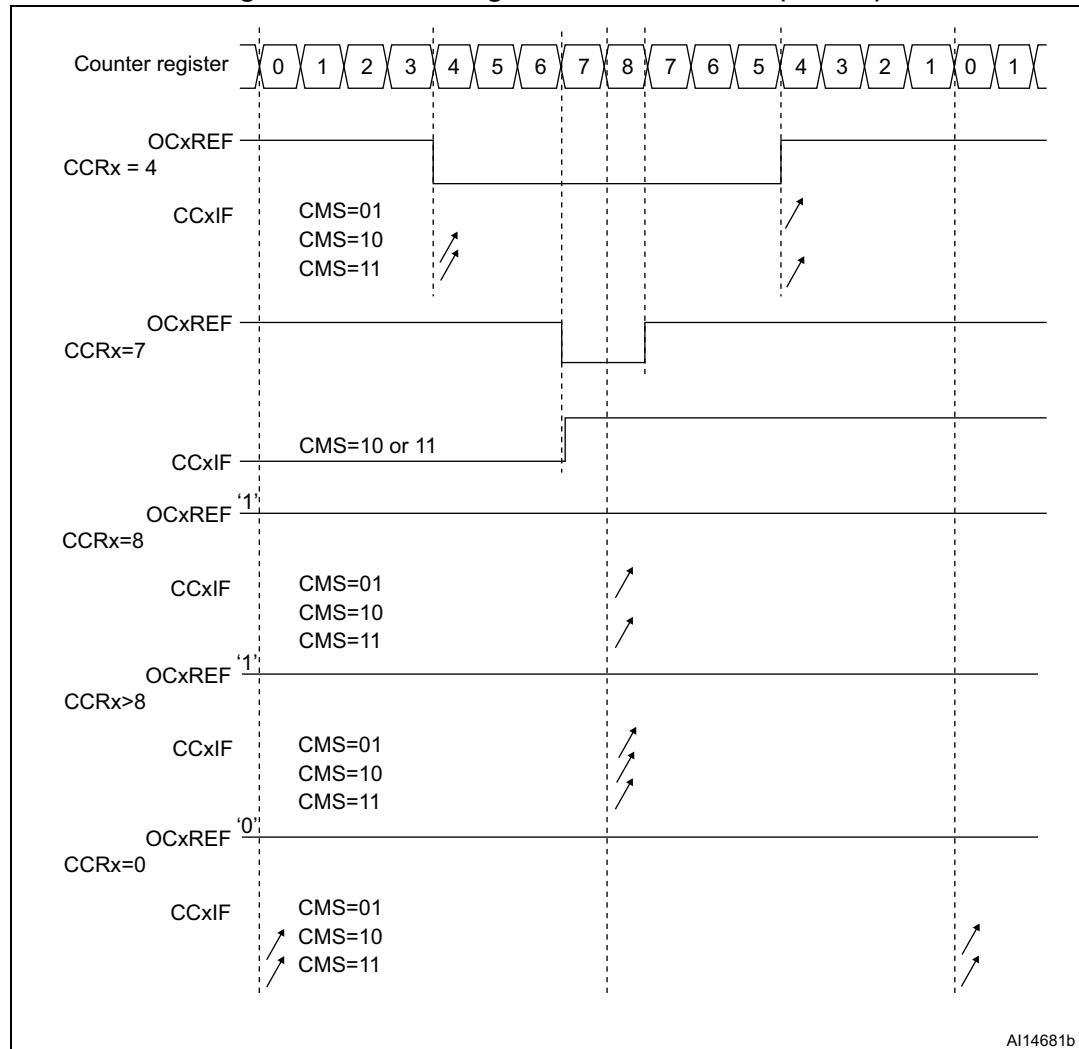
Center-aligned mode is active when the CMS bits in TIMx\_CR1 register are different from '00 (all the remaining configurations having the same effect on the ocxref/OCx signals). The

compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIMx\_CR1 register is updated by hardware and must not be changed by software. Refer to [Center-aligned mode \(up/down counting\) on page 741](#).

[Figure 234](#) 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 234. Center-aligned PWM waveforms (ARR=8)**



Hints on using center-aligned mode:

- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter counts up or down depending on the value written in the DIR bit

in the TIMx\_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.

- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
  - The direction is not updated if a value greater than the auto-reload value is written in the counter (TIMx\_CNT>TIMx\_ARR). For example, if the counter was counting up, it continues to count up.
  - The direction is updated if 0 or the TIMx\_ARR value is written in the counter but no Update Event UEV is generated.
- The safest way to use center-aligned mode is to generate an update by software (setting the UG bit in the TIMx\_EGR register) just before starting the counter and not to write the counter while it is running.

### 23.3.10 Asymmetric PWM mode

Asymmetric mode allows two center-aligned PWM signals to be generated with a programmable phase shift. While the frequency is determined by the value of the TIMx\_ARR register, the duty cycle and the phase-shift are determined by a pair of TIMx\_CCRx registers. One register controls the PWM during up-counting, the second during down counting, so that PWM is adjusted every half PWM cycle:

- OC1REFC (or OC2REFC) is controlled by TIMx\_CCR1 and TIMx\_CCR2
- OC3REFC (or OC4REFC) is controlled by TIMx\_CCR3 and TIMx\_CCR4

Asymmetric PWM mode can be selected independently on two channels (one OCx output per pair of CCR registers) by writing '1110' (Asymmetric PWM mode 1) or '1111' (Asymmetric PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register.

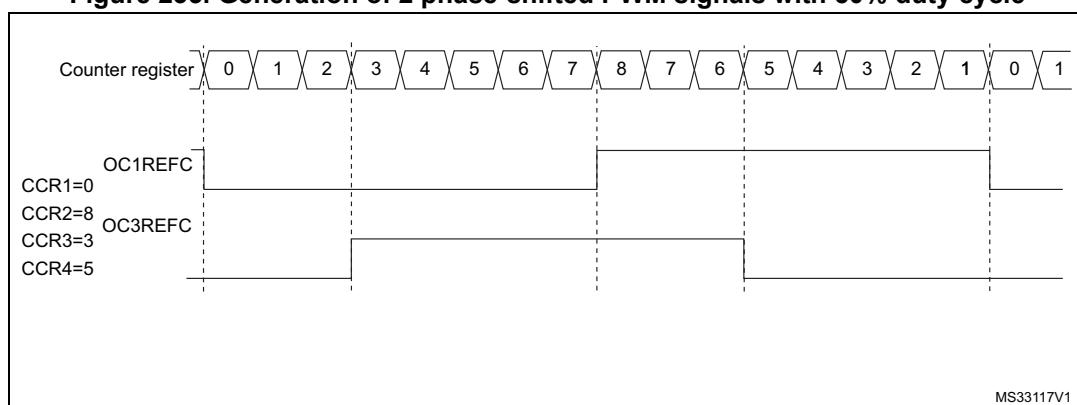
*Note:*

*The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.*

When a given channel is used as asymmetric PWM channel, its secondary channel can also be used. For instance, if an OC1REFC signal is generated on channel 1 (Asymmetric PWM mode 1), it is possible to output either the OC2REF signal on channel 2, or an OC2REFC signal resulting from asymmetric PWM mode 2.

*Figure 235* shows an example of signals that can be generated using Asymmetric PWM mode (channels 1 to 4 are configured in Asymmetric PWM mode 1).

**Figure 235. Generation of 2 phase-shifted PWM signals with 50% duty cycle**



### 23.3.11 Combined PWM mode

Combined PWM mode allows two edge or center-aligned PWM signals to be generated with programmable delay and phase shift between respective pulses. While the frequency is determined by the value of the TIMx\_ARR register, the duty cycle and delay are determined by the two TIMx\_CCRx registers. The resulting signals, OCxREFC, are made of an OR or AND logical combination of two reference PWMs:

- OC1REFC (or OC2REFC) is controlled by TIMx\_CCR1 and TIMx\_CCR2
- OC3REFC (or OC4REFC) is controlled by TIMx\_CCR3 and TIMx\_CCR4

Combined PWM mode can be selected independently on two channels (one OCx output per pair of CCR registers) by writing ‘1100’ (Combined PWM mode 1) or ‘1101’ (Combined PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register.

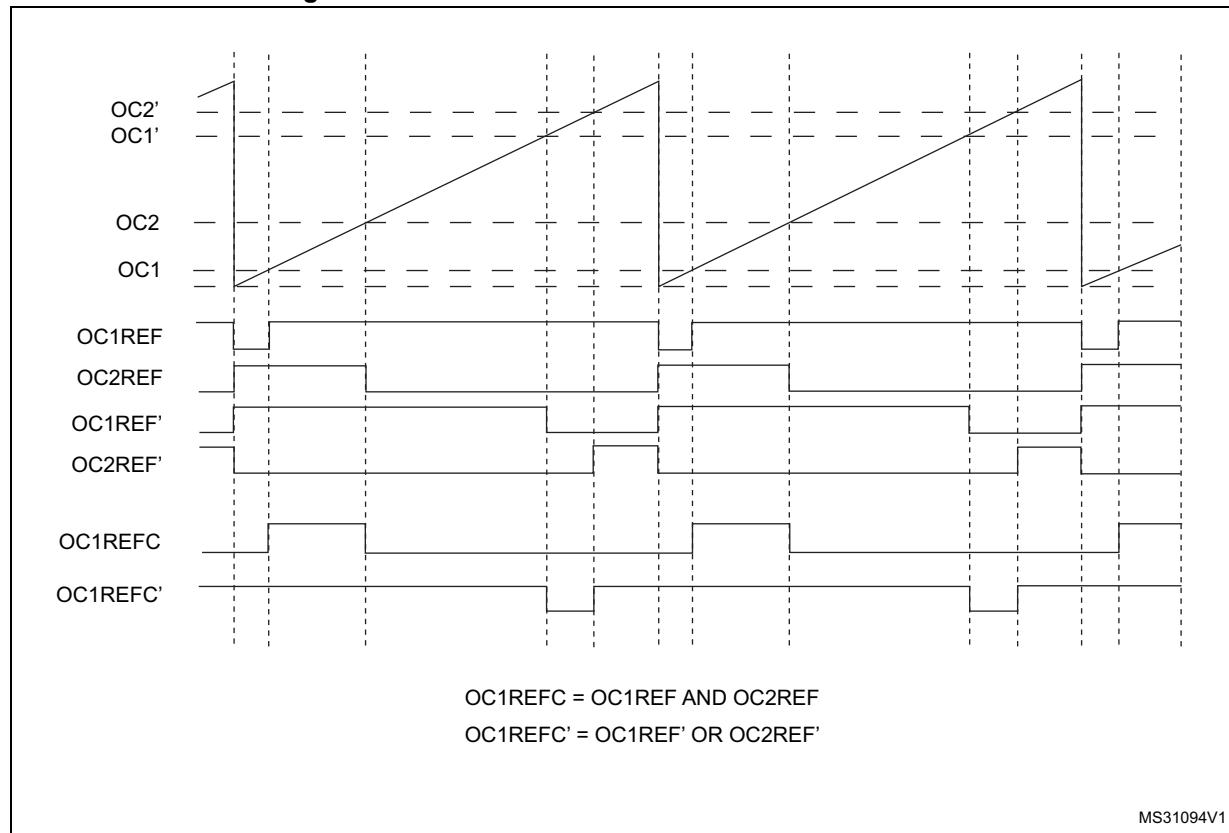
When a given channel is used as combined PWM channel, its secondary channel must be configured in the opposite PWM mode (for instance, one in Combined PWM mode 1 and the other in Combined PWM mode 2).

**Note:** *The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.*

*Figure 236* shows an example of signals that can be generated using Asymmetric PWM mode, obtained with the following configuration:

- Channel 1 is configured in Combined PWM mode 2,
- Channel 2 is configured in PWM mode 1,
- Channel 3 is configured in Combined PWM mode 2,
- Channel 4 is configured in PWM mode 1

Figure 236. Combined PWM mode on channels 1 and 3



### 23.3.12 Clearing the OCxREF signal on an external event

The OCxREF signal of a given channel can be cleared when a high level is applied on the `ocref_clr_int` input (OCxCE enable bit in the corresponding `TIMx_CCMRx` register set to 1). OCxREF remains low until the next transition to the active state, on the following PWM cycle. This function can only be used in Output compare and PWM modes. It does not work in Forced mode.

`OCREF_CLR_INPUT` can be selected between the `OCREF_CLR` input and `ETRF` (`ETR` after the filter) by configuring the `OCCS` bit in the `TIMx_SMCR` register.

The OCxREF signal for a given channel can be reset by applying a high level on the `ETRF` input (OCxCE enable bit set to 1 in the corresponding `TIMx_CCMRx` register). OCxREF remains low until the next transition to the active state, on the following PWM cycle.

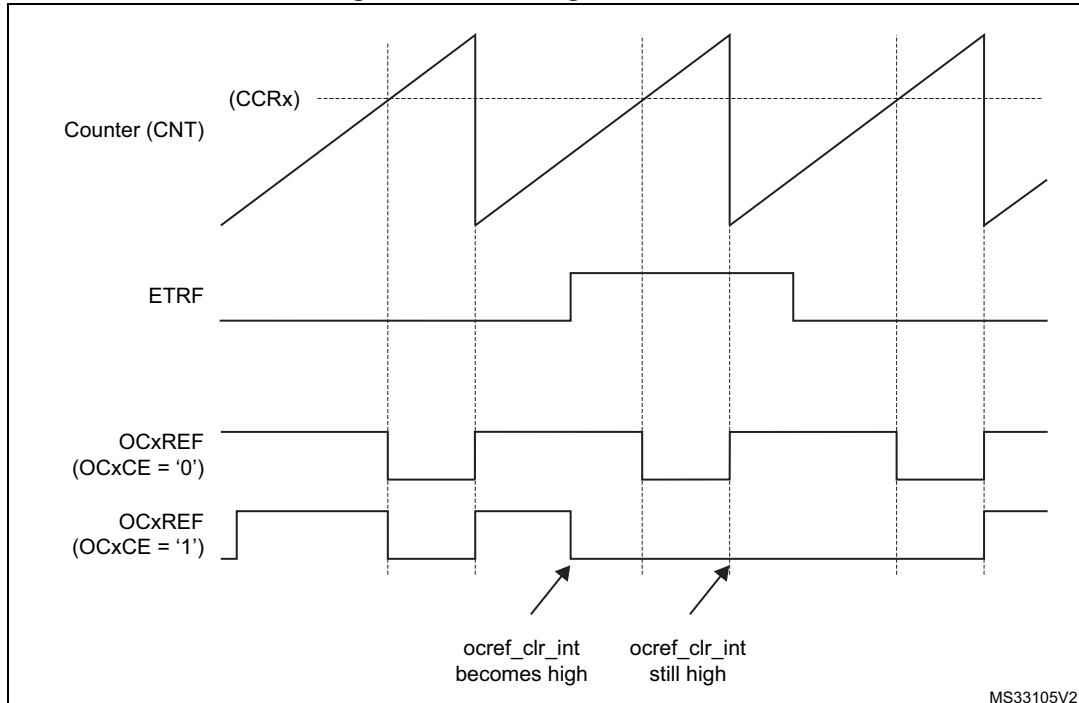
This function can be used only in the output compare and PWM modes. It does not work in forced mode.

For example, the OCxREF signal can be connected to the output of a comparator to be used for current handling. In this case, ETR must be configured as follows:

1. The external trigger prescaler should be kept off: bits ETPS[1:0] in the TIMx\_SMCR register are cleared to 00.
2. The external clock mode 2 must be disabled: bit ECE in the TIM1\_SMCR register is cleared to 0.
3. The external trigger polarity (ETP) and the external trigger filter (ETF) can be configured according to the application's needs.

*Figure 237* 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 237. Clearing TIMx OCxREF**



**Note:** *In case of a PWM with a 100% duty cycle (if  $\text{CCR}_x > \text{ARR}$ ), OCxREF is enabled again at the next counter overflow.*

### 23.3.13 One-pulse mode

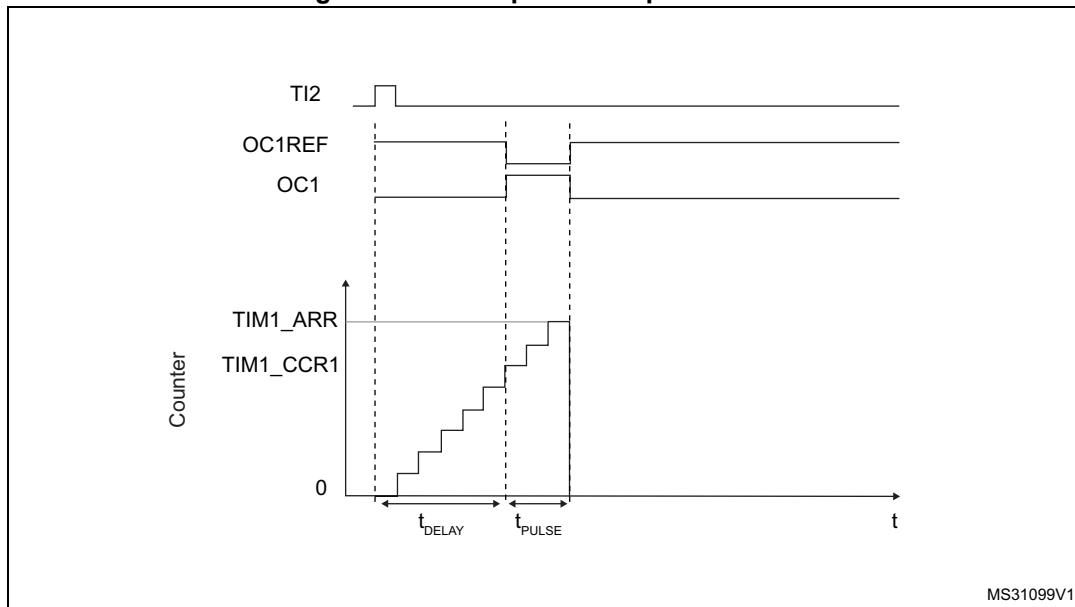
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. One-pulse mode is selected by setting the OPM bit in the TIMx\_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

- CNT<CCR<sub>x</sub> ≤ ARR (in particular, 0<CCR<sub>x</sub>),

**Figure 238. Example of one-pulse mode.**



For example one may want to generate a positive pulse on OC1 with a length of  $t_{PULSE}$  and after a delay of  $t_{DELAY}$  as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

1. Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx\_CCMR1 register.
2. TI2FP2 must detect a rising edge, write CC2P=0 and CC2NP='0' in the TIMx\_CCER register.
3. Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS=110 in the TIMx\_SMCR register.
4. TI2FP2 is used to start the counter by writing SMS to '110 in the TIMx\_SMCR register (trigger mode).

The OPM waveform is defined by writing the compare registers (taking into account the clock frequency and the counter prescaler).

- The  $t_{DELAY}$  is defined by the value written in the TIMx\_CCR1 register.
- The  $t_{PULSE}$  is defined by the difference between the auto-reload value and the compare value (TIMx\_ARR - TIMx\_CCR1).
- Let's say one want to build a waveform with a transition from '0 to '1 when a compare match occurs and a transition from '1 to '0 when the counter reaches the auto-reload value. To do this PWM mode 2 must be enabled by writing OC1M=111 in the TIMx\_CCMR1 register. Optionally the preload registers can be enabled by writing OC1PE=1 in the TIMx\_CCMR1 register and ARPE in the TIMx\_CR1 register. In this case one has to write the compare value in the TIMx\_CCR1 register, the auto-reload value in the TIMx\_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0 in this example.

In our example, the DIR and CMS bits in the TIMx\_CR1 register should be low.

Since only 1 pulse (Single mode) is needed, a 1 must be written in the OPM bit in the TIMx\_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0). When OPM bit in the TIMx\_CR1 register is set to '0', so the Repetitive Mode is selected.

#### **Particular case: OCx fast enable:**

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay  $t_{DELAY}$  min we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set in the TIMx\_CCMRx register. Then OCxRef (and OCx) 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.

### **23.3.14 Retriggerable one pulse mode**

This mode allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length, but with the following differences with Non-retriggerable one pulse mode described in [Section 23.3.13](#):

- The pulse starts as soon as the trigger occurs (no programmable delay)
- The pulse is extended if a new trigger occurs before the previous one is completed

The timer must be in Slave mode, with the bits SMS[3:0] = '1000' (Combined Reset + trigger mode) in the TIMx\_SMCR register, and the OCxM[3:0] bits set to '1000' or '1001' for Retriggerable OPM mode 1 or 2.

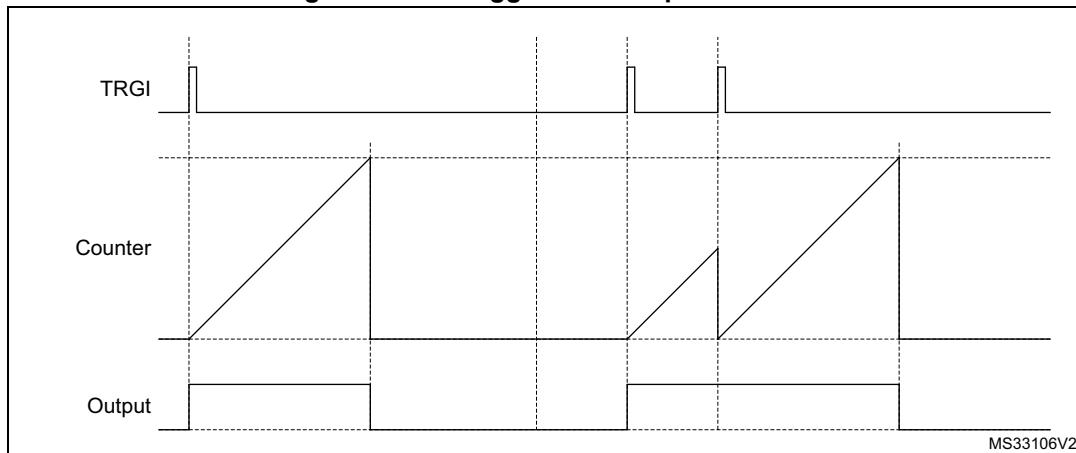
If the timer is configured in Up-counting mode, the corresponding CCRx must be set to 0 (the ARR register sets the pulse length). If the timer is configured in Down-counting mode CCRx must be above or equal to ARR.

*Note:* In retriggerable one pulse mode, the CCxIF flag is not significant.

*The OCxM[3:0] and SMS[3:0] bit fields are split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.*

*This mode must not be used with center-aligned PWM modes. It is mandatory to have CMS[1:0] = 00 in TIMx\_CR1.*

Figure 239. Retriggerable one-pulse mode



### 23.3.15 Encoder interface mode

To select Encoder Interface mode write SMS='001 in the TIMx\_SMCR register if the counter is counting on TI2 edges only, SMS=010 if it is counting on TI1 edges only and SMS=011 if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx\_CCER register. CC1NP and CC2NP must be kept cleared. When needed, the input filter can be programmed 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 153](#). The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx\_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx\_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx\_ARR register (0 to ARR or ARR down to 0 depending on the direction). So the TIMx\_ARR must be configured before starting. In the same way, the capture, compare, prescaler, trigger output features continue to work as normal.

In this mode, the counter is modified automatically following the speed and the direction of the-quadrature encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 do not switch at the same time.

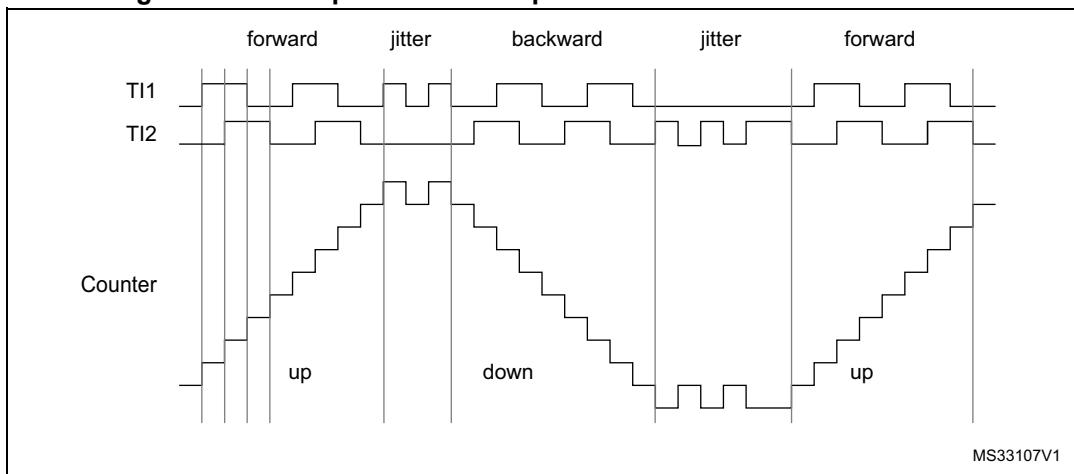
**Table 153. Counting direction versus encoder signals**

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

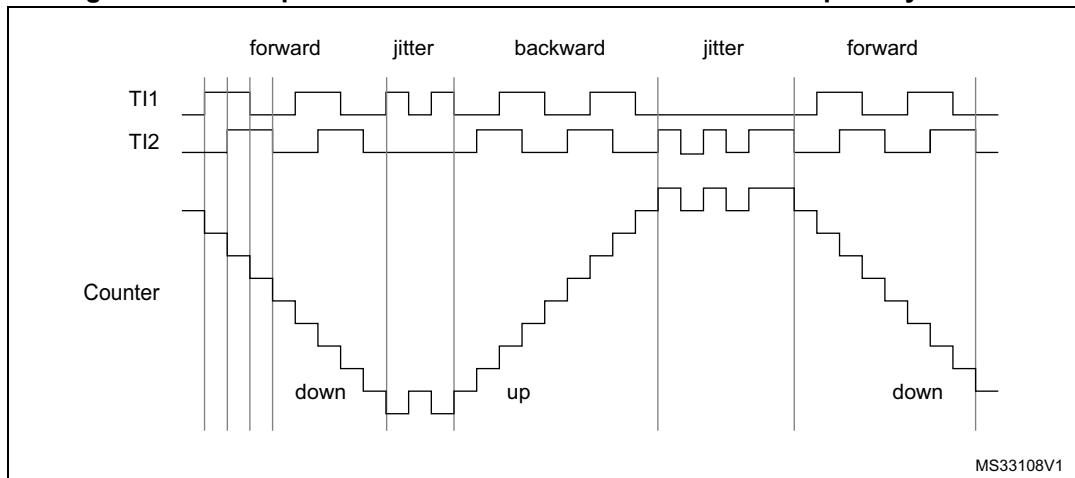
An external incremental encoder can be connected directly to the MCU without external interface logic. However, comparators are normally be used to convert the encoder's differential outputs to digital signals. This greatly increases noise immunity. The third encoder output which indicate the mechanical zero position, may be connected to an external interrupt input and trigger a counter reset.

*Figure 240* 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\_CCMR1 register, TI2FP2 mapped on TI2)
- CC1P and CC1NP = '0' (TIMx\_CCER register, TI1FP1 noninverted, TI1FP1=TI1)
- CC2P and CC2NP = '0' (TIMx\_CCER register, TI2FP2 noninverted, TI2FP2=TI2)
- SMS= 011 (TIMx\_SMCR register, both inputs are active on both rising and falling edges)
- CEN= 1 (TIMx\_CR1 register, Counter is enabled)

**Figure 240. Example of counter operation in encoder interface mode**

*Figure 241* gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P=1).

**Figure 241. Example of encoder interface mode with TI1FP1 polarity inverted**

The timer, when configured in Encoder Interface mode provides information on the sensor's current position. Dynamic information can be obtained (speed, acceleration, deceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. This can be done by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer). when available, it is also possible to read its value through a DMA request generated by a Real-Time clock.

### 23.3.16 UIF bit remapping

The IUFREMAP bit in the TIMx\_CR1 register forces a continuous copy of the update interrupt flag (UIF) into bit 31 of the timer counter register's bit 31 (TIMxCNT[31]). This permits to atomically read both the counter value and a potential roll-over condition signaled by the UIFCPY flag. It eases the calculation of angular speed by avoiding race conditions caused, for instance, by a processing shared between a background task (counter reading) and an interrupt (update interrupt).

There is no latency between the UIF and UIFCPY flag assertions.

In 32-bit timer implementations, when the IUFREMAP bit is set, bit 31 of the counter is overwritten by the UIFCPY flag upon read access (the counter's most significant bit is only accessible in write mode).

### 23.3.17 Timer input XOR function

The TI1S bit in the TIM1xx\_CR2 register, allows the input filter of channel 1 to be connected to the output of a XOR gate, combining the three input pins TIMx\_CH1 to TIMx\_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture.

An example of this feature used to interface Hall sensors is given in [Section 22.3.24: Interfacing with Hall sensors on page 685](#).

### 23.3.18 Timers and external trigger synchronization

The TIMx Timers can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

#### Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

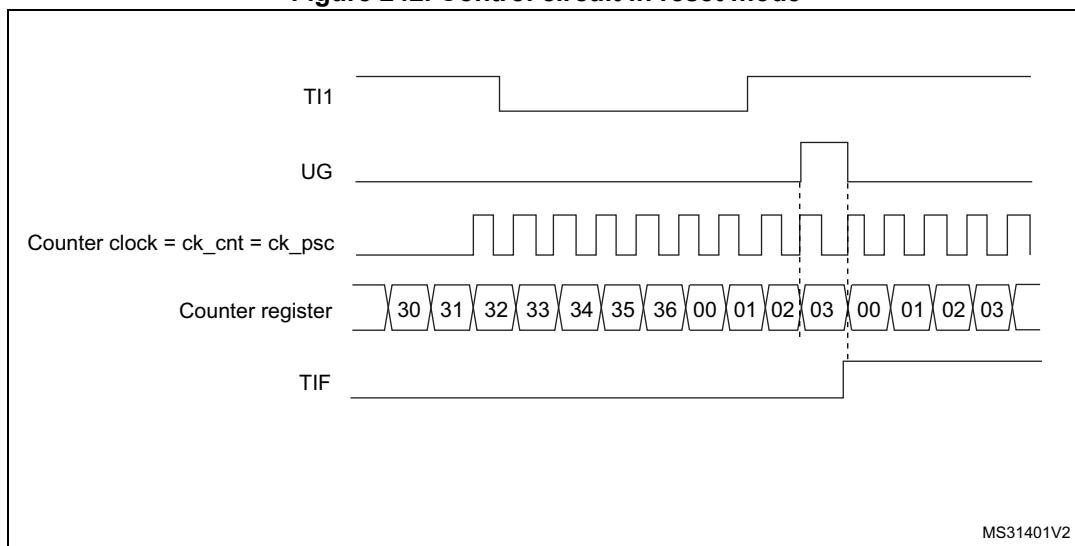
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

1. Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx\_CCMR1 register. Write CC1P=0 and CC1NP=0 in TIMx\_CCER register to validate the polarity (and detect rising edges only).
2. Configure the timer in reset mode by writing SMS=100 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
3. Start the counter by writing CEN=1 in the TIMx\_CR1 register.

The counter starts counting on the internal clock, then behaves normally until TI1 rising edge. When TI1 rises, the counter is cleared and restarts from 0. In the meantime, the trigger flag is set (TIF bit in the TIMx\_SR register) and an interrupt request, or a DMA request can be sent if enabled (depending on the TIE and TDE bits in TIMx\_DIER register).

The following figure shows this behavior when the auto-reload register TIMx\_ARR=0x36. The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

**Figure 242. Control circuit in reset mode**



#### Slave mode: Gated mode

The counter can be enabled depending on the level of a selected input.

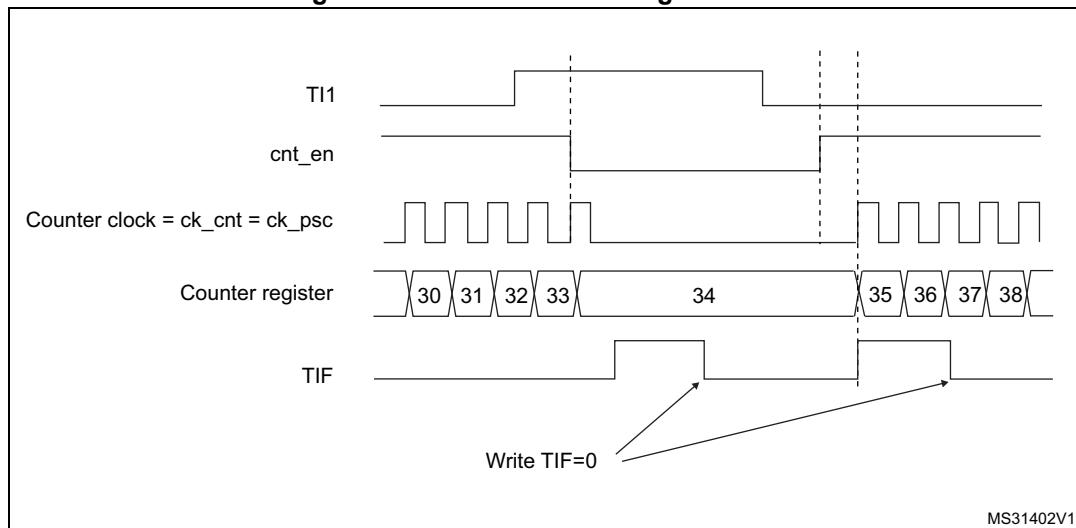
In the following example, the upcounter counts only when TI1 input is low:

1. Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S=01 in TIMx\_CCMR1 register. Write CC1P=1 and CC1NP=0 in TIMx\_CCER register to validate the polarity (and detect low level only).
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 243. Control circuit in gated mode**



1. The configuration “CCxP=CCxNP=1” (detection of both rising and falling edges) does not have any effect in gated mode because gated mode acts on a level and not on an edge.

**Note:**

*The configuration “CCxP=CCxNP=1” (detection of both rising and falling edges) does not have any effect in gated mode because gated mode acts on a level and not on an edge.*

### Slave mode: Trigger mode

The counter can start in response to an event on a selected input.

In the following example, the upcounter starts in response to a rising edge on TI2 input:

1. Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we do not need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. CC2S bits are selecting the input capture source only, CC2S=01 in TIMx\_CCMR1 register. Write

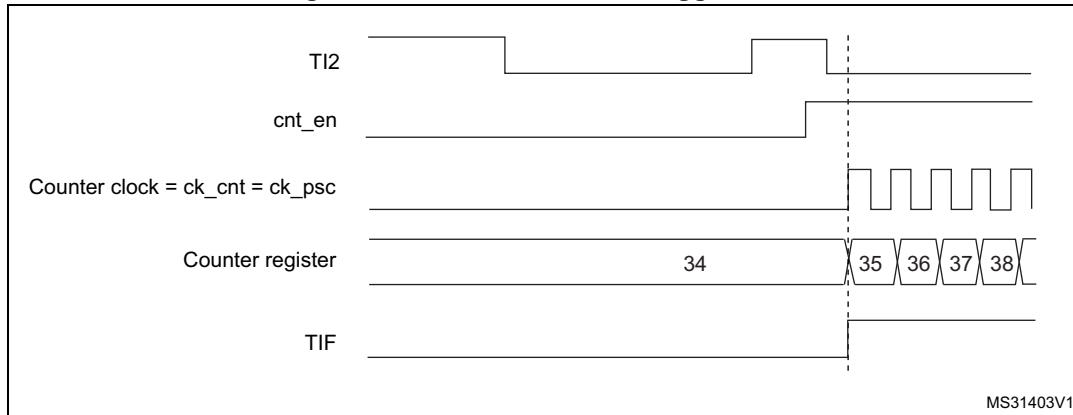
CC2P=1 and CC2NP=0 in TIMx\_CCER register to validate the polarity (and detect low level only).

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 244. Control circuit in trigger mode**



### Slave mode: Combined reset + trigger mode

In this case, a rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers, and starts the counter.

This mode is used for one-pulse mode.

### Slave mode: External Clock mode 2 + trigger mode

The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input when operating in reset mode, gated mode or trigger mode. It is recommended not to select ETR as TRGI through the TS bits of TIMx\_SMCR register.

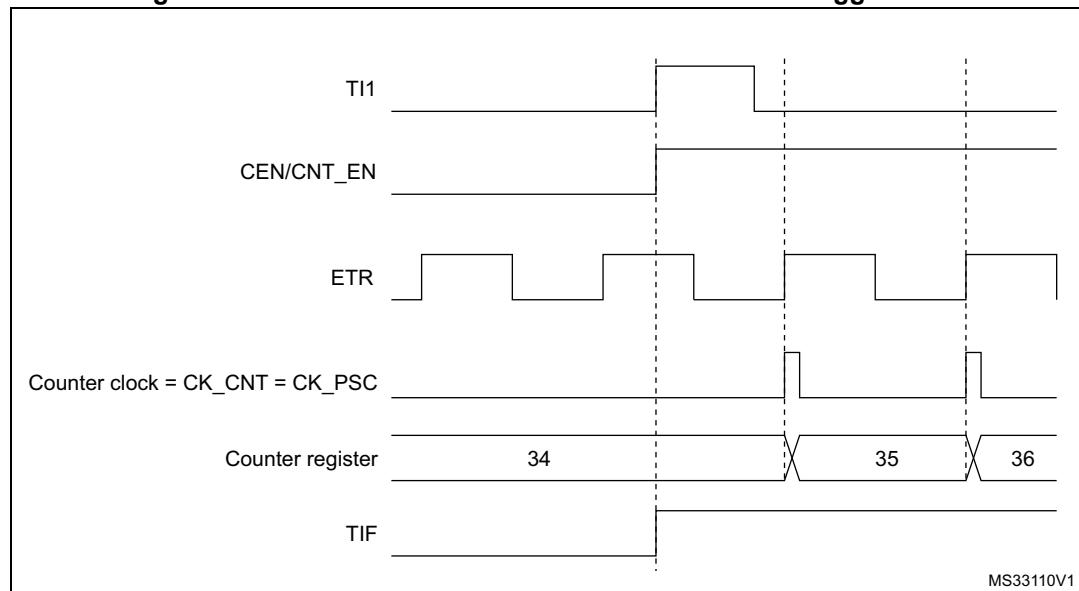
In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

1. Configure the external trigger input circuit by programming the TIMx\_SMCR register as follows:
  - ETF = 0000: no filter
  - ETPS=00: prescaler disabled
  - ETP=0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.
2. Configure the channel 1 as follows, to detect rising edges on TI:
  - IC1F=0000: no filter.
  - The capture prescaler is not used for triggering and does not need to be configured.
  - CC1S=01in TIMx\_CCMR1 register to select only the input capture source
  - CC1P=0 and CC1NP=0 in TIMx\_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.

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.

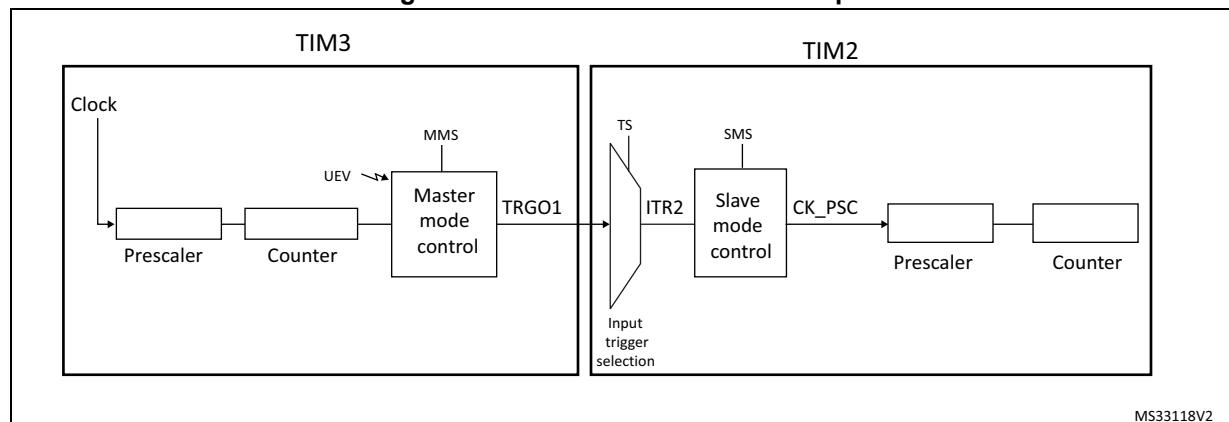
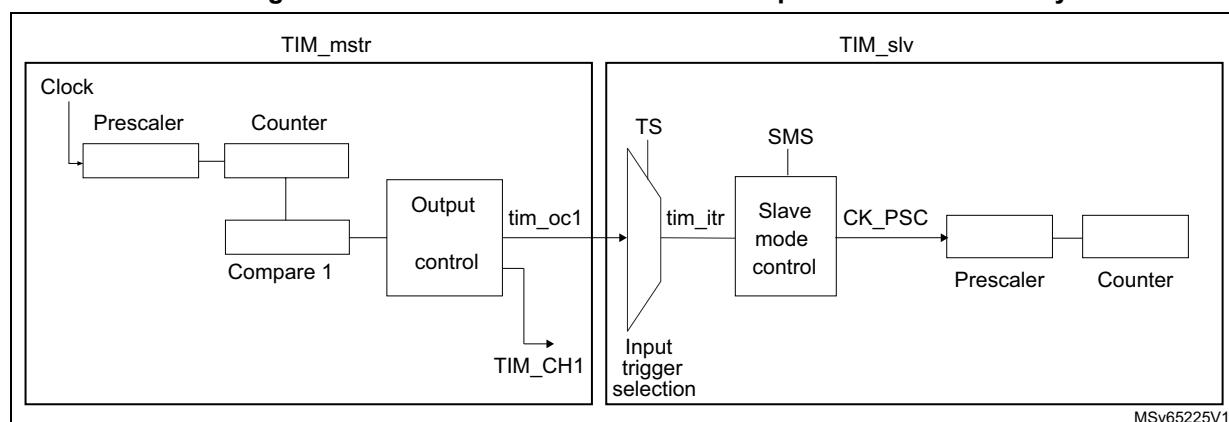
**Figure 245. Control circuit in external clock mode 2 + trigger mode**



### 23.3.19 Timer synchronization

The TIMx timers are linked together internally for timer synchronization or chaining. When one Timer is configured in Master Mode, it can reset, start, stop or clock the counter of another Timer configured in Slave Mode.

[Figure 246: Master/Slave timer example](#) and [Figure 247: Master/slave connection example with 1 channel only timers](#) present an overview of the trigger selection and the master mode selection blocks.

**Figure 246. Master/Slave timer example****Figure 247. Master/slave connection example with 1 channel only timers**

**Note:** The timers with one channel only (see [Figure 247](#)) do not feature a master mode. However, the OC1 output signal can be used to trigger some other timers (including timers described in other sections of this document). Check the “*TIMx internal trigger connection*” table of any **TIMx\_SMCR** register on the device to identify which timers can be targeted as slave. The OC1 signal pulse width must be programmed to be at least 2 clock cycles of the destination timer, to make sure the slave timer detects the trigger. For instance, if the destination’s timer **CK\_INT** clock is 4 times slower than the source timer, the OC1 pulse width must be 8 clock cycles.

### Using one timer as prescaler for another timer

For example, TIM3 can be configured to act as a prescaler for TIM. Refer to [Figure 246](#). To do this:

1. Configure TIM3 in master mode so that it outputs a periodic trigger signal on each update event UEV. If MMS=010 is written in the TIM3\_CR2 register, a rising edge is output on TRGO each time an update event is generated.
2. To connect the TRGO output of TIM3 to TIM, TIM must be configured in slave mode using ITR2 as internal trigger. This is selected through the TS bits in the TIM\_SMCR register (writing TS=010).
3. Then the slave mode controller must be put in external clock mode 1 (write SMS=111 in the TIM\_SMCR register). This causes TIM to be clocked by the rising edge of the periodic TIM3 trigger signal (which correspond to the TIM3 counter overflow).
4. Finally both timers must be enabled by setting their respective CEN bits (TIMx\_CR1 register).

**Note:** *If OCx is selected on TIM3 as the trigger output (MMS=1xx), its rising edge is used to clock the counter of TIM.*

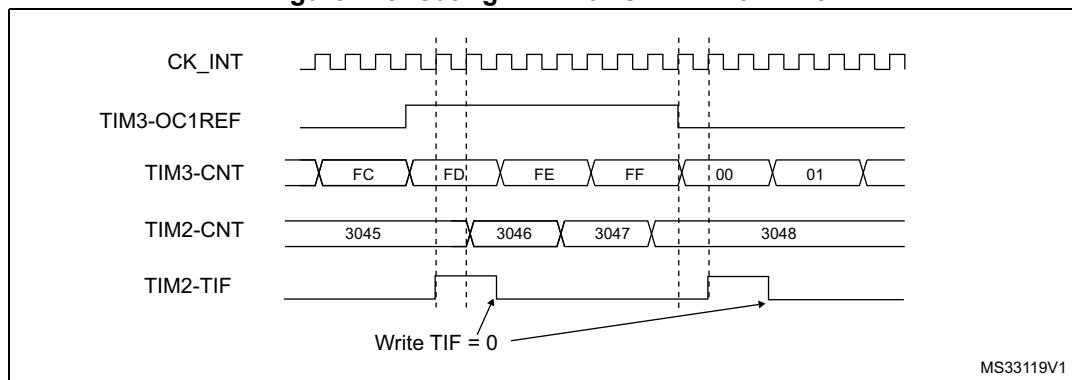
### Using one timer to enable another timer

In this example, we control the enable of TIM with the output compare 1 of Timer 3. Refer to [Figure 246](#) for connections. TIM counts on the divided internal clock only when OC1REF of TIM3 is high. Both counter clock frequencies are divided by 3 by the prescaler compared to CK\_INT ( $f_{CK\_CNT} = f_{CK\_INT}/3$ ).

1. Configure TIM3 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM3\_CR2 register).
2. Configure the TIM3 OC1REF waveform (TIM3\_CCMR1 register).
3. Configure TIM to get the input trigger from TIM3 (TS=010 in the TIM\_SMCR register).
4. Configure TIM in gated mode (SMS=101 in TIM\_SMCR register).
5. Enable TIM by writing '1 in the CEN bit (TIM\_CR1 register).
6. Start TIM3 by writing '1 in the CEN bit (TIM3\_CR1 register).

**Note:** *The counter clock is not synchronized with counter 1, this mode only affects the TIM counter enable signal.*

**Figure 248. Gating TIM with OC1REF of TIM3**

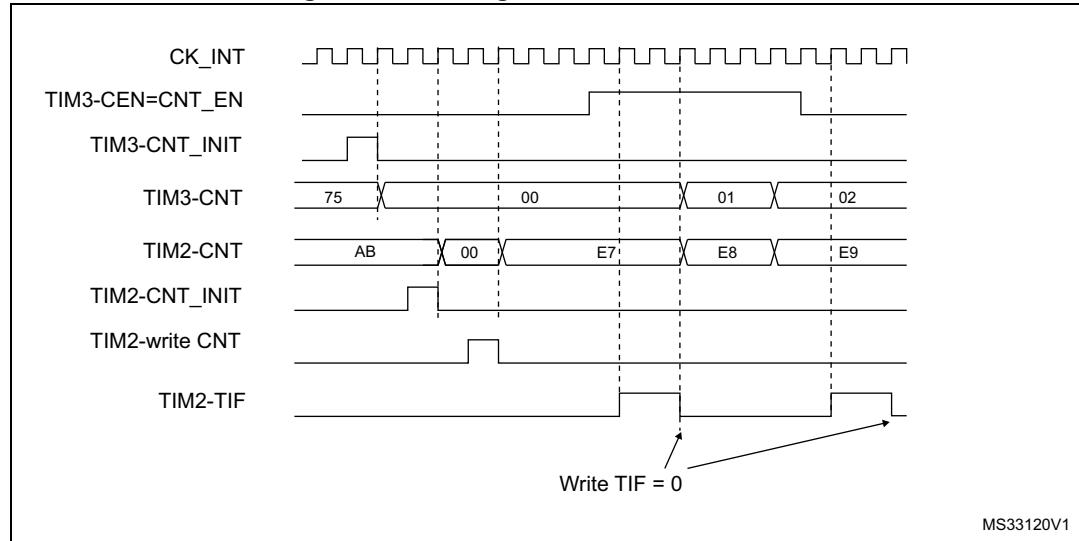


In the example in [Figure 248](#), the TIM counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting TIM3. Then any value can be written in the timer counters. The timers can easily be reset by software using the UG bit in the TIMx\_EGR registers.

In the next example (refer to [Figure 249](#)), we synchronize TIM3 and TIM. TIM3 is the master and starts from 0. TIM is the slave and starts from 0xE7. The prescaler ratio is the same for both timers. TIM stops when TIM3 is disabled by writing '0 to the CEN bit in the TIM3\_CR1 register:

1. Configure TIM3 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM3\_CR2 register).
2. Configure the TIM3 OC1REF waveform (TIM3\_CCMR1 register).
3. Configure TIM to get the input trigger from TIM3 (TS=010 in the TIM\_SMCR register).
4. Configure TIM in gated mode (SMS=101 in TIM\_SMCR register).
5. Reset TIM3 by writing '1 in UG bit (TIM3\_EGR register).
6. Reset TIM by writing '1 in UG bit (TIM\_EGR register).
7. Initialize TIM to 0xE7 by writing '0xE7' in the TIM counter (TIM\_CNTL).
8. Enable TIM by writing '1 in the CEN bit (TIM\_CR1 register).
9. Start TIM3 by writing '1 in the CEN bit (TIM3\_CR1 register).
10. Stop TIM3 by writing '0 in the CEN bit (TIM3\_CR1 register).

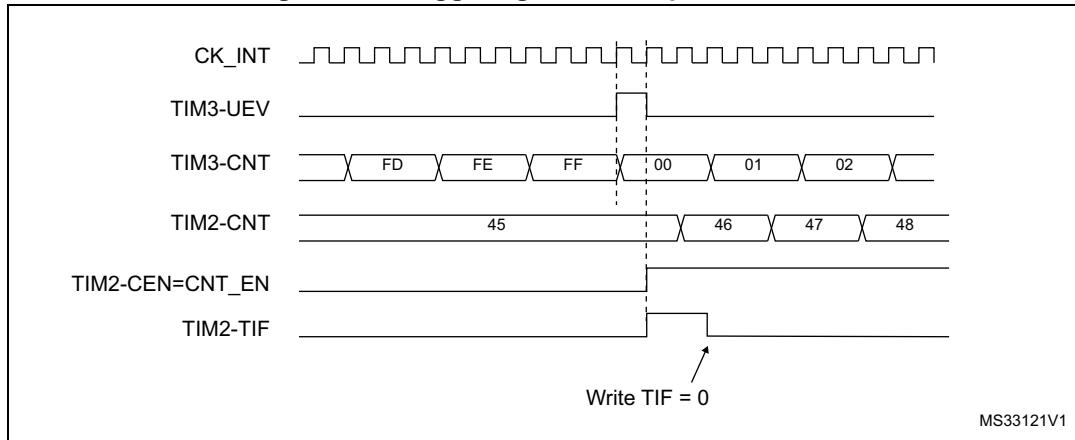
**Figure 249. Gating TIM with Enable of TIM3**



### Using one timer to start another timer

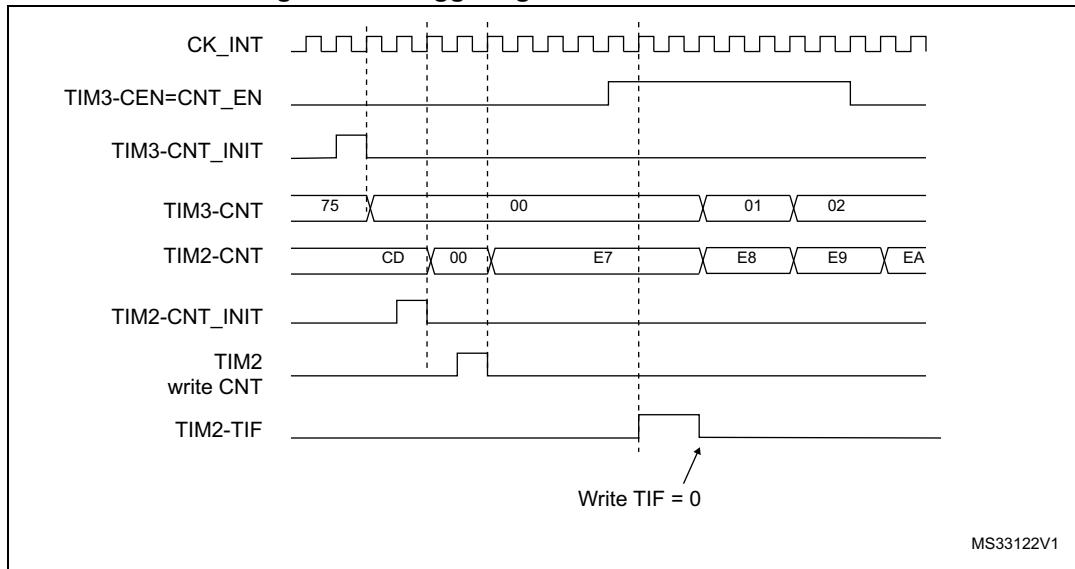
In this example, we set the enable of Timer with the update event of Timer 3. Refer to [Figure 246](#) for connections. Timer starts counting from its current value (which can be non-zero) on the divided internal clock as soon as the update event is generated by Timer 1. When Timer receives the trigger signal its CEN bit is automatically set and the counter counts until we write '0 to the CEN bit in the TIM\_CR1 register. Both counter clock frequencies are divided by 3 by the prescaler compared to CK\_INT ( $f_{CK\_CNT} = f_{CK\_INT}/3$ ).

1. Configure TIM3 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM3\_CR2 register).
2. Configure the TIM3 period (TIM3\_ARR registers).
3. Configure TIM to get the input trigger from TIM3 (TS=010 in the TIM\_SMCR register).
4. Configure TIM in trigger mode (SMS=110 in TIM\_SMCR register).
5. Start TIM3 by writing '1 in the CEN bit (TIM3\_CR1 register).

**Figure 250. Triggering TIM with update of TIM3**

As in the previous example, both counters can be initialized before starting counting.

[Figure 251](#) shows the behavior with the same configuration as in [Figure 250](#) but in trigger mode instead of gated mode (SMS=110 in the TIM\_SMCR register).

**Figure 251. Triggering TIM with Enable of TIM3**

### Starting 2 timers synchronously in response to an external trigger

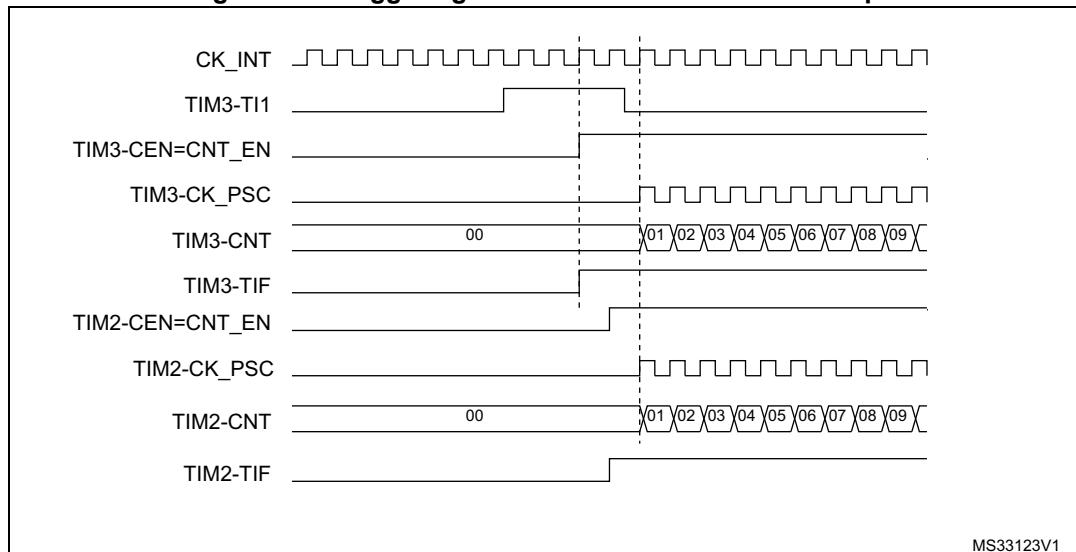
In this example, we set the enable of TIM3 when its TI1 input rises, and the enable of TIM2 with the enable of TIM3. Refer to [Figure 246](#) for connections. To ensure the counters are aligned, TIM3 must be configured in Master/Slave mode (slave with respect to TI1, master with respect to TIM2):

1. Configure TIM3 master mode to send its Enable as trigger output (MMS=001 in the TIM3\_CR2 register).
2. Configure TIM3 slave mode to get the input trigger from TI1 (TS=100 in the TIM3\_SMCR register).
3. Configure TIM3 in trigger mode (SMS=110 in the TIM3\_SMCR register).
4. Configure the TIM3 in Master/Slave mode by writing MSM=1 (TIM3\_SMCR register).
5. Configure TIM2 to get the input trigger from TIM3 (TS=000 in the TIM2\_SMCR register).
6. Configure TIM2 in trigger mode (SMS=110 in the TIM2\_SMCR register).

When a rising edge occurs on TI1 (TIM3), both counters starts counting synchronously on the internal clock and both TIF flags are set.

**Note:** *In this example both timers are initialized before starting (by setting their respective UG bits). Both counters starts from 0, but an offset can easily be inserted between them by writing any of the counter registers (TIMx\_CNT). One can see that the master/slave mode insert a delay between CNT\_EN and CK\_PSC on TIM3.*

**Figure 252. Triggering TIM3 and TIM2 with TIM3 TI1 input**



**Note:** *The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.*

### 23.3.20 DMA burst mode

The TIMx timers have the capability to generate multiple DMA requests upon a single event. The main purpose is to be able to re-program part of the timer multiple times without software overhead, but it can also be used to read several registers in a row, at regular intervals.

The DMA controller destination is unique and must point to the virtual register TIMx\_DMAR. On a given timer event, the timer launches a sequence of DMA requests (burst). Each write into the TIMx\_DMAR register is actually redirected to one of the timer registers.

The DBL[4:0] bits in the TIMx\_DCR register set the DMA burst length. The timer recognizes a burst transfer when a read or a write access is done to the TIMx\_DMAR address), i.e. the number of transfers (either in half-words or in bytes).

The DBA[4:0] bits in the TIMx\_DCR registers define the DMA base address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register:

Example:

00000: TIMx\_CR1

00001: TIMx\_CR2

00010: TIMx\_SMCR

As an example, the timer DMA burst feature is used to update the contents of the CCRx registers ( $x = 2, 3, 4$ ) upon an update event, with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
  - DMA channel peripheral address is the DMAR register address
  - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
  - Number of data to transfer = 3 (See note below).
  - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:  
DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

This example is for the case where every CCRx register has to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let's take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.

*Note:* A null value can be written to the reserved registers.

### 23.3.21 Debug mode

When the microcontroller enters debug mode (Cortex®-M7 core - halted), the TIMx counter either continues to work normally or stops, depending on DBG\_TIMx\_STOP configuration bit in DBGMCU module. For more details, refer to [Section 40.16.2: Debug support for timers, watchdog, bxCAN and I<sup>2</sup>C](#).

## 23.4 TIM2/TIM3/TIM4/TIM5 registers

Refer to [Section 1.2](#) for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

### 23.4.1 TIMx control register 1 (TIMx\_CR1)(x = 2 to 5)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

- 0: No remapping. UIF status bit is not copied to TIMx\_CNT register bit 31.
- 1: Remapping enabled. UIF status bit is copied to TIMx\_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

- 00:  $t_{DTS} = t_{CK\_INT}$
- 01:  $t_{DTS} = 2 \times t_{CK\_INT}$
- 10:  $t_{DTS} = 4 \times t_{CK\_INT}$
- 11: Reserved

Bit 7 **ARPE**: Auto-reload preload enable

- 0: TIMx\_ARR register is not buffered
- 1: TIMx\_ARR register is buffered

Bits 6:5 **CMS[1:0]**: Center-aligned mode selection

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting down.

10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting up.

11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set both when the counter is counting up or down.

*Note:* It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

Bit 4 **DIR**: Direction

- 0: Counter used as upcounter
- 1: Counter used as downcounter

*Note:* This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

**Bit 3 OPM:** One-pulse mode

- 0: Counter is not stopped at update event  
 1: Counter stops counting at the next update event (clearing the bit CEN)

**Bit 2 URS:** Update request source

This bit is set and cleared by software to select the UEV event sources.

- 0: Any of the following events generate an update interrupt or DMA request if enabled.  
 These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

**Bit 1 UDIS:** Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

**Bit 0 CEN:** Counter enable

- 0: Counter disabled  
 1: Counter enabled

*Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.*

CEN is cleared automatically in one-pulse mode, when an update event occurs.

### 23.4.2 TIMx control register 2 (TIMx\_CR2)(x = 2 to 5)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TI1S	MMS[2:0]			CCDS	Res.	Res.	Res.							

Bits 15:8 Reserved, must be kept at reset value.

**Bit 7 TI1S:** TI1 selection

- 0: The TIMx\_CH1 pin is connected to TI1 input  
 1: The TIMx\_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)  
 See also [Section 22.3.24: Interfacing with Hall sensors on page 685](#)

**Bits 6:4 MMS[2:0]: Master mode selection**

These bits permit 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 AND 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** - OC1REFC signal is used as trigger output (TRGO)

101: **Compare** - OC2REFC signal is used as trigger output (TRGO)

110: **Compare** - OC3REFC signal is used as trigger output (TRGO)

111: **Compare** - OC4REFC signal is used as trigger output (TRGO)

*Note: The clock of the slave timer or ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

**Bit 3 CCDS: Capture/compare DMA selection**

0: CCx DMA request sent when CCx event occurs

1: CCx DMA requests sent when update event occurs

Bits 2:0 Reserved, must be kept at reset value.

### 23.4.3 TIMx slave mode control register (TIMx\_SMCR)(x = 2 to 5)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SMS[3]
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			OCCS	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

Bit 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:* Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).

*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).*

*If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.*

Bits 13:12 **ETPS[1:0]**: External trigger prescaler

External trigger signal ETRP frequency must be at most 1/4 of CK\_INT frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

00: Prescaler OFF

01: ETRP frequency divided by 2

10: ETRP frequency divided by 4

11: ETRP frequency divided by 8

**Bits 11:8 ETF[3:0]: External trigger filter**

This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at  $f_{DTS}$
- 0001:  $f_{SAMPLING} = f_{CK\_INT}$ , N=2
- 0010:  $f_{SAMPLING} = f_{CK\_INT}$ , N=4
- 0011:  $f_{SAMPLING} = f_{CK\_INT}$ , N=8
- 0100:  $f_{SAMPLING} = f_{DTS}/2$ , N=6
- 0101:  $f_{SAMPLING} = f_{DTS}/2$ , N=8
- 0110:  $f_{SAMPLING} = f_{DTS}/4$ , N=6
- 0111:  $f_{SAMPLING} = f_{DTS}/4$ , N=8
- 1000:  $f_{SAMPLING} = f_{DTS}/8$ , N=6
- 1001:  $f_{SAMPLING} = f_{DTS}/8$ , N=8
- 1010:  $f_{SAMPLING} = f_{DTS}/16$ , N=5
- 1011:  $f_{SAMPLING} = f_{DTS}/16$ , N=6
- 1100:  $f_{SAMPLING} = f_{DTS}/16$ , N=8
- 1101:  $f_{SAMPLING} = f_{DTS}/32$ , N=5
- 1110:  $f_{SAMPLING} = f_{DTS}/32$ , N=6
- 1111:  $f_{SAMPLING} = f_{DTS}/32$ , N=8

**Bit 7 MSM: Master/Slave mode**

0: No action

1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 **TS**: Trigger selection

This bit-field selects the trigger input to be used to synchronize the counter.

000: Internal Trigger 0 (ITR0).

001: Internal Trigger 1 (ITR1).

010: Internal Trigger 2 (ITR2).

011: Internal Trigger 3 (ITR3).

100: TI1 Edge Detector (TI1F\_ED)

101: Filtered Timer Input 1 (TI1FP1)

110: Filtered Timer Input 2 (TI2FP2)

111: External Trigger input (ETRF)

See [Table 154: TIMx internal trigger connection on page 783](#) for more details on ITRx meaning for each Timer.

*Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

Bit 3 **OCCS**: OCREF clear selection

This bit is used to select the OCREF clear source

0: OCREF\_CLR\_INT is connected to the OCREF\_CLR input

1: OCREF\_CLR\_INT is connected to ETRF

Bits 16, 2, 1, 0 **SMS[3:0]**: Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

0000: Slave mode disabled - if CEN = '1 then the prescaler is clocked directly by the internal clock.

0001: Encoder mode 1 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

0010: Encoder mode 2 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

0011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

0100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

0101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

0110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

0111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

1000: Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter.

*Note: The gated mode must not be used if TI1F\_ED is selected as the trigger input (TS=100). Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.*

*Note: The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.*

Table 154. TIMx internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1	TIM8/ETH_PTP/OTG_FS_SOF/OTG_HS_SOF <sup>(1)</sup>	TIM3	TIM4
TIM3	TIM1	TIM2	TIM5	TIM4
TIM4	TIM1	TIM2	TIM3	TIM8
TIM5	TIM2	TIM3	TIM4	TIM8

1. Depends on the bit ITR1\_RMP in TIM2\_OR1 register.

### 23.4.4 TIMx DMA/Interrupt enable register (TIMx\_DIER)(x = 2 to 5)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res.	CC4IE	CC3IE	CC2IE	CC1IE	UIE

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TDE**: Trigger DMA request enable

- 0: Trigger DMA request disabled.
- 1: Trigger DMA request enabled.

Bit 13 Reserved, must be kept at reset value.

Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable

- 0: CC4 DMA request disabled.
- 1: CC4 DMA request enabled.

Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable

- 0: CC3 DMA request disabled.
- 1: CC3 DMA request enabled.

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable

- 0: CC2 DMA request disabled.
- 1: CC2 DMA request enabled.

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable

- 0: CC1 DMA request disabled.
- 1: CC1 DMA request enabled.

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled.
- 1: Update DMA request enabled.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **TIE**: Trigger interrupt enable

- 0: Trigger interrupt disabled.
- 1: Trigger interrupt enabled.

Bit 5 Reserved, must be kept at reset value.

- Bit 4 **CC4IE**: Capture/Compare 4 interrupt enable  
0: CC4 interrupt disabled.  
1: CC4 interrupt enabled.
- Bit 3 **CC3IE**: Capture/Compare 3 interrupt enable  
0: CC3 interrupt disabled.  
1: CC3 interrupt enabled.
- Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable  
0: CC2 interrupt disabled.  
1: CC2 interrupt enabled.
- Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable  
0: CC1 interrupt disabled.  
1: CC1 interrupt enabled.
- Bit 0 **UIE**: Update interrupt enable  
0: Update interrupt disabled.  
1: Update interrupt enabled.

### 23.4.5 TIMx status register (TIMx\_SR)(x = 2 to 5)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	Res.	TIF	Res.	CC4IF	CC3IF	CC2IF	CC1IF	UIF

Bits 15:13 Reserved, must be kept at reset value.

Bit 12 **CC4OF**: Capture/Compare 4 overcapture flag  
refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 overcapture flag  
refer to CC1OF description

Bit 10 **CC2OF**: Capture/compare 2 overcapture flag  
refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag  
This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.  
0: No overcapture has been detected.  
1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on the TRG 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  
This flag is set by hardware. It is cleared by software (input capture or output compare mode) or by reading the TIMx\_CCR1 register (input capture mode only).  
0: No compare match / No input capture occurred  
1: A compare match or an input capture occurred  
**If channel CC1 is configured as output:** this flag is set when the content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register. When the content of TIMx\_CCR1 is greater than the content of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in up-counting and up/down-counting modes) or underflow (in down-counting mode). There are 3 possible options for flag setting in center-aligned mode, refer to the CMS bits in the TIMx\_CR1 register for the full description.  
**If channel CC1 is configured as input:** this bit is set when counter value has been captured in TIMx\_CCR1 register (an edge has been detected on IC1, as per the edge sensitivity defined with the CC1P and CC1NP bits setting, in TIMx\_CCER).
- Bit 0 **UIF**: Update interrupt flag  
This bit is set by hardware on an update event. It is cleared by software.  
0: No update occurred  
1: Update interrupt pending. This bit is set by hardware when the registers are updated:  
At overflow or underflow (for TIM2 to TIM4) and if UDIS=0 in the TIMx\_CR1 register.  
When CNT is reinitialized by software using the UG bit in TIMx\_EGR register, if URS=0 and UDIS=0 in the TIMx\_CR1 register.  
When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx\_CR1 register.

### 23.4.6 TIMx event generation register (TIMx\_EGR)(x = 2 to 5)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TG	Res.	CC4G	CC3G	CC2G	CC1G	UG								

Bits 15:7 Reserved, must be kept at reset value.

- Bit 6 **TG**: Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action  
1: The TIF flag is set in TIMx\_SR register. Related interrupt or DMA transfer can occur if enabled.

- Bit 5 Reserved, must be kept at reset value.

- Bit 4 **CC4G**: Capture/compare 4 generation

Refer to CC1G description

- Bit 3 **CC3G**: Capture/compare 3 generation  
Refer to CC1G description
- Bit 2 **CC2G**: Capture/compare 2 generation  
Refer to CC1G description
- Bit 1 **CC1G**: Capture/compare 1 generation  
This bit is set by software in order to generate an event, it is automatically cleared by hardware.  
0: No action  
1: A capture/compare event is generated on channel 1:  
**If channel CC1 is configured as output:**  
CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.  
**If channel CC1 is configured as input:**  
The current value of the counter is captured in TIMx\_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.
- Bit 0 **UG**: Update generation  
This bit can be set by software, it is automatically cleared by hardware.  
0: No action  
1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx\_ARR) if DIR=1 (downcounting).

### 23.4.7 TIMx capture/compare mode register 1 (TIMx\_CCMR1)(x = 2 to 5)

Address offset: 0x18

Reset value: 0x0000 0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode.

#### Input capture 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
IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:12 **IC2F[3:0]**: Input capture 2 filter

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Bits 9:8 **CC2S[1:0]**: Capture/compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output.

01: CC2 channel is configured as input, IC2 is mapped on TI2.

10: CC2 channel is configured as input, IC2 is mapped on TI1.

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx\_CCER).*

Bits 7:4 **IC1F[3:0]**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

0000: No filter, sampling is done at  $f_{DTS}$

0001:  $f_{SAMPLING} = f_{CK\_INT}$ , N=2

0010:  $f_{SAMPLING} = f_{CK\_INT}$ , N=4

0011:  $f_{SAMPLING} = f_{CK\_INT}$ , N=8

0100:  $f_{SAMPLING} = f_{DTS}/2$ , N=6

0101:  $f_{SAMPLING} = f_{DTS}/2$ , N=8

0110:  $f_{SAMPLING} = f_{DTS}/4$ , N=6

0111:  $f_{SAMPLING} = f_{DTS}/4$ , N=8

1000:  $f_{SAMPLING} = f_{DTS}/8$ , N=6

1001:  $f_{SAMPLING} = f_{DTS}/8$ , N=8

1010:  $f_{SAMPLING} = f_{DTS}/16$ , N=5

1011:  $f_{SAMPLING} = f_{DTS}/16$ , N=6

1100:  $f_{SAMPLING} = f_{DTS}/16$ , N=8

1101:  $f_{SAMPLING} = f_{DTS}/32$ , N=5

1110:  $f_{SAMPLING} = f_{DTS}/32$ , N=6

1111:  $f_{SAMPLING} = f_{DTS}/32$ , N=8

Bits 3:2 **IC1PSC[1:0]**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1). The prescaler is reset as soon as CC1E=0 (TIMx\_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input

01: capture is done once every 2 events

10: capture is done once every 4 events

11: capture is done once every 8 events

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

### 23.4.8 TIMx capture/compare mode register 1 [alternate] (TIMx\_CCMR1) (x = 2 to 5)

Address offset: 0x18

Reset value: 0x0000 0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode.

#### Output compare mode:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
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 23:17 Reserved, must be kept at reset value.

Bit 15 **OC2CE**: Output compare 2 clear enable

Bits 24, 14:12 **OC2M[3:0]**: Output compare 2 mode  
refer to OC1M description on bits 6:4

Bit 11 **OC2PE**: Output compare 2 preload enable

Bit 10 **OC2FE**: Output compare 2 fast enable

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx\_CCER).*

Bit 7 **OC1CE**: Output compare 1 clear enable

0: OC1Ref is not affected by the ETRF input

1: OC1Ref is cleared as soon as a High level is detected on ETRF input

Bits 16, 6:4 **OC1M[3:0]**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

0000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs. This mode can be used when the timer serves as a software timebase. When the frozen mode is enabled during timer operation, the output keeps the state (active or inactive) it had before entering the frozen state.

0001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

0010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

0011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.

0100: Force inactive level - OC1REF is forced low.

0101: Force active level - OC1REF is forced high.

0110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0) as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF=1).

0111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

1000: Retriggerable OPM mode 1 - In up-counting mode, the channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes inactive again at the next update. In down-counting mode, the channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes inactive again at the next update.

1001: Retriggerable OPM mode 2 - In up-counting mode, the channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 2 and the channels becomes inactive again at the next update. In down-counting mode, the channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes active again at the next update.

1010: Reserved,

1011: Reserved,

1100: Combined PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC is the logical OR between OC1REF and OC2REF.

1101: Combined PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC is the logical AND between OC1REF and OC2REF.

1110: Asymmetric PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC outputs OC1REF when the counter is counting up, OC2REF when it is counting down.

1111: Asymmetric PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC outputs OC1REF when the counter is counting up, OC2REF when it is counting down.

*Note: In PWM mode, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode.*

*Note: The OC1M[3] bit is not contiguous, located in bit 16.*

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.

Bit 2 **OC1FE**: Output compare 1 fast enable

This bit decreases the latency between a trigger event and a transition on the timer output. It must be used in one-pulse mode (OPM bit set in TIMx\_CR1 register), to have the output pulse starting as soon as possible after the starting trigger.

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[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: CC1 channel is configured as input, IC1 is mapped on TI2.

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

### 23.4.9 TIMx capture/compare mode register 2 (TIMx\_CCMR2)(x = 2 to 5)

Address offset: 0x1C

Reset value: 0x0000 0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode.

**Input capture 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															
IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]		IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
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 15:12 **IC4F[3:0]**: Input capture 4 filter

Bits 11:10 **IC4PSC[1:0]**: Input capture 4 prescaler

Bits 9:8 **CC4S[1:0]**: 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[3:0]**: Input capture 3 filter

Bits 3:2 **IC3PSC[1:0]**: Input capture 3 prescaler

Bits 1:0 **CC3S[1:0]**: Capture/Compare 3 selection

This 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).*

### 23.4.10 TIMx capture/compare mode register 2 [alternate] (TIMx\_CCMR2) (x = 2 to 5)

Address offset: 0x1C

Reset value: 0x0000 0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode.

**Output compare mode:**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC4M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC3M [3]
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]		OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
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 23:17 Reserved, must be kept at reset value.

Bit 15 **OC4CE**: Output compare 4 clear enable

Bits 24, 14:12 **OC4M[3:0]**: Output compare 4 mode

Refer to OC1M description (bits 6:4 in TIMx\_CCMR1 register)

Bit 11 **OC4PE**: Output compare 4 preload enable

Bit 10 **OC4FE**: Output compare 4 fast enable

Bits 9:8 **CC4S[1:0]**: 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 16, 6:4 **OC3M[3:0]**: Output compare 3 mode

Refer to OC1M description (bits 6:4 in TIMx\_CCMR1 register)

Bit 3 **OC3PE**: Output compare 3 preload enable

Bit 2 **OC3FE**: Output compare 3 fast enable

Bits 1:0 **CC3S[1:0]**: 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).*

### 23.4.11 TIMx capture/compare enable register (TIMx\_CCER)(x = 2 to 5)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E

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.*

0: OC1 active high (output mode) / Edge sensitivity selection (input mode, see below)

1: OC1 active low (output mode) / Edge sensitivity selection (input mode, see below)

**When CC1 channel is configured as input**, both CC1NP/CC1P bits select the active polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

CC1NP=0, CC1P=0: 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).

CC1NP=0, CC1P=1: 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).

CC1NP=1, CC1P=1: 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.

CC1NP=1, CC1P=0: This configuration is reserved, it must not be used.

Bit 0 **CC1E**: *Capture/Compare 1 output enable.*

0: Capture mode disabled / OC1 is not active

1: Capture mode enabled / OC1 signal is output on the corresponding output pin

**Table 155. Output control bit for standard OCx channels**

CCxE bit	OCx output state
0	Output disabled (not driven by the timer: Hi-Z)
1	Output enabled (tim_ocx = tim_ocxref + Polarity)

**Note:** The state of the external IO pins connected to the standard OCx channels depends on the OCx channel state and the GPIO control and alternate function registers.

### 23.4.12 TIMx counter (TIMx\_CNT)(x = 2 to 5)

Bit 31 of this register has two possible definitions depending on the value of UIFREMAP in TIMx\_CR1 register:

- This section is for UIFREMAP = 0
- Next section is for UIFREMAP = 1

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNT[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CNT[31:16]**: Most significant part counter value (TIM2 and TIM5)

Bits 15:0 **CNT[15:0]**: Least significant part of counter value

### 23.4.13 TIMx counter [alternate] (TIMx\_CNT)(x = 2 to 5)

Bit 31 of this register has two possible definitions depending on the value of UIFREMAP in TIMx\_CR1 register:

- Previous section is for UIFREMAP = 0
- This section is for UIFREMAP = 1

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UIFCPY															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[30:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UIFCPY**: UIF Copy

This bit is a read-only copy of the UIF bit of the TIMx\_ISR register

Bits 30:16 **CNT[30:16]**: Most significant part counter value (TIM2 and TIM5)

Bits 15:0 **CNT[15:0]**: Least significant part of counter value

### 23.4.14 TIMx prescaler (TIMx\_PSC)(x = 2 to 5)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

### 23.4.15 TIMx auto-reload register (TIMx\_ARR)(x = 2 to 5)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **ARR[31:16]**: High auto-reload value (TIM2 and TIM5)

Bits 15:0 **ARR[15:0]**: Low Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 23.3.1: Time-base unit on page 733](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 23.4.16 TIMx capture/compare register 1 (TIMx\_CCR1)(x = 2 to 5)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR1[31:16]															
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 (TIM2 and TIM5)

Bits 15:0 **CCR1[15:0]**: Low Capture/Compare 1 value

**If channel CC1 is configured as output:**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**If channel CC1 is configured as input:**

CCR1 is the counter value transferred by the last input capture 1 event (IC1). The TIMx\_CCR1 register is read-only and cannot be programmed.

### 23.4.17 TIMx capture/compare register 2 (TIMx\_CCR2)(x = 2 to 5)

Address offset: 0x38

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR2[31:16]															
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 (TIM2 and TIM5)

Bits 15:0 **CCR2[15:0]**: Low Capture/Compare 2 value

**If channel CC2 is configured as output:**

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC2 output.

**If channel CC2 is configured as input:**

CCR2 is the counter value transferred by the last input capture 2 event (IC2). The TIMx\_CCR2 register is read-only and cannot be programmed.

### 23.4.18 TIMx capture/compare register 3 (TIMx\_CCR3)(x = 2 to 5)

Address offset: 0x3C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR3[31:16]															
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 (TIM2 and TIM5)

Bits 15:0 **CCR3[15:0]**: Low Capture/Compare value

**If channel CC3 is configured as output:**

CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC3 output.

**If channel CC3 is configured as input:**

CCR3 is the counter value transferred by the last input capture 3 event (IC3). The TIMx\_CCR3 register is read-only and cannot be programmed.

### 23.4.19 TIMx capture/compare register 4 (TIMx\_CCR4)(x = 2 to 5)

Address offset: 0x40

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR4[31:16]															
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 (TIM2 and TIM5)

Bits 15:0 **CCR4[15:0]**: Low Capture/Compare value

1. if CC4 channel is configured as output (CC4S bits):  
CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (bit OC4PE). Else the preload value is copied in the active capture/compare 4 register when an update event occurs.  
The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC4 output.
2. if CC4 channel is configured as input (CC4S bits in TIMx\_CCMR4 register):  
CCR4 is the counter value transferred by the last input capture 4 event (IC4). The TIMx\_CCR4 register is read-only and cannot be programmed.

### 23.4.20 TIMx DMA control register (TIMx\_DCR)(x = 2 to 5)

Address offset: 0x48

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	DBL[4:0]				Res.	Res.	Res.	DBA[4:0]					
			rw	rw	rw	rw	rw				rw	rw	rw	rw	rw

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:8 **DBL[4:0]**: DMA burst length

This 5-bit vector defines the number of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx\_DMAR address).

00000: 1 transfer,  
00001: 2 transfers,  
00010: 3 transfers,  
...  
10001: 18 transfers.

Bits 7:5 Reserved, must be kept at reset value.

Bits 4:0 **DBA[4:0]**: DMA base address

This 5-bit vector defines the base-address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register.

Example:  
00000: TIMx\_CR1  
00001: TIMx\_CR2  
00010: TIMx\_SMCR  
...

**Example:** Let us consider the following transfer: DBL = 7 transfers & DBA = TIMx\_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx\_CR1 address.

### 23.4.21 TIMx DMA address for full transfer (TIMx\_DMAR)(x = 2 to 5)

Address offset: 0x4C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **DMAB[15:0]**: DMA register for burst accesses

A read or write 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).

### 23.4.22 TIM2 option register (TIM2\_OR)

Address offset: 0x50

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	ITR1_RMP[1:0]	Res.										
				rw	rw										

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:10 **ITR1\_RMP[1:0]**: Internal trigger 1 remap

Set and cleared by software.

00: TIM8\_TRGOUT

01: ETH\_PTP trigger output is connected to TIM2\_ITR1

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.

### 23.4.23 TIM5 option register (TIM5\_OR)

Address offset: 0x50

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TI4_RMP[1:0]	Res.													
								rw	rw						

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:6 **TI4\_RMP[1:0]**: Timer Input 4 remap

Set and cleared by software.

00: TIM5 channel4 is connected to the GPIO: Refer to the alternate function mapping table in the datasheets.

01: The LSI internal clock is connected to the TIM5\_CH4 input for calibration purposes

10: The LSE internal clock is connected to the TIM5\_CH4 input for calibration purposes

11: The RTC wakeup interrupt is connected to the TIM5\_CH4 input for calibration purposes.

Wakeup interrupt should be enabled.

Bits 5:0 Reserved, must be kept at reset value.

### 23.4.24 TIMx register map

TIMx registers are mapped as described in the table below:

**Table 156. TIM2/TIM3/TIM4/TIM5 register map and reset values**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Res																															
	Reset value																																
0x04	TIMx_CR2	Res																															
	Reset value																																
0x08	TIMx_SMCR	Res																															
	Reset value																																
0x0C	TIMx_DIER	Res																															
	Reset value																																
0x10	TIMx_SR	Res																															
	Reset value																																
0x14	TIMx_EGR	Res																															
	Reset value																																
0x18	TIMx_CCMR1 Output Compare mode	Res																															
	Reset value																																
	TIMx_CCMR1 Input Capture mode	Res																															
0x1C	TIMx_CCMR2 Output Compare mode	Res																															
	Reset value																																
	TIMx_CCMR2 Input Capture mode	Res																															
0x20	TIMx_CCER	Res																															
	Reset value																																

Table 156. TIM2/TIM3/TIM4/TIM5 register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x24	<b>TIMx_CNT</b>	CNT[31] or UIFCPY	CNT[30:16] (TIM2 and TIM5 only, reserved on the other timers)															CNT[15:0]															
			Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x28	<b>TIMx_PSC</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PSC[15:0]															
			Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x2C	<b>TIMx_ARR</b>	ARR[31:16] (TIM2 and TIM5 only, reserved on the other timers)	ARR[31:16] (TIM2 and TIM5 only, reserved on the other timers)															ARR[15:0]															
			Reset value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
0x30		Reserved																															
0x34	<b>TIMx_CCR1</b>	CCR1[31:16] (TIM2 and TIM5 only, reserved on the other timers)	CCR1[31:16] (TIM2 and TIM5 only, reserved on the other timers)															CCR1[15:0]															
			Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x38	<b>TIMx_CCR2</b>	CCR2[31:16] (TIM2 and TIM5 only, reserved on the other timers)	CCR2[31:16] (TIM2 and TIM5 only, reserved on the other timers)															CCR2[15:0]															
			Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x3C	<b>TIMx_CCR3</b>	CCR3[31:16] (TIM2 and TIM5 only, reserved on the other timers)	CCR3[31:16] (TIM2 and TIM5 only, reserved on the other timers)															CCR3[15:0]															
			Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x40	<b>TIMx_CCR4</b>	CCR4[31:16] (TIM2 and TIM5 only, reserved on the other timers)	CCR4[31:16] (TIM2 and TIM5 only, reserved on the other timers)															CCR4[15:0]															
			Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x44		Reserved																															
0x48	<b>TIMx_DCR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBL[4:0]					Res.	Res.	Res.	DBA[4:0]		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		
0x4C	<b>TIMx_DMAR</b>	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.	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	

**Table 156. TIM2/TIM3/TIM4/TIM5 register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x50	TIM5_OR	Res.																															
	Reset value																																

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 24 General-purpose timers (TIM9/TIM10/TIM11/TIM12/TIM13/TIM14)

### 24.1 TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 introduction

The TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 general-purpose timers consist in a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

The TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 24.3.16: Timer synchronization \(TIM9/TIM12\)](#).

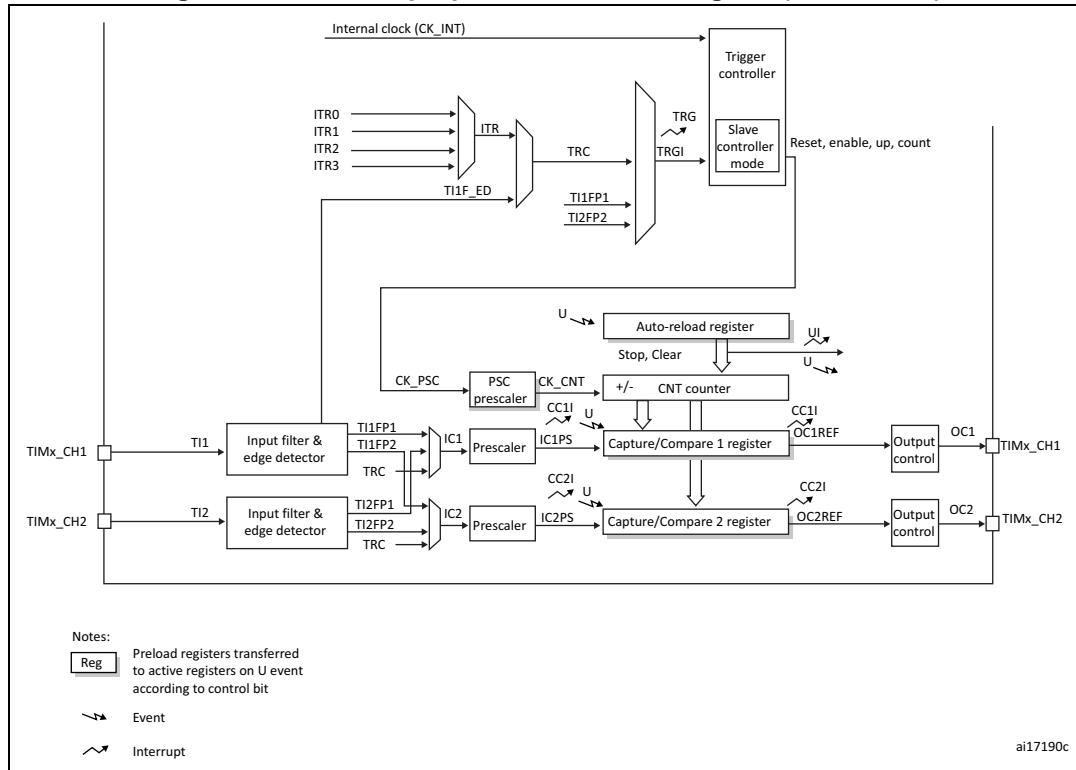
### 24.2 TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 main features

#### 24.2.1 TIM9/TIM12 main features

The features of the TIM9/TIM12 general-purpose timers include:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65536 (can be changed “on the fly”)
- Up to 2 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (edge-aligned mode)
  - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers together
- Interrupt generation on the following events:
  - Update: counter overflow, counter initialization (by software or internal trigger)
  - Trigger event (counter start, stop, initialization or count by internal trigger)
  - Input capture
  - Output compare

Figure 253. General-purpose timer block diagram (TIM9/TIM12)

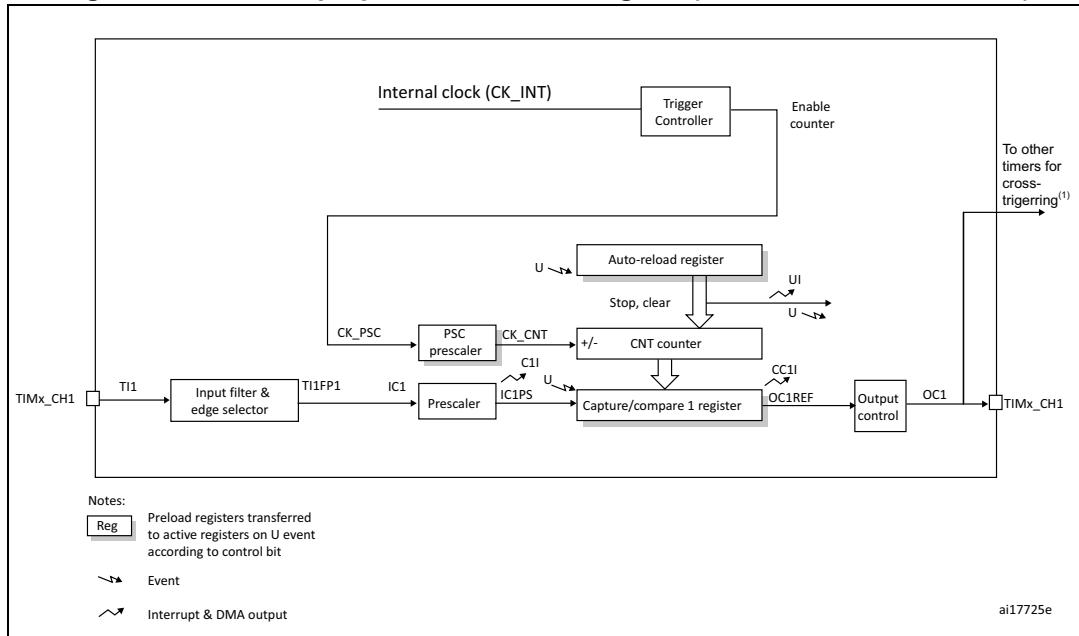


## 24.2.2 TIM10/TIM11/TIM13/TIM14 main features

The features of general-purpose timers TIM10/TIM11/TIM13/TIM14 include:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65536 (can be changed “on the fly”)
- independent channel for:
  - Input capture
  - Output compare
  - PWM generation (edge-aligned mode)
  - One-pulse mode output
- Interrupt generation on the following events:
  - Update: counter overflow, counter initialization (by software)
  - Input capture
  - Output compare

Figure 254. General-purpose timer block diagram (TIM10/TIM11/TIM13/TIM14)



1. This signal can be used as trigger for some slave timers, see [Section 24.3.17: Using timer output as trigger for other timers \(TIM10/TIM11/TIM13/TIM14\)](#).

## 24.3 TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 functional description

### 24.3.1 Time-base unit

The main block of the timer is a 16-bit up-counter with its related auto-reload register. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter register (TIMx\_CNT)
- Prescaler register (TIMx\_PSC)
- Auto-reload register (TIMx\_ARR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx\_CR1 register. The update event is sent when the counter reaches the overflow and if the UDIS bit equals 0 in the TIMx\_CR1 register. It can also be generated by software. The generation of the update event is described in details for each configuration.

The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

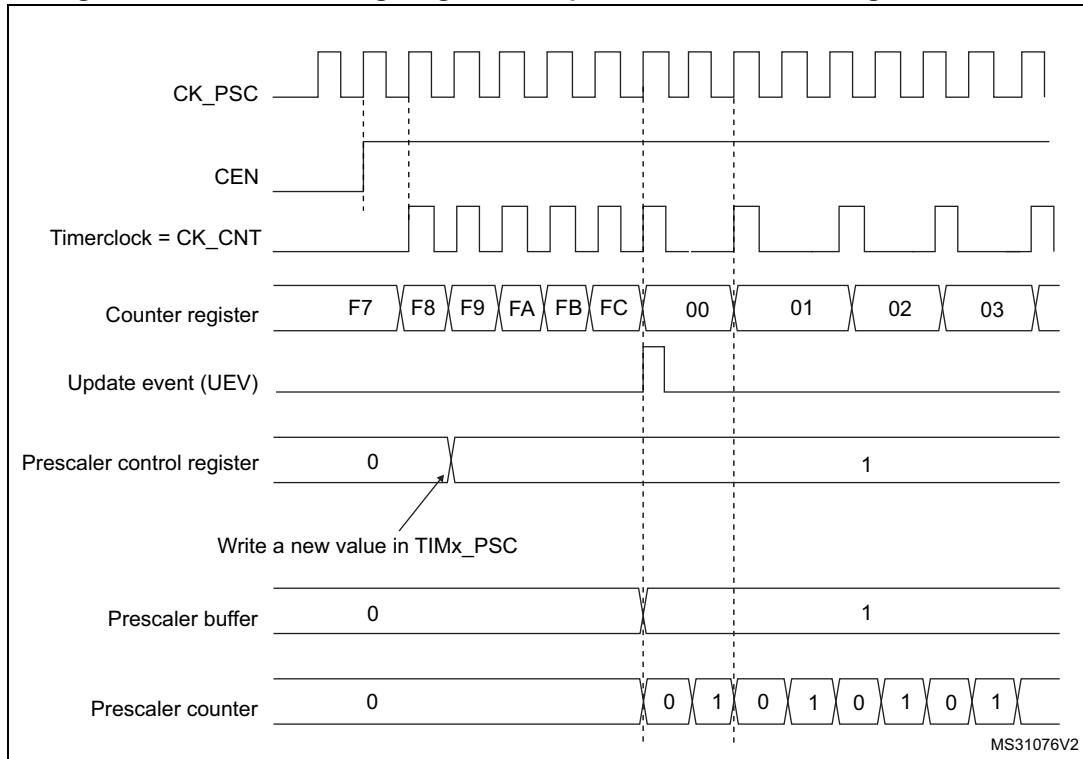
Note that the counter starts counting 1 clock cycle after setting the CEN bit in the TIMx\_CR1 register.

#### Prescaler description

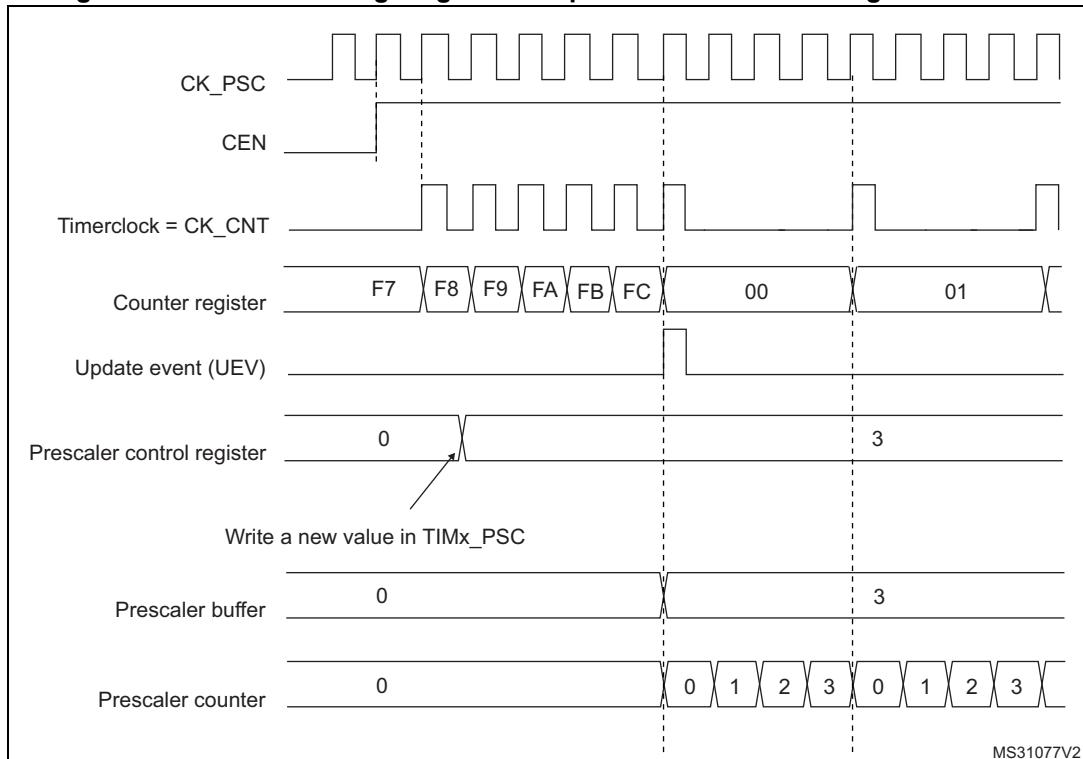
The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx\_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

*Figure 255* and *Figure 256* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

**Figure 255.** Counter timing diagram with prescaler division change from 1 to 2



**Figure 256.** Counter timing diagram with prescaler division change from 1 to 4



## 24.3.2 Counter modes

### Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register), then restarts from 0 and generates a counter overflow event.

Setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller on TIM9/TIM12) also generates an update event.

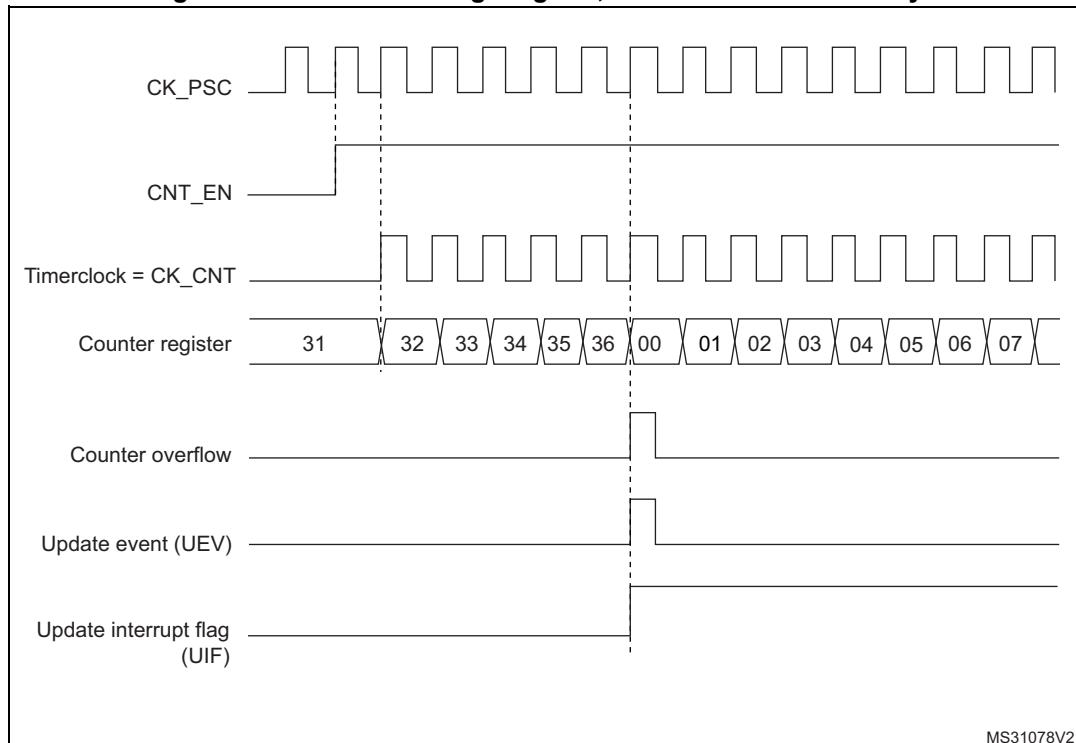
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

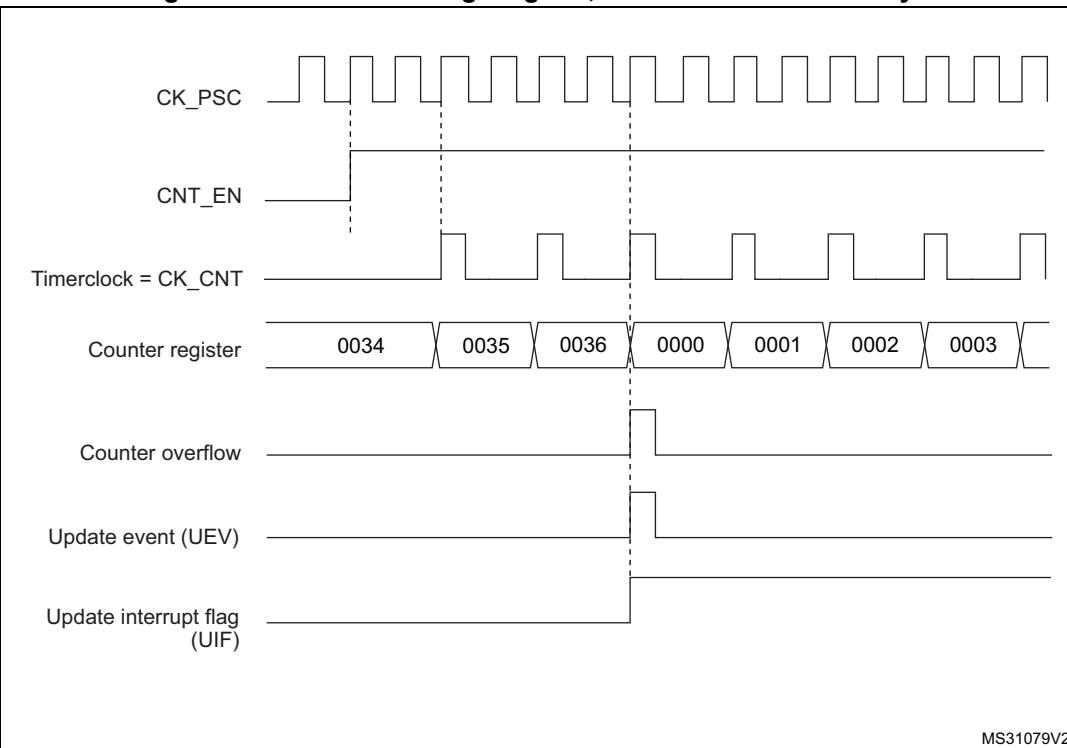
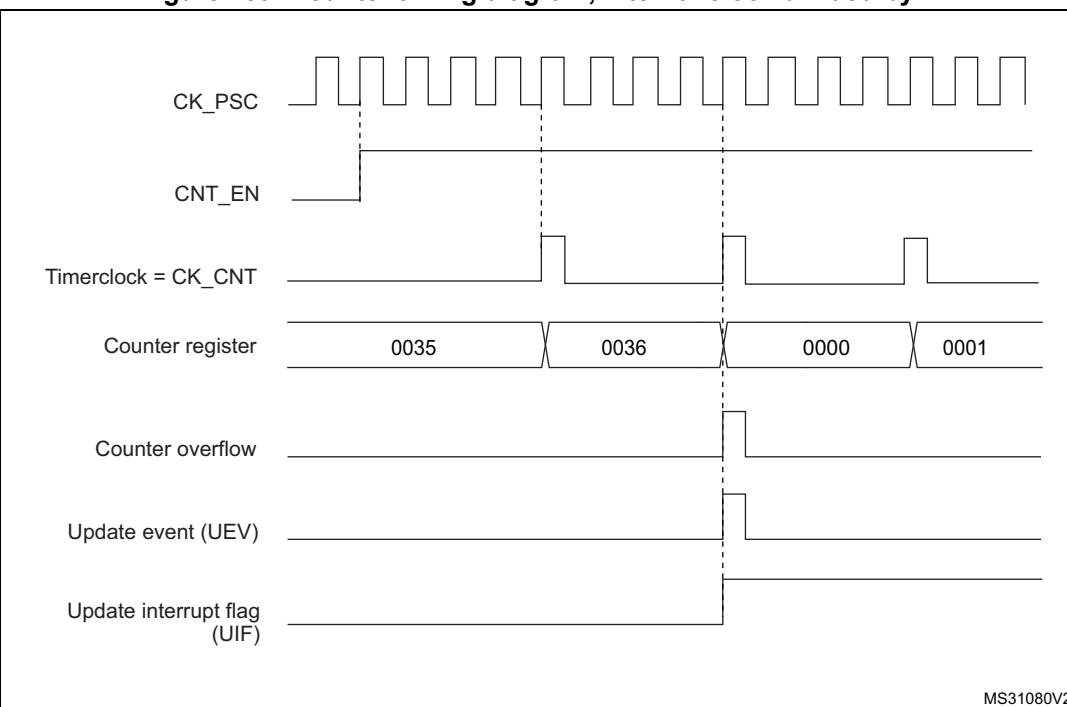
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

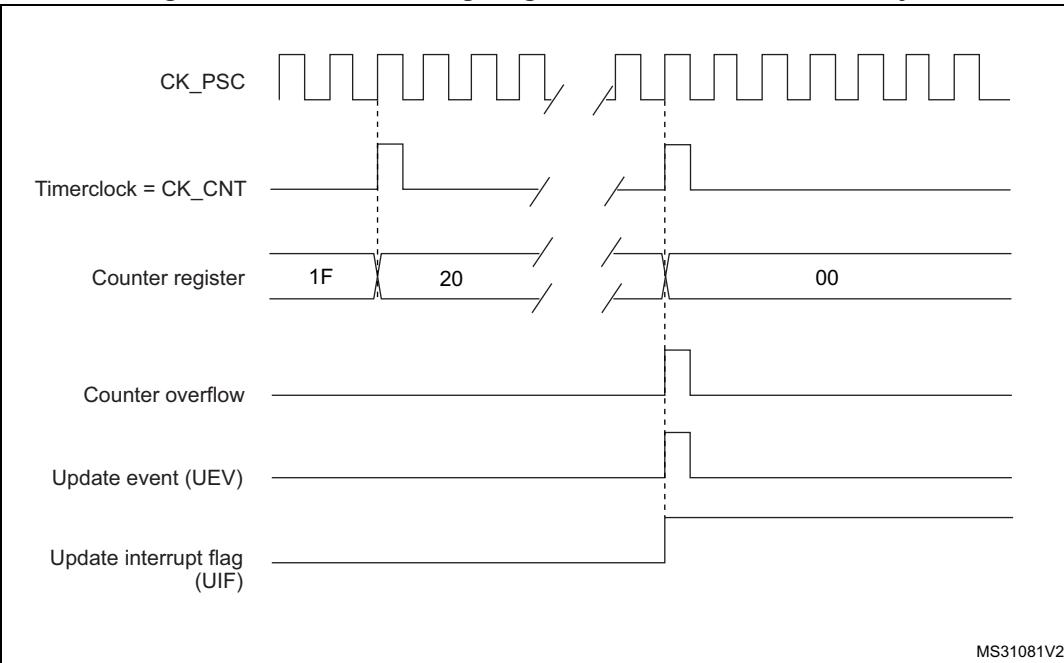
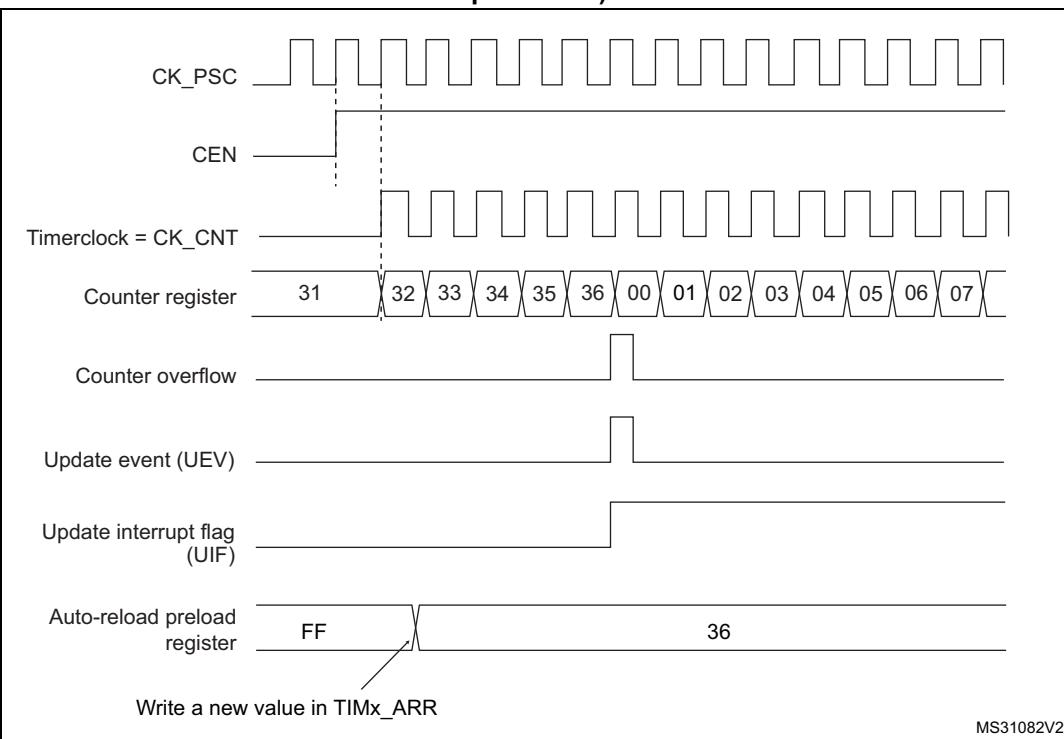
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

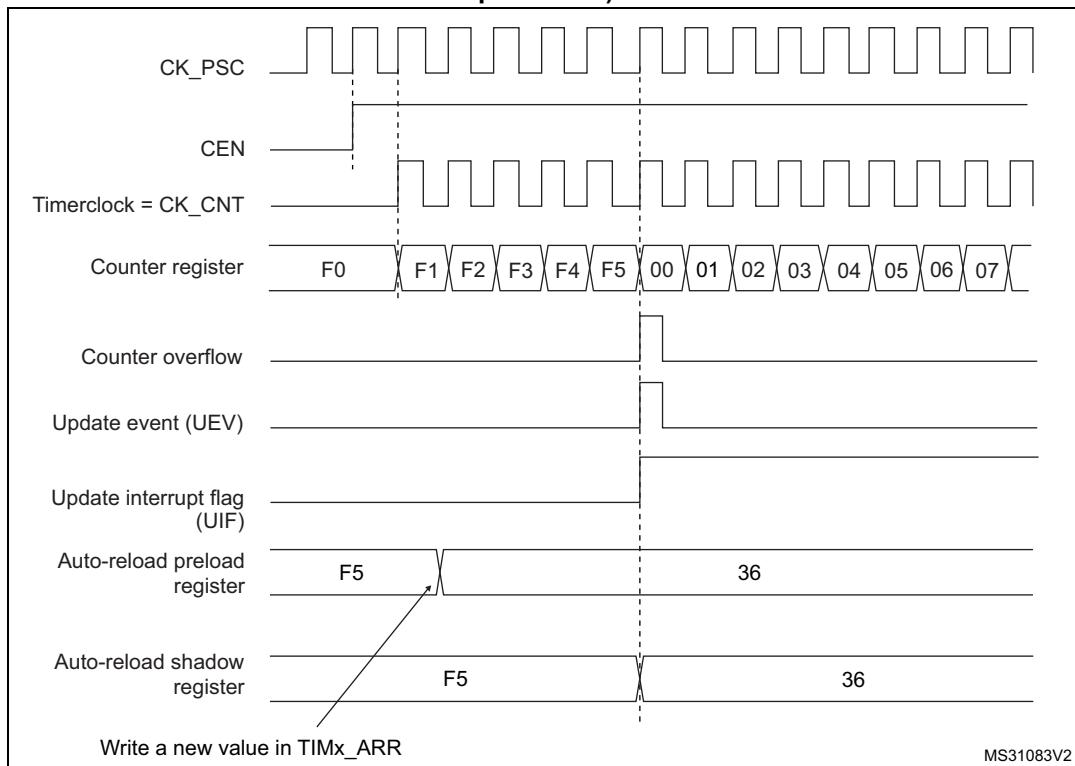
**Figure 257. Counter timing diagram, internal clock divided by 1**



**Figure 258. Counter timing diagram, internal clock divided by 2****Figure 259. Counter timing diagram, internal clock divided by 4**

**Figure 260. Counter timing diagram, internal clock divided by N****Figure 261. Counter timing diagram, update event when ARPE=0 (TIMx\_ARR not preloaded)**

**Figure 262. Counter timing diagram, update event when ARPE=1 (TIMx\_ARR preloaded)**



MS31083V2

### 24.3.3 Clock selection

The counter clock can be provided by the following clock sources:

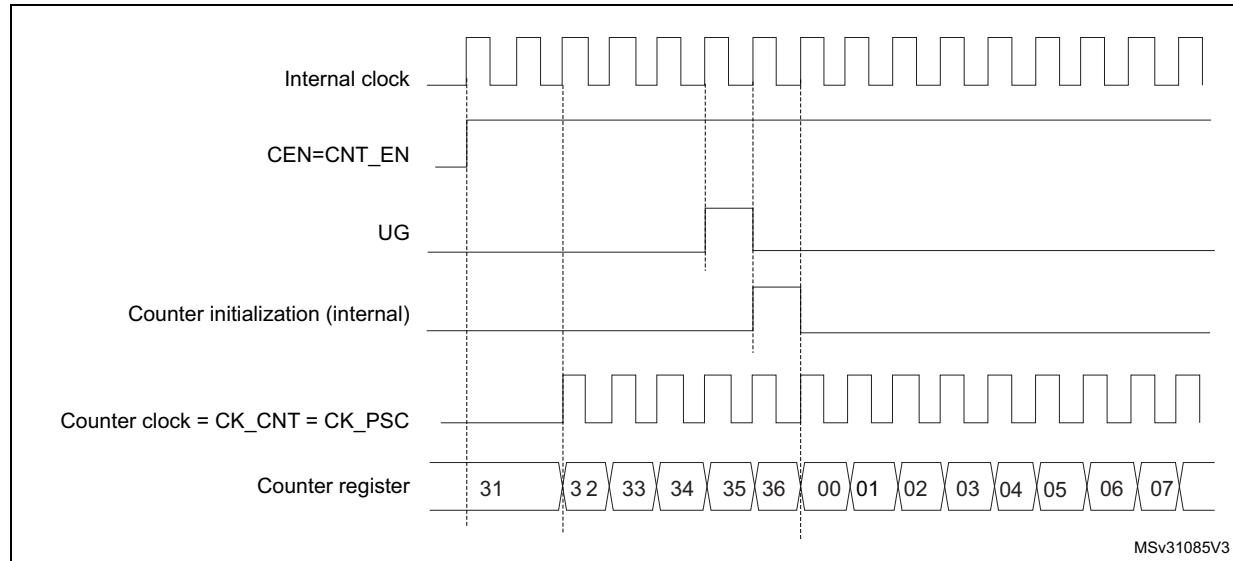
- Internal clock (CK\_INT)
- External clock mode1 (for TIM9/TIM12): external input pin (TIx)
- Internal trigger inputs (ITRx) (for TIM9/TIM12): connecting the trigger output from another timer. For instance, another timer can be configured as a prescaler for TIM12. Refer to [Section : Using one timer as prescaler for another timer](#) for more details.

#### Internal clock source (CK\_INT)

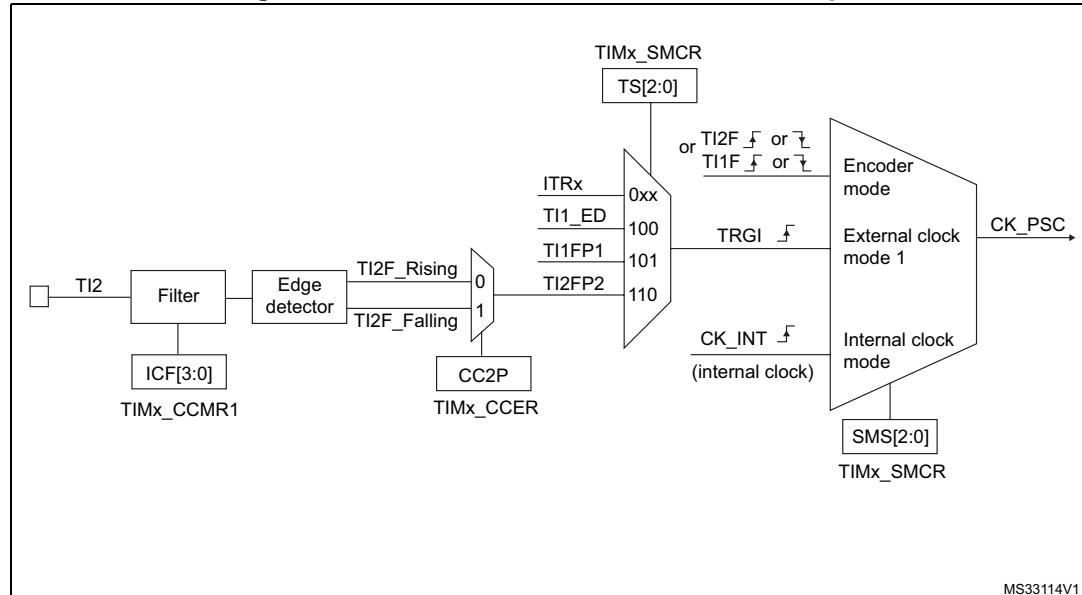
The internal clock source is the default clock source for TIM10/TIM11/TIM13/TIM14.

For TIM9/TIM12, the internal clock source is selected when the slave mode controller is disabled (SMS='000'). The CEN bit in the TIMx\_CR1 register and the UG bit in the TIMx\_EGR register are then used as control bits and can be changed only by software (except for UG which remains cleared). As soon as the CEN bit is programmed to 1, the prescaler is clocked by the internal clock CK\_INT.

[Figure 263](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

**Figure 263. Control circuit in normal mode, internal clock divided by 1****External clock source mode 1 (TIM9/TIM12)**

This mode is selected when SMS='111' in the TIMx\_SMCR register. The counter can count at each rising or falling edge on a selected input.

**Figure 264. 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:

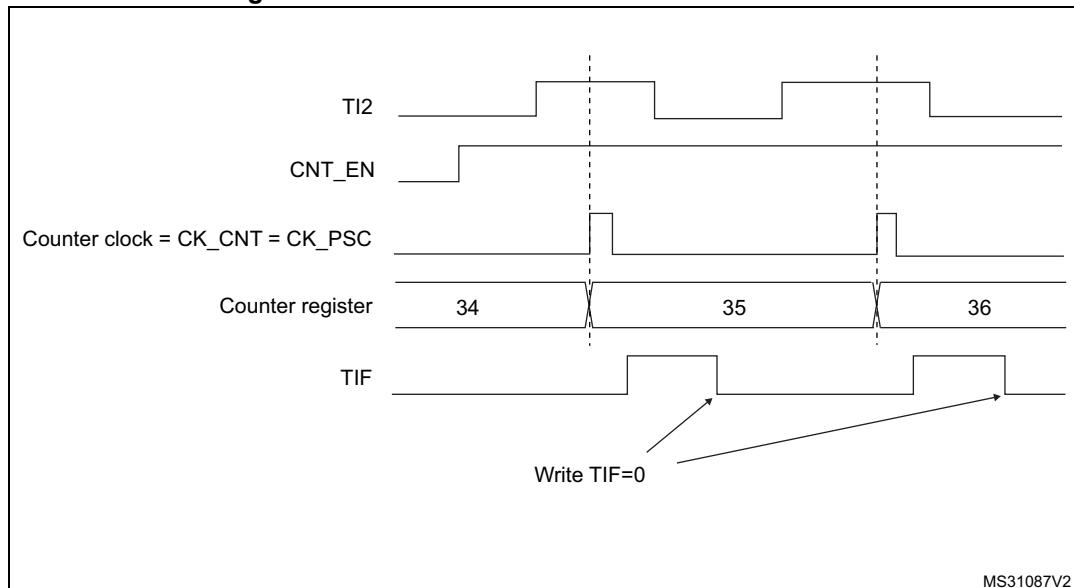
1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = '01' in the TIMx\_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx\_CCMR1 register (if no filter is needed, keep IC2F='0000').
3. Select the rising edge polarity by writing 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 it does not need to be configured.

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 265. Control circuit in external clock mode 1**



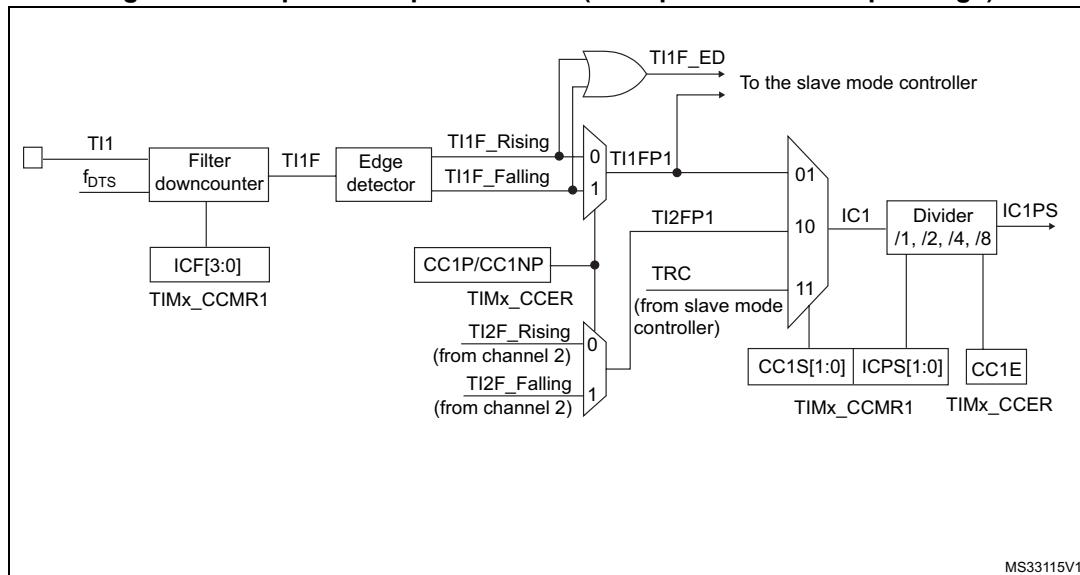
#### 24.3.4 Capture/compare channels

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 266](#) to [Figure 268](#) 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 266. 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 267. Capture/compare channel 1 main circuit

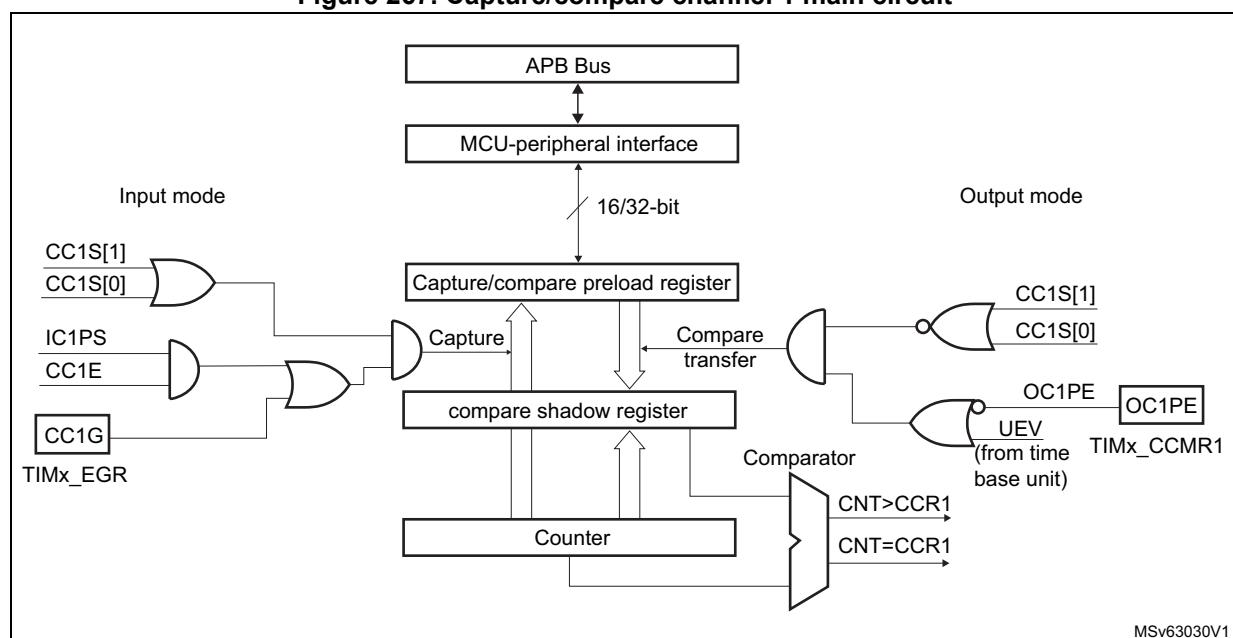
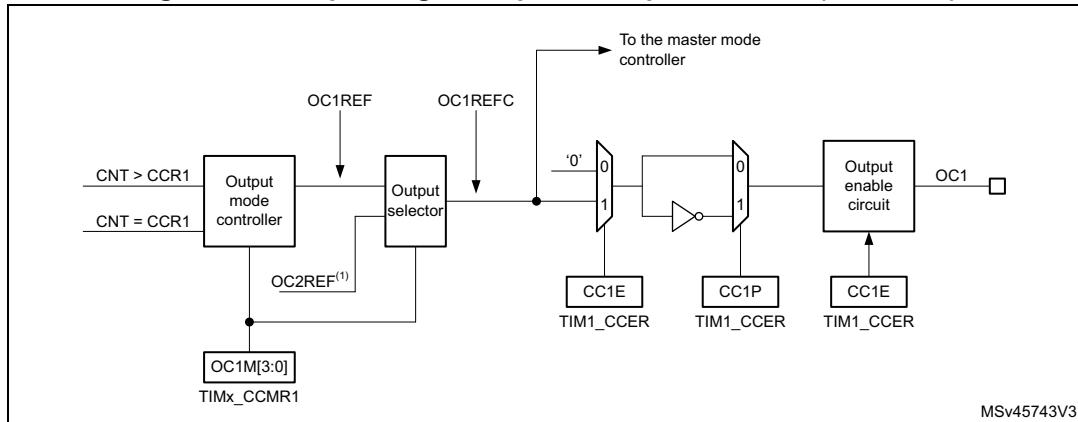


Figure 268. Output stage of capture/compare channel (channel 1)



1. Available on TIM12 only.

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.

#### 24.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCxIF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx\_SR register) is set. CCxIF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx\_CCRx register. CCxOF is cleared when it is written with 0.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

1. Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to '01' in the TIMx\_CCMR1 register. As soon as CC1S becomes different from '00', the channel is configured in input mode and the TIMx\_CCR1 register becomes read-only.
2. Program the appropriate input filter duration in relation with the signal connected to the timer (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.*

#### 24.3.6 PWM input mode (only for TIM9/TIM12)

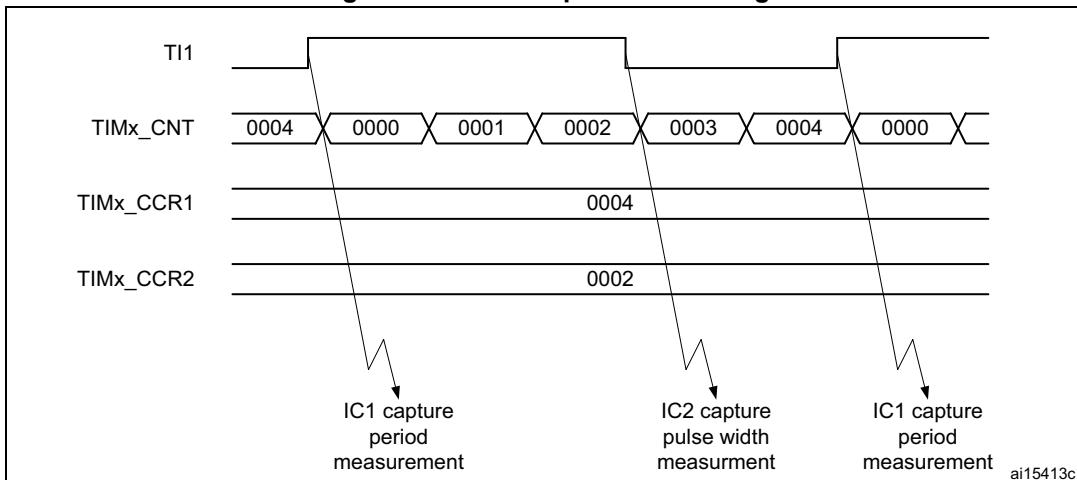
This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, one can measure the period (in TIMx\_CCR1 register) and the duty cycle (in TIMx\_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK\_INT frequency and prescaler value):

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 '10' (active on falling edge).
5. Select the valid trigger input: write the TS bits to '101' in the TIMx\_SMCR register (TI1FP1 selected).
6. Configure the slave mode controller in reset mode: write the SMS bits to '100' in the TIMx\_SMCR register.
7. Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx\_CCER register.

Figure 269. 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.

### 24.3.7 Forced output mode

In output mode (CCxS bits = '00' in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, one just needs to write '0101' in the OCxM bits in the corresponding TIMx\_CCMRx register. Thus OCXREF is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

For example: CCxP='0' (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to '0100' in the TIMx\_CCMRx register.

Anyway, the comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt requests can be sent accordingly. This is described in the output compare mode section below.

### 24.3.8 Output compare mode

This function is used to control an output waveform or indicating when a period of time has elapsed.

When a match is found between the capture/compare register and the counter, the output compare function:

1. Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCxM='0000'), be set active (OCxM='0001'), be set inactive (OCxM='0010') or can toggle (OCxM='0011') on match.
2. Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
3. Generates an interrupt if the corresponding interrupt mask is set (CCxIE bit in the TIMx\_DIER register).

The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

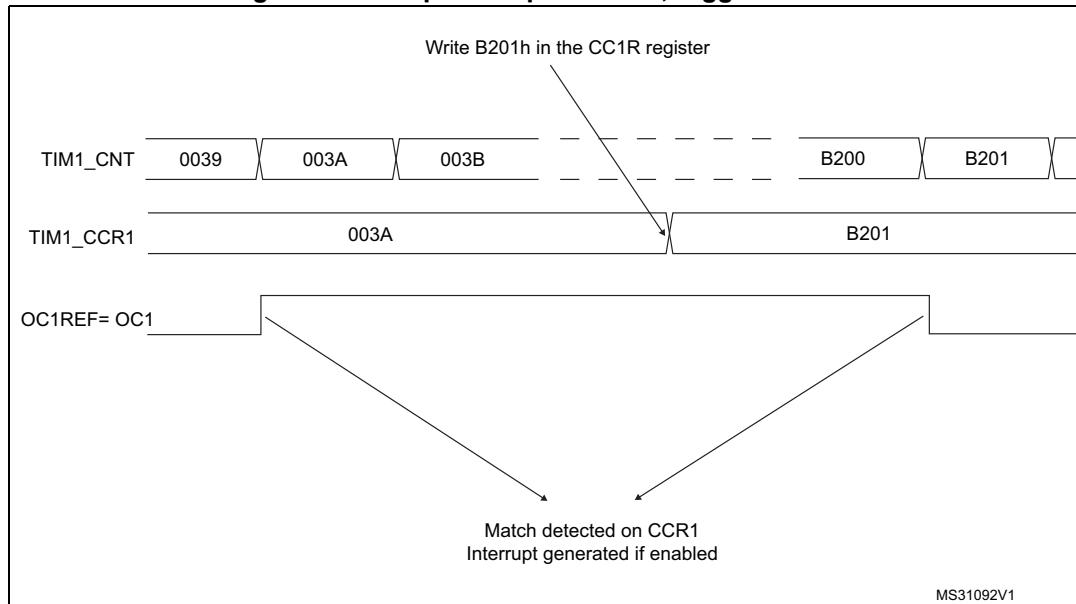
In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One-pulse mode).

Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx\_ARR and TIMx\_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
  - Write OCxM = '0011' to toggle OCx output pin when CNT matches CCRx
  - Write OCxPE = '0' to disable preload register
  - Write CCxP = '0' to select active high polarity
  - Write CCxE = '1' to enable the output
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

The TIMx\_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE='0', else TIMx\_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 270](#).

**Figure 270. Output compare mode, toggle on OC1.**



### 24.3.9 PWM mode

Pulse Width Modulation mode allows to generate a signal with a frequency determined by the value of the TIMx\_ARR register and a duty cycle determined by the value of the TIMx\_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '0110' (PWM mode 1) or '0111' (PWM mode 2) in the OCxM bits in the

`TIMx_CCMRx` register. The corresponding preload register must be enabled by setting the `OCxPE` bit in the `TIMx_CCMRx` register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the `ARPE` bit in the `TIMx_CR1` register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, all registers must be initialized by setting the `UG` bit in the `TIMx_EGR` register.

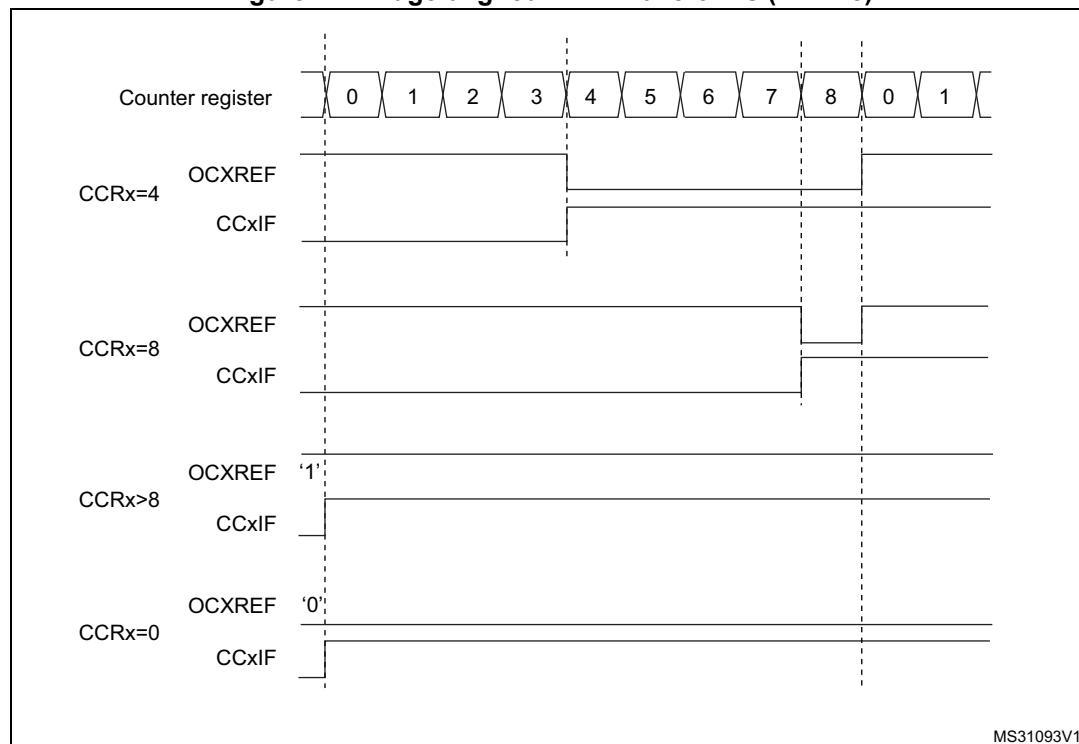
The `OCx` polarity is software programmable using the `CCxP` bit in the `TIMx_CCER` register. It can be programmed as active high or active low. The `OCx` output is enabled by the `CCxE` bit in the `TIMx_CCER` register. Refer to the `TIMx_CCERx` register description for more details.

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.

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 271* shows some edge-aligned PWM waveforms in an example where `TIMx_ARR=8`.

**Figure 271. Edge-aligned PWM waveforms (ARR=8)**



### 24.3.10 Combined PWM mode (TIM9/TIM12 only)

Combined PWM mode allows two edge or center-aligned PWM signals to be generated with programmable delay and phase shift between respective pulses. While the frequency is determined by the value of the `TIMx_ARR` register, the duty cycle and delay are determined

by the two TIMx\_CCRx registers. The resulting signals, OCxREFC, are made of an OR or AND logical combination of two reference PWMs:

- OC1REFC (or OC2REFC) is controlled by the TIMx\_CCR1 and TIMx\_CCR2 registers

Combined PWM mode can be selected independently on two channels (one OCx output per pair of CCR registers) by writing '1100' (Combined PWM mode 1) or '1101' (Combined PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register.

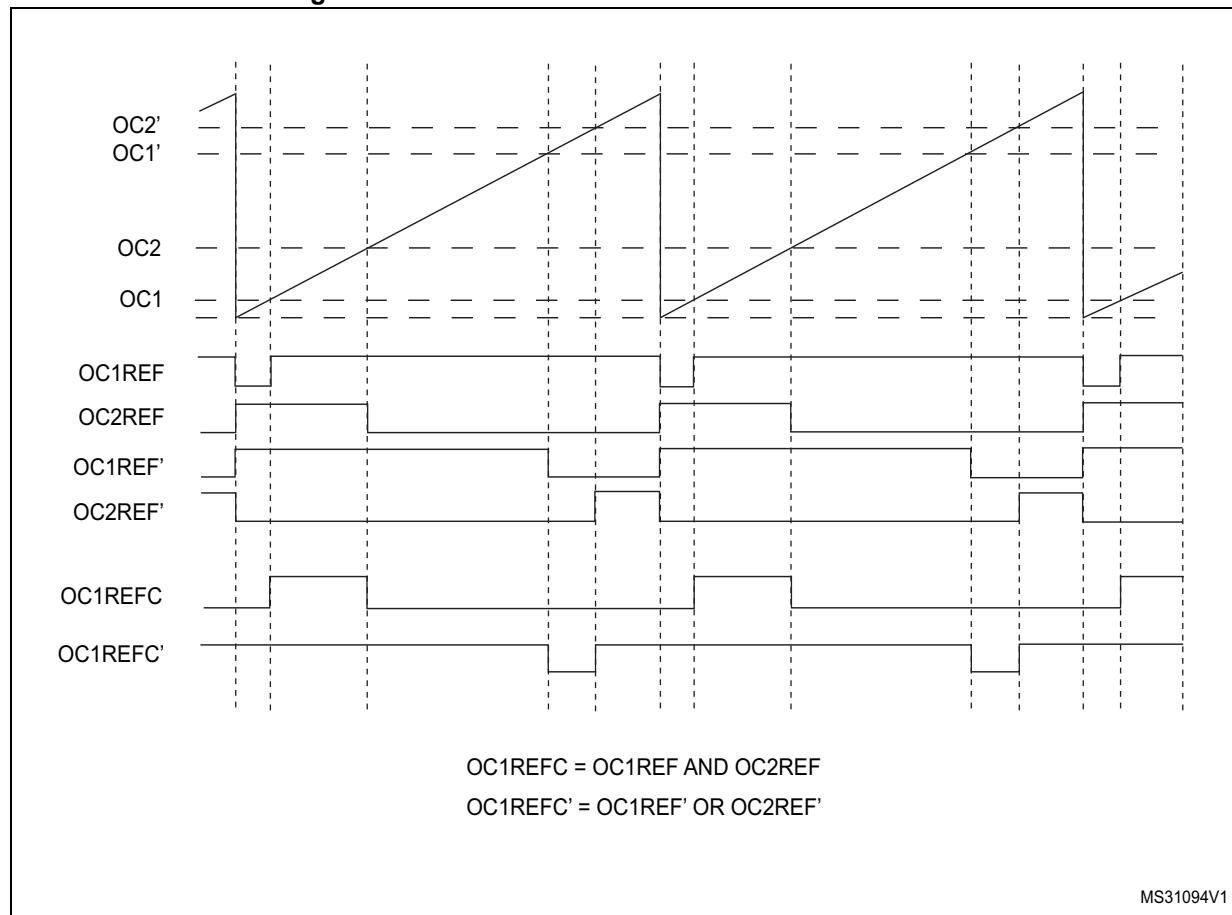
When a given channel is used as a combined PWM channel, its complementary channel must be configured in the opposite PWM mode (for instance, one in Combined PWM mode 1 and the other in Combined PWM mode 2).

**Note:** *The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.*

*Figure 272* represents an example of signals that can be generated using combined PWM mode, obtained with the following configuration:

- Channel 1 is configured in Combined PWM mode 2,
- Channel 2 is configured in PWM mode 1,

**Figure 272. Combined PWM mode on channel 1 and 2**



### 24.3.11 One-pulse mode

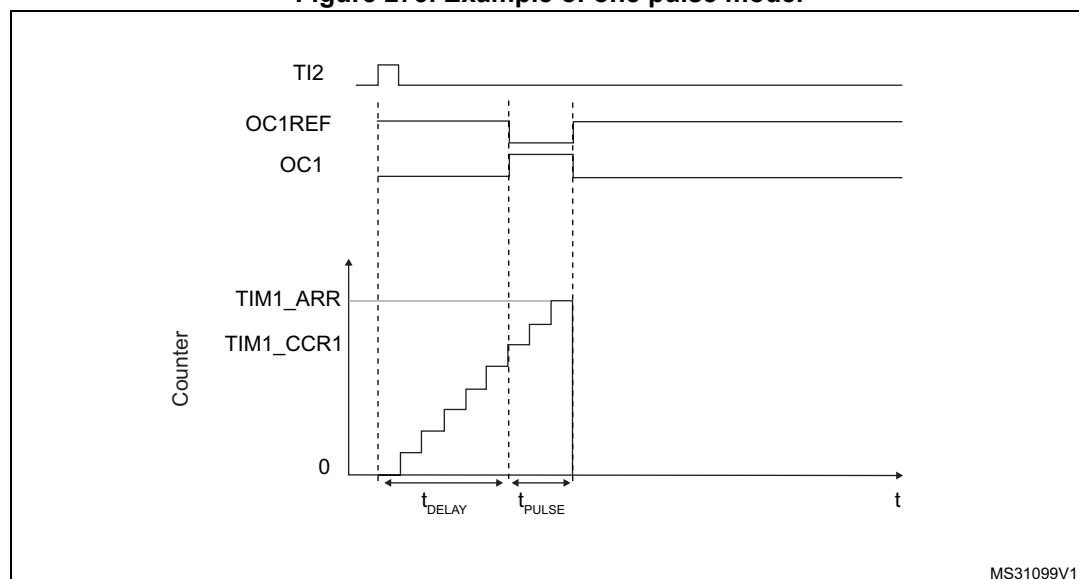
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. One-pulse mode is selected by setting the OPM bit in the TIMx\_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be as follows:

$$\text{CNT} < \text{CCRx} \leq \text{ARR} \text{ (in particular, } 0 < \text{CCRx})$$

**Figure 273. Example of one pulse mode.**



For example one may want to generate a positive pulse on OC1 with a length of  $t_{PULSE}$  and after a delay of  $t_{DELAY}$  as soon as a positive edge is detected on the TI2 input pin.

Use TI2FP2 as trigger 1:

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 one want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this PWM mode 2 must be enabled by writing OC1M='0111' in the TIMx\_CCMR1 register. Optionally the preload registers can be enabled by writing OC1PE='1' in the TIMx\_CCMR1 register and ARPE in the TIMx\_CR1 register. In this case one has to write the compare value in the TIMx\_CCR1 register, the auto-reload value in the TIMx\_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

Since only 1 pulse (Single mode) is needed, a 1 must be written in the OPM bit in the TIMx\_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0). When OPM bit in the TIMx\_CR1 register is set to '0', so the Repetitive Mode is selected.

#### Particular case: OCx fast enable

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay  $t_{DELAY}$  min we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set in the TIMx\_CCMRx register. Then OCxRef (and OCx) are forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

### 24.3.12 Retriggerable one pulse mode (TIM12 only)

This mode allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length, but with the following differences with non-retriggerable one pulse mode described in [Section 24.3.11: One-pulse mode](#):

- The pulse starts as soon as the trigger occurs (no programmable delay)
- The pulse is extended if a new trigger occurs before the previous one is completed

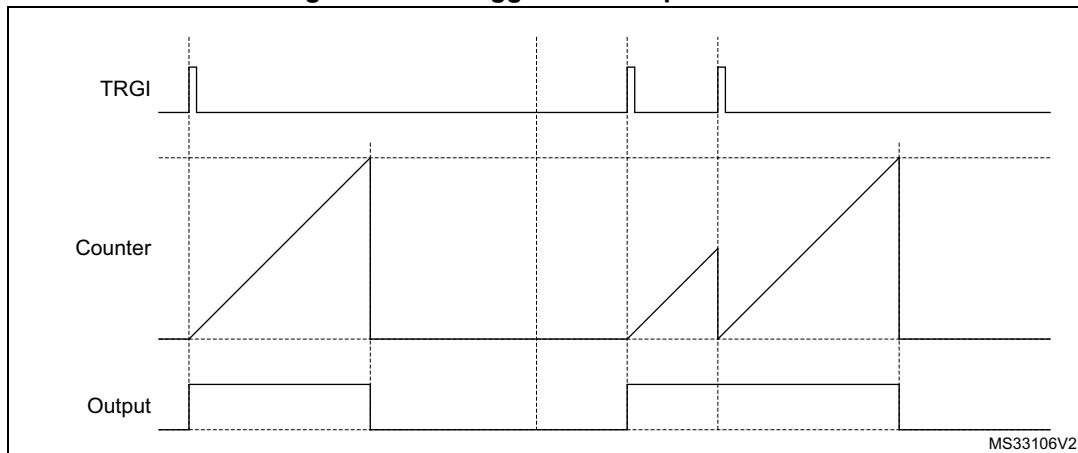
The timer must be in Slave mode, with the bits SMS[3:0] = '1000' (Combined Reset + trigger mode) in the TIMx\_SMCR register, and the OCxM[3:0] bits set to '1000' or '1001' for retrigerrable OPM mode 1 or 2.

If the timer is configured in up-counting mode, the corresponding CCRx must be set to 0 (the ARR register sets the pulse length). If the timer is configured in down-counting mode, CCRx must be above or equal to ARR.

**Note:** *The OCxM[3:0] and SMS[3:0] bit fields are split into two parts for compatibility reasons, the most significant bit are not contiguous with the 3 least significant ones.*

*This mode must not be used with center-aligned PWM modes. It is mandatory to have CMS[1:0] = 00 in TIMx\_CR1.*

Figure 274. Retriggerable one pulse mode



#### 24.3.13 UIF bit remapping

The IUFREMAP bit in the TIMx\_CR1 register forces a continuous copy of the Update Interrupt Flag UIF into bit 31 of the timer counter register (TIMxCNT[31]). This allows to atomically read both the counter value and a potential roll-over condition signaled by the UIFCPY flag. In particular cases, it can ease the calculations by avoiding race conditions caused for instance by a processing shared between a background task (counter reading) and an interrupt (Update Interrupt).

There is no latency between the assertions of the UIF and UIFCPY flags.

#### 24.3.14 TIM9/TIM12 external trigger synchronization

The TIM9/TIM12 timers can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

##### Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

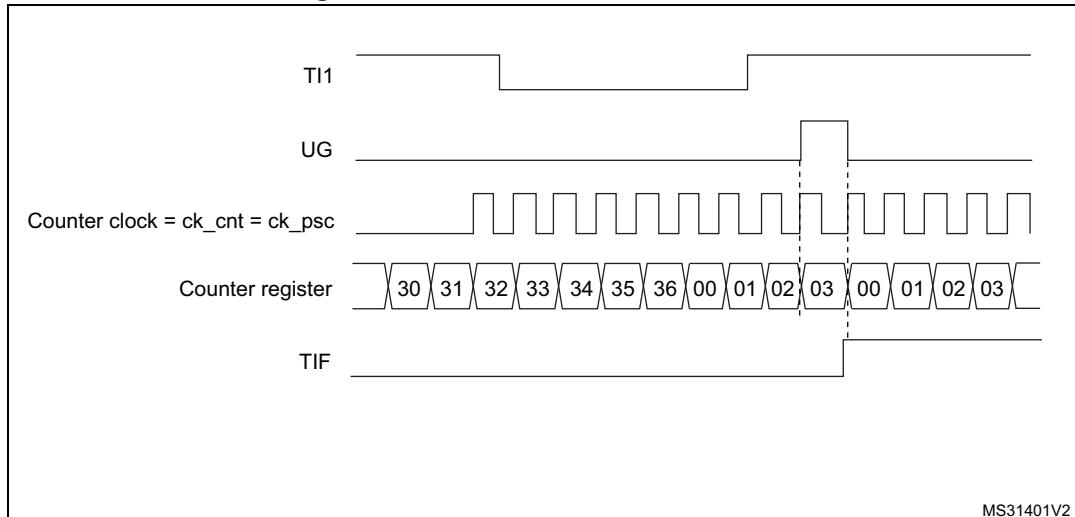
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

1. Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F='0000'). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S = '01' in the TIMx\_CCMR1 register. 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  $\text{TIMx\_ARR}=0x36$ . 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 275. Control circuit in reset mode**



### Slave mode: Gated mode

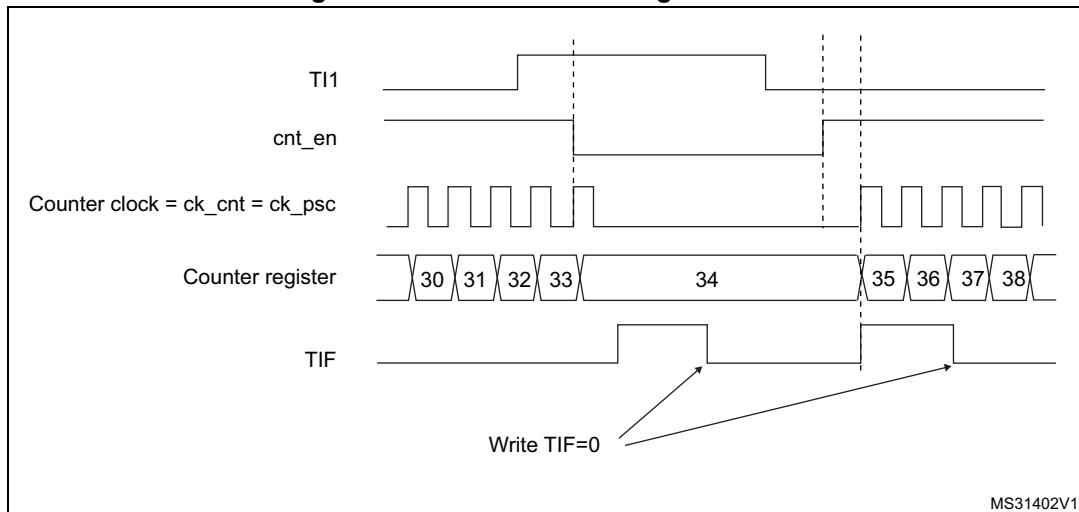
The counter can be enabled depending on the level of a selected input.

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 do not need any filter, so we keep IC1F='0000'). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S='01' in  $\text{TIMx\_CCMR1}$  register. Program CC1P='1' and CC1NP='0' in  $\text{TIMx\_CCER}$  register to validate the polarity (and detect low level only).
2. Configure the timer in gated mode by writing SMS='101' in  $\text{TIMx\_SMCR}$  register. Select TI1 as the input source by writing TS='101' in  $\text{TIMx\_SMCR}$  register.
3. Enable the counter by writing CEN='1' in the  $\text{TIMx\_CR1}$  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  $\text{TIMx\_SR}$  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 276. Control circuit in gated mode****Slave mode: Trigger mode**

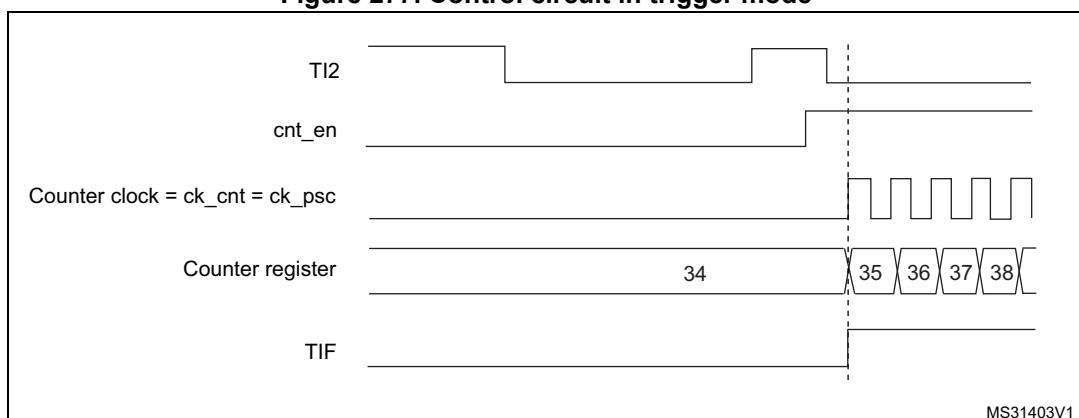
The counter can start in response to an event on a selected input.

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 do not need any filter, so we keep IC2F='0000'). The capture prescaler is not used for triggering, so it does not need to be configured. The CC2S bits are configured to select the input capture source only, CC2S='01' in TIMx\_CCMR1 register. 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 277. Control circuit in trigger mode**

#### 24.3.15 Slave mode – combined reset + trigger mode

In this case, a rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers, and starts the counter.

This mode is used for one-pulse mode.

### 24.3.16 Timer synchronization (TIM9/TIM12)

The TIM timers are linked together internally for timer synchronization or chaining. Refer to [Section 23.3.19: Timer synchronization](#) for details.

**Note:** *The clock of the slave timer must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

### 24.3.17 Using timer output as trigger for other timers (TIM10/TIM11/TIM13/TIM14)

The timers with one channel only do not feature a master mode. However, the OC1 output signal can be used to trigger some other timers (including timers described in other sections of this document). Check the “TIMx internal trigger connection” table of any TIMx\_SMCR register on the device to identify which timers can be targeted as slave.

The OC1 signal pulse width must be programmed to be at least 2 clock cycles of the destination timer, to make sure the slave timer will detect the trigger.

For instance, if the destination's timer CK\_INT clock is 4 times slower than the source timer, the OC1 pulse width must be 8 clock cycles.

### 24.3.18 Debug mode

When the microcontroller enters debug mode (Cortex®-M7 core halted), the TIMx counter either continues to work normally or stops, depending on DBG\_TIMx\_STOP configuration bit in DBG module. For more details, refer to [Section 40.16.2: Debug support for timers, watchdog, bxCAN and I<sup>2</sup>C](#).

## 24.4 TIM9/TIM12 registers

Refer to [Section 1.2](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be written by half-words (16 bits) or words (32 bits). Read accesses can be done by bytes (8 bits), half-words (16 bits) or words (32 bits).

### 24.4.1 TIMx control register 1 (TIMx\_CR1)(x = 9, 12)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFREMAP	Res.	CKD[1:0]		ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
				rw		rw	rw	rw				rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

0: No remapping. UIF status bit is not copied to TIMx\_CNT register bit 31.

1: Remapping enabled. UIF status bit is copied to TIMx\_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (Tlx),

- 00:  $t_{DTS} = t_{CK\_INT}$
- 01:  $t_{DTS} = 2 \times t_{CK\_INT}$
- 10:  $t_{DTS} = 4 \times t_{CK\_INT}$
- 11: Reserved

Bit 7 **ARPE**: Auto-reload preload enable

- 0: TIMx\_ARR register is not buffered.
- 1: TIMx\_ARR register is buffered.

## Bits 6:4 Reserved, must be kept at reset value.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped on the update event
- 1: Counter stops counting on the next update event (clearing the CEN bit).

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the UEV event sources.

- 0: Any of the following events generates an update interrupt if enabled. These events can be:

- Counter overflow
- Setting the UG bit
- Update generation through the slave mode controller

- 1: Only counter overflow generates an update interrupt if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable update event (UEV) generation.

- 0: UEV enabled. An UEV is generated by one of the following events:

- Counter overflow
- Setting the UG bit

Buffered registers are then loaded with their preload values.

- 1: UEV disabled. No UEV is generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are reinitialized if the UG bit is set.

Bit 0 **CEN**: Counter enable

- 0: Counter disabled
- 1: Counter enabled

CEN is cleared automatically in one-pulse mode, when an update event occurs.

*Note: External clock and gated mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.*

**24.4.2 TIMx slave mode control register (TIMx\_SMCR)(x = 9, 12)**

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	SMS[3]									
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MSM	TS[2:0]				Res.	SMS[2:0]								
								rw	rw	rw	rw		rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **SMS**: Master/Slave mode

0: No action

1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful in order to synchronize several timers on a single external event.

Bits 6:4 **TS[2:0]**: Trigger selection

This bitfield selects the trigger input to be used to synchronize the counter.

000: Internal Trigger 0 (ITR0)

001: Internal Trigger 1 (ITR1)

010: Internal Trigger 2 (ITR2)

011: Internal Trigger 3 (ITR3)

100: TI1 Edge Detector (TI1F\_ED)

101: Filtered Timer Input 1 (TI1FP1)

110: Filtered Timer Input 2 (TI2FP2)

111: Reserved.

See [Table 157: TIMx internal trigger connection on page 830](#) for more details on the meaning of ITRx for each timer.

*Note: These bits must be changed only when they are not used (e.g. when SMS='000) to avoid wrong edge detections at the transition.*

Bit 3 Reserved, must be kept at reset value.

Bits 16, 2, 1, 0 **SMS[3:0]**: Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (refer to ETP bit in TIMx\_SMCR for tim\_etr\_in and CCxP/CCxNP bits in TIMx\_CCER register for tim\_ti1fp1 and tim\_ti2fp2).

0000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

0001: Reserved

0010: Reserved

0011: Reserved

0100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

0101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

0110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

0111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

1000: Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter.

Other codes: reserved.

*Note: The gated mode must not be used if TI1F\_ED is selected as the trigger input (TS='100'). Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.*

*Note: The clock of the slave timer must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

Table 157. TIMx internal trigger connection

Slave TIM	ITR0 (TS = '000')	ITR1 (TS = '001')	ITR2 (TS = '010')	ITR3 (TS = '011')
TIM9	TIM2	TIM3	TIM10_OC	TIM11_OC
TIM12	TIM4	TIM5	TIM13_OC	TIM14_OC

#### 24.4.3 TIMx Interrupt enable register (TIMx\_DIER)(x = 9, 12)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIE	Res.	Res.	Res.	CC2IE	CC1IE	UIE								
									rw				rw	rw	rw

Bits 15:7 Reserved, must be kept at reset value.

Bit 6 **TIE**: Trigger interrupt enable

- 0: Trigger interrupt disabled.
- 1: Trigger interrupt enabled.

Bits 5:3 Reserved, must be kept at reset value.

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable

- 0: CC2 interrupt disabled.
- 1: CC2 interrupt enabled.

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable

- 0: CC1 interrupt disabled.
- 1: CC1 interrupt enabled.

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled.
- 1: Update interrupt enabled.

#### 24.4.4 TIMx status register (TIMx\_SR)(x = 9, 12)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	CC2OF	CC1OF	Res.	Res.	TIF	Res.	Res.	Res.	CC2IF	CC1IF	UIF
					rc_w0	rc_w0			rc_w0				rc_w0	rc_w0	rc_w0

Bits 15:11 Reserved, must be kept at reset value.

Bit 10 **CC2OF**: Capture/compare 2 overcapture flag  
refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on the TRG 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

This flag is set by hardware. It is cleared by software (input capture or output compare mode) or by reading the TIMx\_CCR1 register (input capture mode only).

0: No compare match / No input capture occurred

1: A compare match or an input capture occurred.

**If channel CC1 is configured as output:** this flag is set when the content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register. When the content of TIMx\_CCR1 is greater than the content of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in up-counting and up/down-counting modes) or underflow (in down-counting mode). There are 3 possible options for flag setting in center-aligned mode, refer to the CMS bits in the TIMx\_CR1 register for the full description.

**If channel CC1 is configured as input:** this bit is set when counter value has been captured in TIMx\_CCR1 register (an edge has been detected on IC1, as per the edge sensitivity defined with the CC1P and CC1NP bits setting, in TIMx\_CCER).

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.

### 24.4.5 TIMx event generation register (TIMx\_EGR)(x = 9, 12)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TG	Res.	Res.	Res.	CC2G	CC1G	UG								
									w				w	w	w

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.

#### 24.4.6 TIMx capture/compare mode register 1 (TIMx\_CCMR1)(x = 9, 12)

Address offset: 0x18

Reset value: 0x0000 0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CC<sub>x</sub>S bits. All the other bits in this register have different functions in input and output modes.

##### Input capture 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
IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:12 **IC2F[3:0]**: Input capture 2 filter

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Bits 9:8 **CC2S[1:0]**: Capture/compare 2 selection

This 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[3:0]**: 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<sub>DTS</sub>

0001: f<sub>SAMPLING</sub>=f<sub>CK\_INT</sub>, N=2

0010: f<sub>SAMPLING</sub>=f<sub>CK\_INT</sub>, N=4

0011: f<sub>SAMPLING</sub>=f<sub>CK\_INT</sub>, N=8

0100: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/2, N=6

0101: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/2, N=8

0110: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/4, N=6

0111: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/4, N=8

1000: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/8, N=6

1001: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/8, N=8

1010: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/16, N=5

1011: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/16, N=6

1100: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/16, N=8

1101: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/32, N=5

1110: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/32, N=6

1111: f<sub>SAMPLING</sub>=f<sub>DTS</sub>/32, N=8

Bits 3:2 **IC1PSC[1:0]**: Input capture 1 prescaler

This bitfield defines the ratio of the prescaler acting on the CC1 input (IC1).

The prescaler is reset as soon as CC1E='0' (TIMx\_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input

01: capture is done once every 2 events

10: capture is done once every 4 events

11: capture is done once every 8 events

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: The CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

#### 24.4.7 TIMx capture/compare mode register 1 [alternate] (TIMx\_CCMR1)(x = 9, 12)

Address offset: 0x18

Reset value: 0x0000 0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits in this register have different functions in input and output modes.

##### Output compare mode:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		Res.	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
	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 23:17 Reserved, must be kept at reset value.

Bit 15 Reserved, must be kept at reset value.

Bits 24, 14:12 **OC2M[3:0]**: Output compare 2 mode

Refer to OC1M[3:0] for bit description.

Bit 11 **OC2PE**: Output compare 2 preload enable

Bit 10 **OC2FE**: Output compare 2 fast enable

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode works only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

*Note: The CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx\_CCER).*

Bit 7 Reserved, must be kept at reset value.

Bits 16, 6:4 **OC1M[3:0]**: Output compare 1 mode (refer to bit 16 for OC1M[3])

These bits define the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas the active level of OC1 depends on the CC1P.

0000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs. This mode can be used when the timer serves as a software timebase. When the frozen mode is enabled during timer operation, the output keeps the state (active or inactive) it had before entering the frozen state.

0001: Set channel 1 to active level on match. The OC1REF signal is forced high when the TIMx\_CNT counter matches the capture/compare register 1 (TIMx\_CCR1).

0010: Set channel 1 to inactive level on match. The OC1REF signal is forced low when the TIMx\_CNT counter matches the capture/compare register 1 (TIMx\_CCR1).

0011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1

0100: Force inactive level - OC1REF is forced low

0101: Force active level - OC1REF is forced high

0110: PWM mode 1 - channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else it is inactive

0111: PWM mode 2 - channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else it is active

1000: Retriggerable OPM mode 1 - The channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes active again at the next update.

1001: Retriggerable OPM mode 2 - The channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 2 and the channels becomes inactive again at the next update.

1010: Reserved,

1011: Reserved,

1100: Combined PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC is the logical OR between OC1REF and OC2REF.

1101: Combined PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC is the logical AND between OC1REF and OC2REF.

1110: Reserved,

1111: Reserved

*Note: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.*

*Note: The OC1M[3] bit is not contiguous, located in bit 16.*

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

Bit 2 **OC1FE**: Output compare 1 fast enable

This bit decreases the latency between a trigger event and a transition on the timer output. It must be used in one-pulse mode (OPM bit set in TIMx\_CR1 register), to have the output pulse starting as soon as possible after the starting trigger.

- 0: CC1 behaves normally depending on the counter and CCR1 values even when the trigger is ON. The minimum delay to activate the CC1 output when an edge occurs on the trigger input is 5 clock cycles
- 1: An active edge on the trigger input acts like a compare match on the CC1 output. Then, OC is set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bitfield defines the direction of the channel (input/output) as well as the used input.

- 00: CC1 channel is configured as output
- 01: CC1 channel is configured as input, IC1 is mapped on TI1
- 10: CC1 channel is configured as input, IC1 is mapped on TI2
- 11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode works only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

*Note: The CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

#### 24.4.8 TIMx capture/compare enable register (TIMx\_CCER)(x = 9, 12)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E							

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.

0: OC1 active high (output mode) / Edge sensitivity selection (input mode, see below)

1: OC1 active low (output mode) / Edge sensitivity selection (input mode, see below)

**When CC1 channel is configured as input**, both CC1NP/CC1P bits select the active polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

CC1NP=0, CC1P=0: 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).

CC1NP=0, CC1P=1: 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).

CC1NP=1, CC1P=1: 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.

CC1NP=1, CC1P=0: This configuration is reserved, it must not be used.

Bit 0 **CC1E**: Capture/Compare 1 output enable.

0: Capture mode disabled / OC1 is not active

1: Capture mode enabled / OC1 signal is output on the corresponding output pin

**Table 158. Output control bit for standard OCx channels**

CCxE bit	OCx output state
0	Output disabled (not driven by the timer: Hi-Z)
1	Output enabled ( $\text{tim\_ocx} = \text{tim\_ocxref} + \text{Polarity}'$ )

**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.

#### 24.4.9 TIMx counter (TIMx\_CNT)(x = 9, 12)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UIF CPY	Res.														
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UIFCPY**: UIF Copy

This bit is a read-only copy of the UIF bit in the TIMx\_ISR register.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

#### 24.4.10 TIMx prescaler (TIMx\_PSC)(x = 9, 12)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded into the active prescaler register at each update event.  
(including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

#### 24.4.11 TIMx auto-reload register (TIMx\_ARR)(x = 9, 12)

Address offset: 0x2C

Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded into the actual auto-reload register.

Refer to the [Section 24.3.1: Time-base unit on page 806](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

#### 24.4.12 TIMx capture/compare register 1 (TIMx\_CCR1)(x = 9, 12)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**If channel CC1 is configured as output:**

CCR1 is the value to be loaded into the actual capture/compare 1 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (OC1PE bit). Else the preload value is copied into the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the TIMx\_CNT counter and signaled on the OC1 output.

**If channel CC1 is configured as input:**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

#### 24.4.13 TIMx capture/compare register 2 (TIMx\_CCR2)(x = 9, 12)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]**: Capture/Compare 2 value

**If channel CC2 is configured as output:**

CCR2 is the value to be loaded into the actual capture/compare 2 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (OC2PE bit). Else the preload value is copied into the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the TIMx\_CNT counter and signalled on the OC2 output.

**If channel CC2 is configured as input:**

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

## 24.4.14 TIM9/TIM12 register map

TIM9/TIM12 registers are mapped as 16-bit addressable registers as described below:

**Table 159. TIM9/TIM12 register map and reset values**

Offset	Register name	Reset value
0x00	<b>TIMx_CR1</b>	31
	Reset value	Res.
0x08	<b>TIMx_SMCR</b>	30
	Reset value	Res.
0x0C	<b>TIMx_DIER</b>	29
	Reset value	Res.
0x10	<b>TIMx_SR</b>	28
	Reset value	Res.
0x14	<b>TIMx_EGR</b>	27
	Reset value	Res.
0x18	<b>TIMx_CCMR1</b> Output Compare mode	26
	Reset value	Res.
	<b>TIMx_CCMR1</b> Input Capture mode	25
	Reset value	Res.
0x1C	Reserved	0 OC2M[3]
0x20	<b>TIMx_CCER</b>	24
	Reset value	Res.
0x24	<b>TIMx_CNT</b>	23
	Reset value	Res.
0x28	<b>TIMx_PSC</b>	22
	Reset value	Res.
0x2C	<b>TIMx_ARR</b>	21
	Reset value	Res.
Reserved		
CNT[15:0]		
PSC[15:0]		
ARR[15:0]		

**Table 159. TIM9/TIM12 register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x30	Reserved																																
0x34	<b>TIMx_CCR1</b>	Res.																															
	Reset value																																
0x38	<b>TIMx_CCR2</b>	Res.																															
	Reset value																																

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 24.5 TIM10/TIM11/TIM13/TIM14 registers

The peripheral registers have to be written by half-words (16 bits) or words (32 bits). Read accesses can be done by bytes (8 bits), half-words (16 bits) or words (32 bits).

### 24.5.1 TIMx control register 1 (TIMx\_CR1)(x = 10, 11, 13, 14)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
				rw		rw	rw	rw				rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

- 0: No remapping. UIF status bit is not copied to TIMx\_CNT register bit 31.
- 1: Remapping enabled. UIF status bit is copied to TIMx\_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (Tlx),

- 00:  $t_{DTS} = t_{CK\_INT}$
- 01:  $t_{DTS} = 2 \times t_{CK\_INT}$
- 10:  $t_{DTS} = 4 \times t_{CK\_INT}$
- 11: Reserved

Bit 7 **ARPE**: Auto-reload preload enable

- 0: TIMx\_ARR register is not buffered
- 1: TIMx\_ARR register is buffered

Bits 6:4 Reserved, must be kept at reset value.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped on the update event
- 1: Counter stops counting on the next update event (clearing the CEN bit).

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the update interrupt (UEV) sources.

0: Any of the following events generate an UEV if enabled:

- Counter overflow
- Setting the UG bit

1: Only counter overflow generates an UEV if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable update interrupt (UEV) event generation.

0: UEV enabled. An UEV is generated by one of the following events:

- Counter overflow
- Setting the UG bit.

Buffered registers are then loaded with their preload values.

1: UEV disabled. No UEV is generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are reinitialized if the UG bit is set.

Bit 0 **CEN**: Counter enable

0: Counter disabled

1: Counter enabled

*Note: External clock and gated mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.*

**24.5.2 TIMx Interrupt enable register (TIMx\_DIER)(x = 10, 11, 13, 14)**

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1IE	UIE													

Bits 15:2 Reserved, must be kept at reset value.

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable

0: CC1 interrupt disabled

1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable

0: Update interrupt disabled

1: Update interrupt enabled

**24.5.3 TIMx status register (TIMx\_SR)(x = 10, 11, 13, 14)**

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CC1OF	Res.	CC1IF	UIF						

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bits 8:2 Reserved, must be kept at reset value.

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

This flag is set by hardware. It is cleared by software (input capture or output compare mode) or by reading the TIMx\_CCR1 register (input capture mode only).

0: No compare match / No input capture occurred

1: A compare match or an input capture occurred.

**If channel CC1 is configured as output:** this flag is set when the content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register. When the content of TIMx\_CCR1 is greater than the content of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in up-counting and up/down-counting modes) or underflow (in down-counting mode). There are 3 possible options for flag setting in center-aligned mode, refer to the CMS bits in the TIMx\_CR1 register for the full description.

**If channel CC1 is configured as input:** this bit is set when counter value has been captured in TIMx\_CCR1 register (an edge has been detected on IC1, as per the edge sensitivity defined with the CC1P and CC1NP bits setting, in TIMx\_CCER).

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.

#### 24.5.4 TIMx event generation register (TIMx\_EGR)(x = 10, 11, 13, 14)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1G	UG													

Bits 15:2 Reserved, must be kept at reset value.

**Bit 1 CC1G:** Capture/compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

**If channel CC1 is configured as output:**

CC1IF flag is set, Corresponding interrupt or is sent if enabled.

**If channel CC1 is configured as input:**

The current value of the counter is captured in TIMx\_CCR1 register. The CC1IF flag is set, the corresponding interrupt is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

**Bit 0 UG:** Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared.

## 24.5.5 TIMx capture/compare mode register 1 (TIMx\_CCMR1)(x = 10, 11, 13, 14)

Address offset: 0x18

Reset value: 0x0000 0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode.

**Input capture 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.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]								
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:4 **IC1F[3:0]**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N 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[1:0]**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).

The prescaler is reset as soon as CC1E='0' (TIMx\_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input

01: capture is done once every 2 events

10: capture is done once every 4 events

11: capture is done once every 8 events

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: Reserved

11: Reserved

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

#### 24.5.6 TIMx capture/compare mode register 1 [alternate] (TIMx\_CCMR1)(x = 10, 11, 13, 14)

Address offset: 0x18

Reset value: 0x0000 0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the

corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode.

#### Output compare mode:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	OC1M[3]									
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OC1M[2:0]	OC1PE	OC1FE	CC1S[1:0]											
									rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

Bits 15:7 Reserved, must be kept at reset value.

Bits 16, 6:4 **OC1M[3:0]**: Output compare 1 mode (refer to bit 16 for OC1M[3])

These bits define the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas OC1 active level depends on CC1P bit.

0000: Frozen. The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs. This mode can be used when the timer serves as a software timebase. When the frozen mode is enabled during timer operation, the output keeps the state (active or inactive) it had before entering the frozen state.

0001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

0010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

0011: Toggle - OC1REF toggles when TIMx\_CNT = TIMx\_CCR1.

0100: Force inactive level - OC1REF is forced low.

0101: Force active level - OC1REF is forced high.

0110: PWM mode 1 - Channel 1 is active as long as TIMx\_CNT < TIMx\_CCR1 else inactive.

0111: PWM mode 2 - Channel 1 is inactive as long as TIMx\_CNT < TIMx\_CCR1 else active

Others: Reserved

*Note: In PWM mode 1 or 2, the OCREF level changes when the result of the comparison changes or when the output compare mode switches from frozen to PWM mode.*

*Note: The OC1M[3] bit is not contiguous, located in bit 16.*

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.

Bit 2 **OC1FE**: Output compare 1 fast enable

This bit decreases the latency between a trigger event and a transition on the timer output. It must be used in one-pulse mode (OPM bit set in TIMx\_CR1 register), to have the output pulse starting as soon as possible after the starting trigger.

- 0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.
- 1: An active edge on the trigger input acts like a compare match on CC1 output. OC is then set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

- 00: CC1 channel is configured as output.
- 01: CC1 channel is configured as input, IC1 is mapped on TI1.
- 10: Reserved.
- 11: Reserved.

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

#### 24.5.7 TIMx capture/compare enable register (TIMx\_CCER)(x = 10, 11, 13, 14)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1NP	Res.	CC1P	CC1E											
												rw		rw	rw

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.

0: OC1 active high (output mode) / Edge sensitivity selection (input mode, see below)

1: OC1 active low (output mode) / Edge sensitivity selection (input mode, see below)

**When CC1 channel is configured as input**, both CC1NP/CC1P bits select the active polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

CC1NP=0, CC1P=0: 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).

CC1NP=0, CC1P=1: 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).

CC1NP=1, CC1P=1: 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.

CC1NP=1, CC1P=0:This configuration is reserved, it must not be used.

Bit 0 **CC1E**: Capture/Compare 1 output enable.

0: Capture mode disabled / OC1 is not active

1: Capture mode enabled / OC1 signal is output on the corresponding output pin

**Table 160. Output control bit for standard OCx channels**

CCxE bit	OCx output state
0	Output disabled (not driven by the timer: Hi-Z)
1	Output enabled (tim_ocx = tim_ocxref + Polarity)

**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.

#### 24.5.8 TIMx counter (TIMx\_CNT)(x = 10, 11, 13, 14)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UIF CPY	Res.														
rw															
15      14      13      12      11      10      9      8      7      6      5      4      3      2      1      0															
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UIFCPY**: UIF Copy

This bit is a read-only copy of the UIF bit in the TIMx\_ISR register.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

### 24.5.9 TIMx prescaler (TIMx\_PSC)(x = 10, 11, 13, 14)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event.

(including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

### 24.5.10 TIMx auto-reload register (TIMx\_ARR)(x = 10, 11, 13, 14)

Address offset: 0x2C

Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to [Section 24.3.1: Time-base unit on page 806](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 24.5.11 TIMx capture/compare register 1 (TIMx\_CCR1)(x = 10, 11, 13, 14)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**If channel CC1 is configured as output:**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**If channel CC1 is configured as input:**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

#### 24.5.12 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	TI1_RMP[1:0]
Res.	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: TIM11 Channel1 is connected to GPIO (refer to the Alternate function mapping)

01: SPDIFRX\_FRAME\_SYNC is connected to TIM11\_CH1 to measure the clock drift of received SPDIF frames.

10: HSE internal clock (1MHz for RTC) is connected to TIM11\_CH1 input for measurement purposes

11: MCO1 is connected to TIM11\_CH1 input

#### 24.5.13 TIM10/TIM11/TIM13/TIM14 register map

TIMx registers are mapped as 16-bit addressable registers as described in the tables below:

**Table 161. TIM10/TIM11/TIM13/TIM14 register map and reset values**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Res.	UIFREMA	CKD [1:0]	ARPE	Res.	Res.	Res.	OPM	URS	UDS	CEN																					
	Reset value																				0	0	0	0			0	0	0	0			
0x04 to 0x08	Reserved																																
0x0C	TIMx_DIER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CC1IE	UIE																						
	Reset value																												0	0			

**Table 161. TIM10/TIM11/TIM13/TIM14 register map and reset values (continued)**

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 25 Basic timers (TIM6/TIM7)

### 25.1 TIM6/TIM7 introduction

The basic timers TIM6 and TIM7 consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used as generic timers for time base generation but they are also specifically used to drive the digital-to-analog converter (DAC). In fact, the timers are internally connected to the DAC and are able to drive it through their trigger outputs.

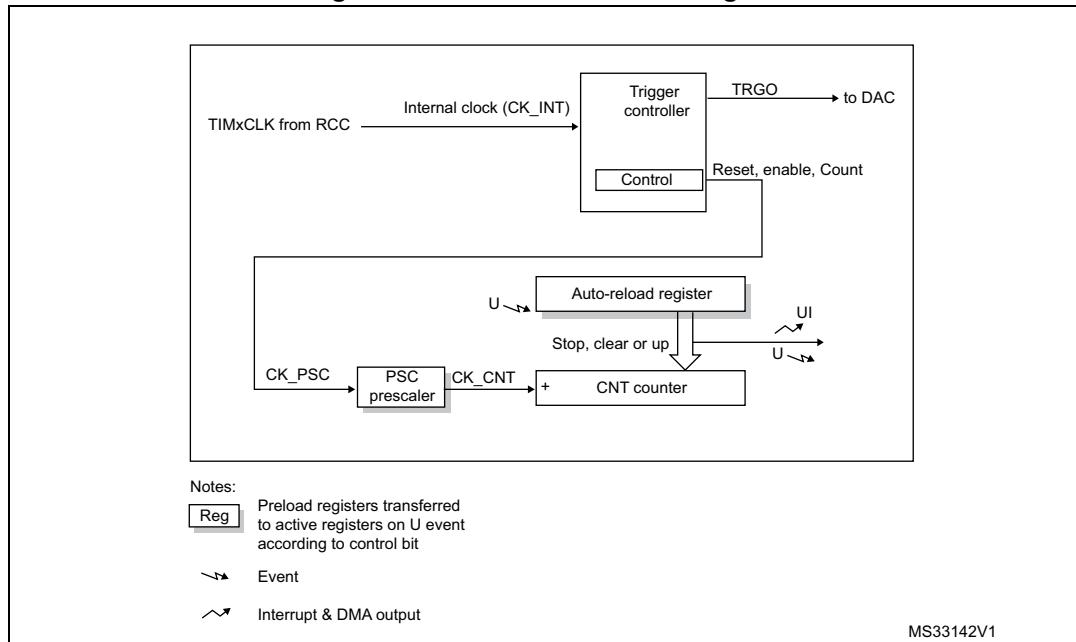
The timers are completely independent, and do not share any resources.

### 25.2 TIM6/TIM7 main features

Basic timer (TIM6/TIM7) features include:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535
- Synchronization circuit to trigger the DAC
- Interrupt/DMA generation on the update event: counter overflow

Figure 278. Basic timer block diagram



## 25.3 TIM6/TIM7 functional description

### 25.3.1 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.

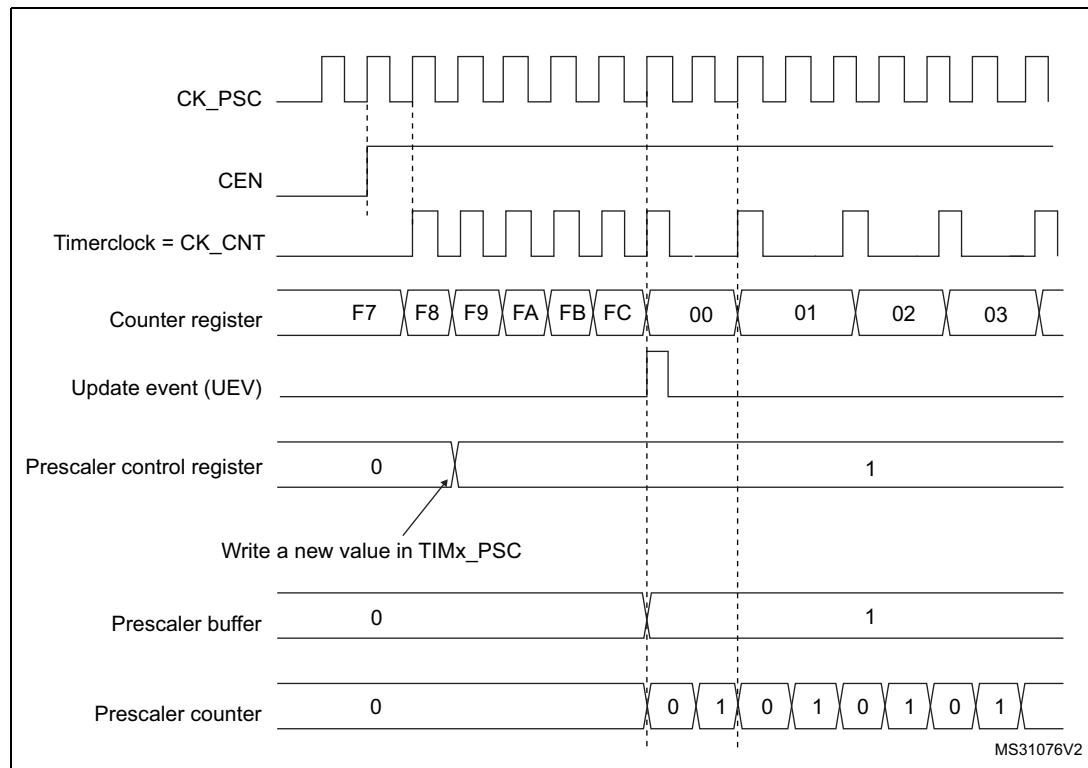
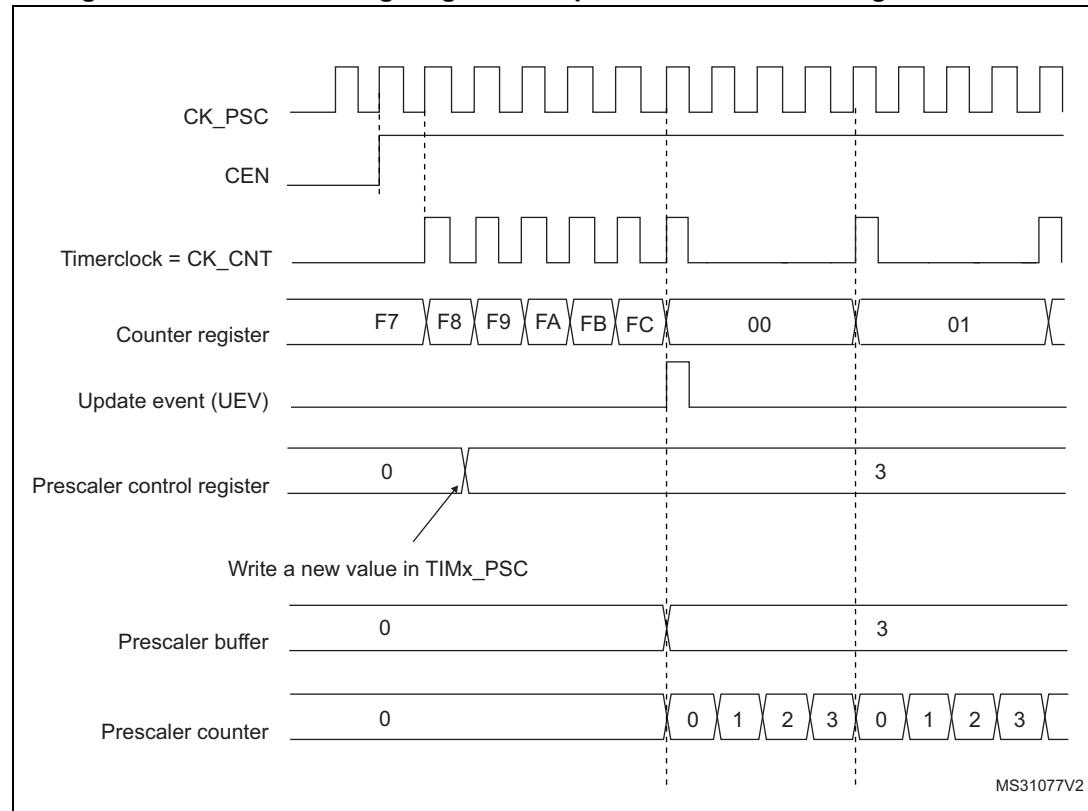
The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in the TIMx\_CR1 register is set.

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 279* and *Figure 280* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

**Figure 279. Counter timing diagram with prescaler division change from 1 to 2****Figure 280. Counter timing diagram with prescaler division change from 1 to 4**

### 25.3.2 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).

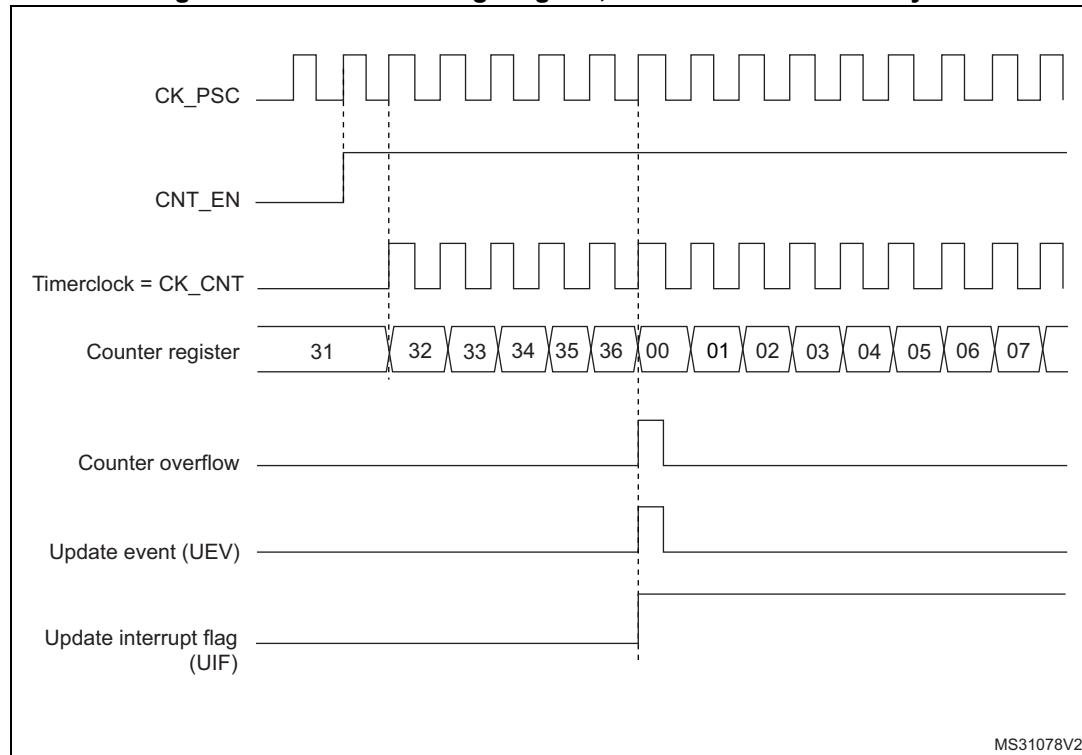
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This avoids updating the shadow registers while writing new values into the preload registers. In this way, no update event occurs until the UDIS bit has been cleared, however, the counter and the prescaler counter 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):

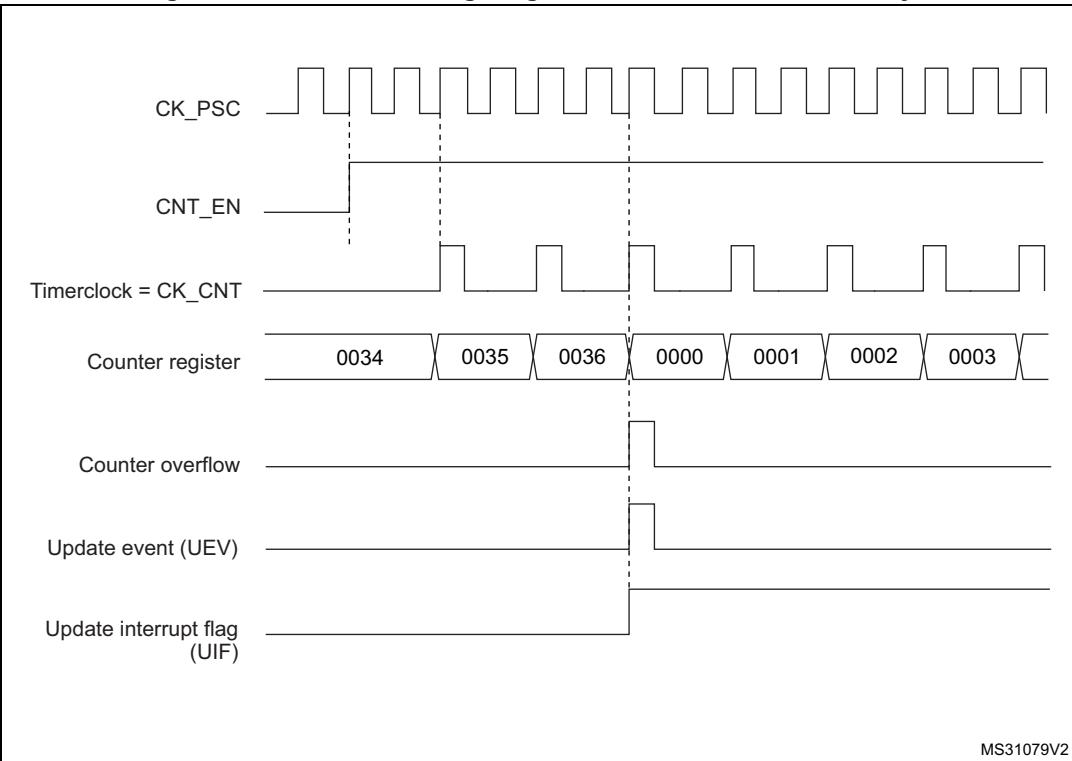
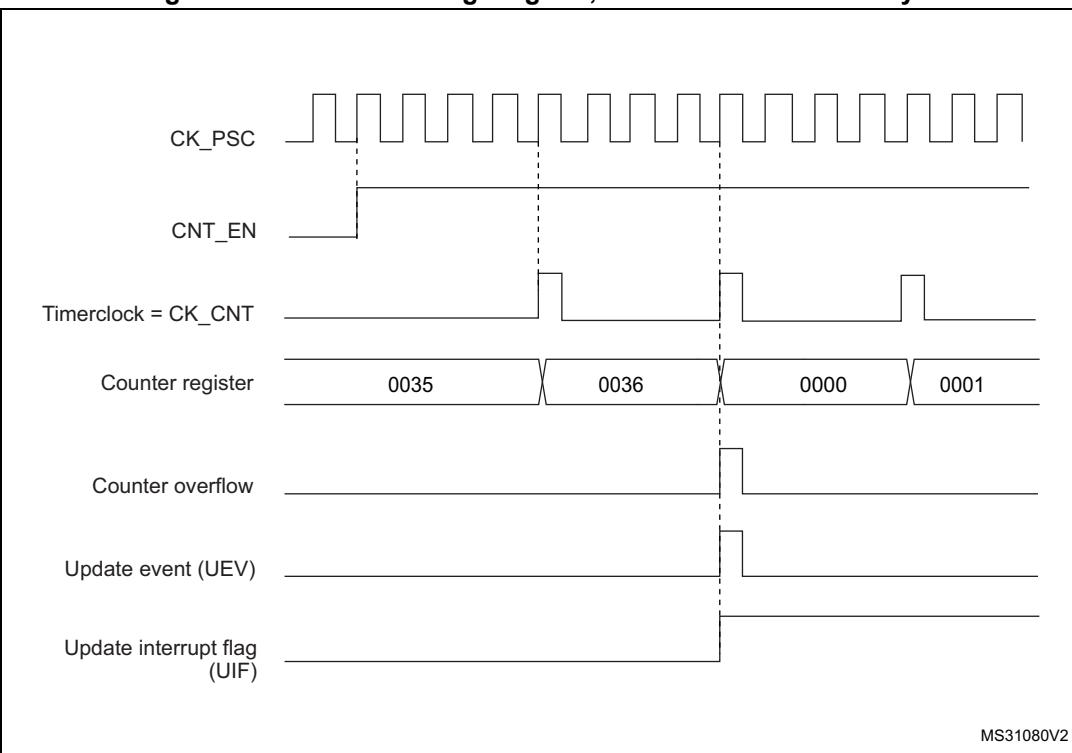
- The buffer of the prescaler is reloaded with the preload value (contents of the TIMx\_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR)

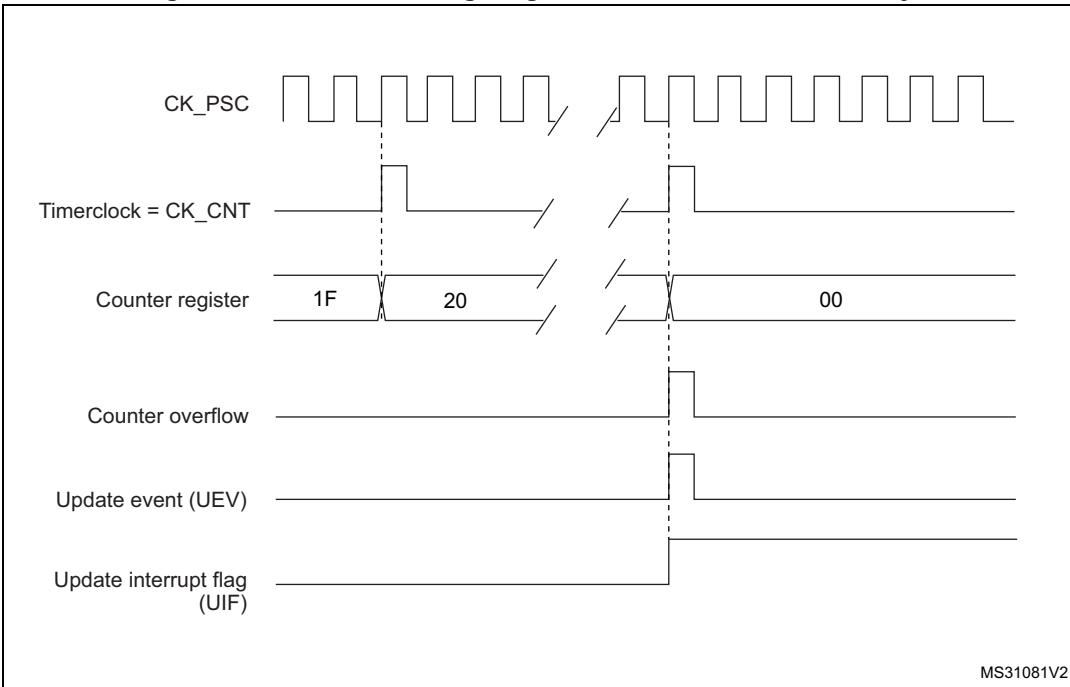
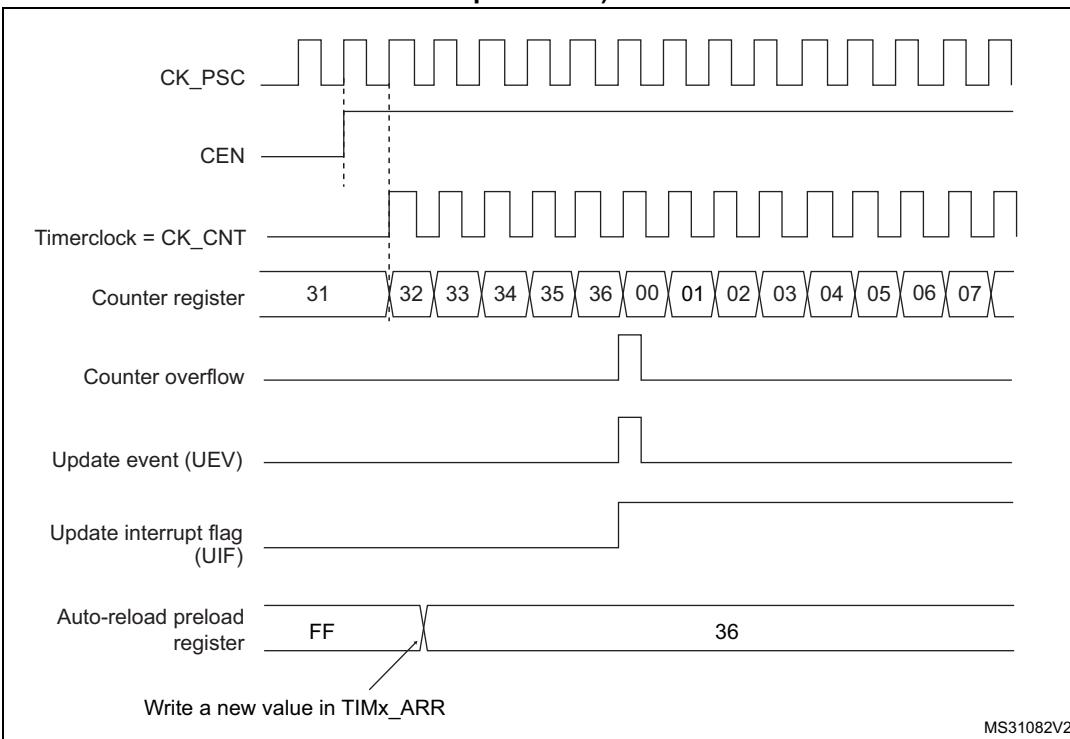
The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR = 0x36.

**Figure 281. Counter timing diagram, internal clock divided by 1**

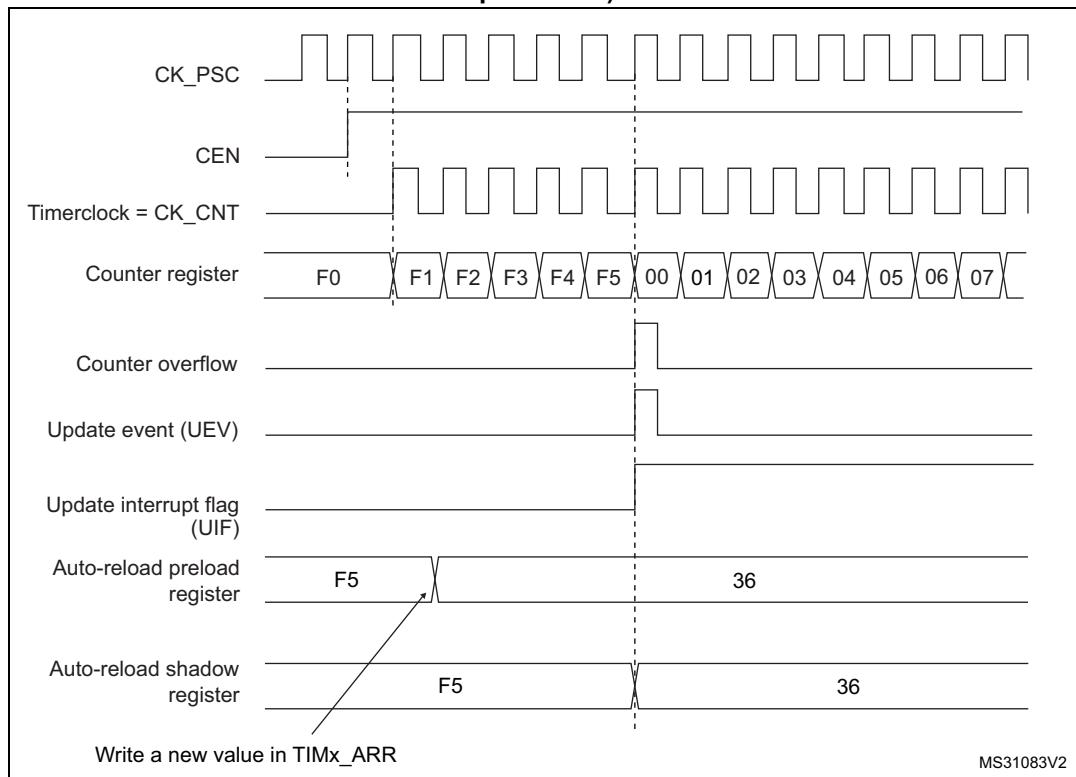


MS31078V2

**Figure 282. Counter timing diagram, internal clock divided by 2****Figure 283. Counter timing diagram, internal clock divided by 4**

**Figure 284. Counter timing diagram, internal clock divided by N****Figure 285. Counter timing diagram, update event when ARPE = 0 (TIMx\_ARR not preloaded)**

**Figure 286. Counter timing diagram, update event when ARPE=1 (TIMx\_ARR preloaded)**



### 25.3.3 UIF bit remapping

The IUFREMAP bit in the TIMx\_CR1 register forces a continuous copy of the Update Interrupt Flag UIF into the timer counter register's bit 31 (TIMxCNT[31]). This allows to atomically read both the counter value and a potential roll-over condition signaled by the UIFCPY flag. In particular cases, it can ease the calculations by avoiding race conditions caused for instance by a processing shared between a background task (counter reading) and an interrupt (Update Interrupt).

There is no latency between the assertions of the UIF and UIFCPY flags.

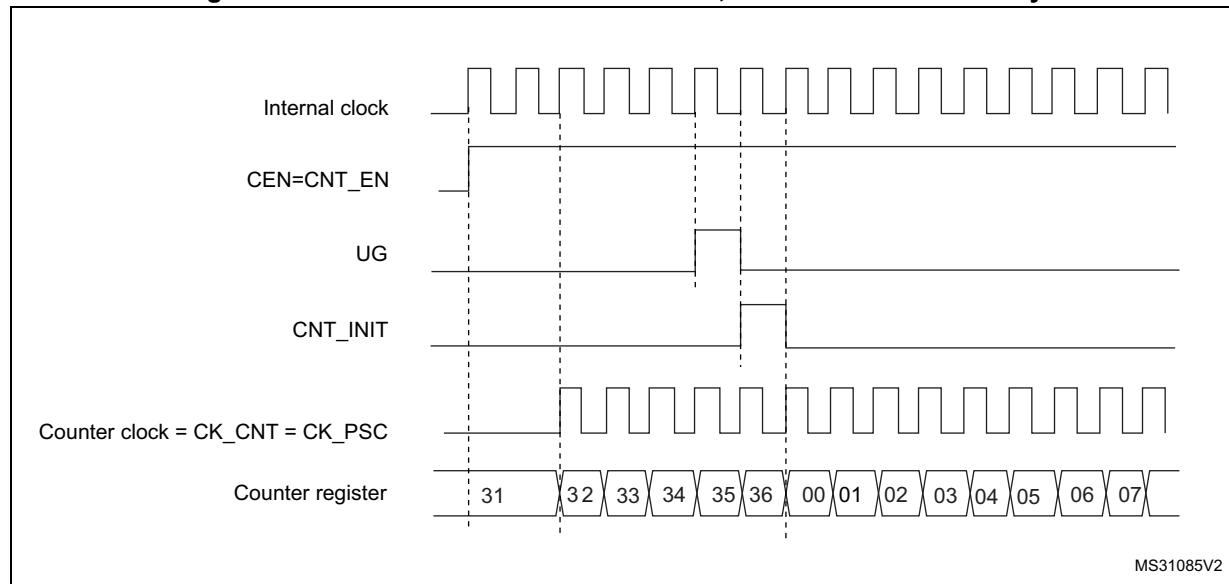
### 25.3.4 Clock source

The counter clock is provided by the Internal clock (CK\_INT) source.

The CEN (in the TIMx\_CR1 register) and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except for UG that remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

[Figure 287](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

Figure 287. Control circuit in normal mode, internal clock divided by 1



### 25.3.5 Debug mode

When the microcontroller enters the debug mode (Cortex®-M7 core - halted), the TIMx counter either continues to work normally or stops, depending on the `DBG_TIMx_STOP` configuration bit in the DBG module. For more details, refer to [Section 40.16.2: Debug support for timers, watchdog, bxCAN and I<sup>2</sup>C](#).

## 25.4 TIM6/TIM7 registers

Refer to [Section 1.2 on page 63](#) for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

### 25.4.1 TIMx control register 1 (TIMx\_CR1)(x = 6 to 7)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	Res.	Res.	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
				rw				rw				rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

0: No remapping. UIF status bit is not copied to TIMx\_CNT register bit 31.

1: Remapping enabled. UIF status bit is copied to TIMx\_CNT register bit 31.

Bits 10:8 Reserved, must be kept at reset value.

Bit 7 **ARPE**: Auto-reload preload enable

- 0: TIMx\_ARR register is not buffered.
- 1: TIMx\_ARR register is buffered.

Bits 6:4 Reserved, must be kept at reset value.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped at update event
- 1: Counter stops counting at the next update event (clearing the CEN bit).

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the UEV event sources.

- 0: Any of the following events generates an update interrupt or DMA request if enabled.  
These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

- 1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

- 0: UEV enabled. The Update (UEV) event is generated by one of the following events:
  - Counter overflow/underflow
  - Setting the UG bit
  - Update generation through the slave mode 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.

### 25.4.2 TIMx control register 2 (TIMx\_CR2)(x = 6 to 7)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MMS[2:0]	Res.	Res.	Res.	Res.	Res.	Res.								
									rw	rw	rw				

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **MMS[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.

*Note: The clock of the slave timer or ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

Bits 3:0 Reserved, must be kept at reset value.

### 25.4.3 TIMx DMA/Interrupt enable register (TIMx\_DIER)(x = 6 to 7)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	UDE	Res.	UIE												
							rw								rw

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled.
- 1: Update DMA request enabled.

Bits 7:1 Reserved, must be kept at reset value.

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled.
- 1: Update interrupt enabled.

### 25.4.4 TIMx status register (TIMx\_SR)(x = 6 to 7)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	UIF														
															rc_w0

Bits 15:1 Reserved, must be kept at reset value.

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition counter value and if UDIS = 0 in the TIMx\_CR1 register.
- When CNT is reinitialized by software using the UG bit in the TIMx\_EGR register, if URS = 0 and UDIS = 0 in the TIMx\_CR1 register.

### 25.4.5 TIMx event generation register (TIMx\_EGR)(x = 6 to 7)

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).

### 25.4.6 TIMx counter (TIMx\_CNT)(x = 6 to 7)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UIF CPY	Res.														
r															
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UIFCPY**: UIF Copy

This bit is a read-only copy of the UIF bit of the TIMx\_ISR register. If the UIFREMAP bit in TIMx\_CR1 is reset, bit 31 is reserved and read as 0.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

### 25.4.7 TIMx prescaler (TIMx\_PSC)(x = 6 to 7)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded into the active prescaler register at each update event.

(including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

### 25.4.8 TIMx auto-reload register (TIMx\_ARR)(x = 6 to 7)

Address offset: 0x2C

Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: Prescaler value

ARR is the value to be loaded into the actual auto-reload register.

Refer to [Section 25.3.1: Time-base unit on page 854](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

## 25.4.9 TIMx register map

TIMx registers are mapped as 16-bit addressable registers as described in the table below:

**Table 162. TIMx register map and reset values**

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<b>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.	Res.	Res.	Res.	
			UICOPY or 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.	Res.	Res.		
			Reset value	Res.																														
0x08																																		
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.	Res.	Res.		
			Reset value	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.	Res.	Res.		
			Reset value	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.	Res.	Res.		
			Reset value	Res.																														
0x18-0x20																																		
0x24	<b>TIMx_CNT</b>	UICOPY or 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	
			Reset value	Res.																														
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.	Res.	Res.	Res.		
			Reset value	Res.																														
0x2C	<b>TIMx_ARR</b>	Res.	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	
			Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 26 Low-power timer (LPTIM)

### 26.1 Introduction

The LPTIM is a 16-bit timer that benefits from the ultimate developments in power consumption reduction. Thanks to its diversity of clock sources, the LPTIM is able to keep running in all power modes except for Standby mode. Given its capability to run even with no internal clock source, the LPTIM can be used as a “Pulse Counter” which can be useful in some applications. Also, the LPTIM capability to wake up the system from low-power modes, makes it suitable to realize “Timeout functions” with extremely low power consumption.

The LPTIM introduces a flexible clock scheme that provides the needed functionalities and performance, while minimizing the power consumption.

### 26.2 LPTIM main features

- 16 bit upcounter
- 3-bit prescaler with 8 possible dividing factors (1,2,4,8,16,32,64,128)
- Selectable clock
  - Internal clock sources: configurable internal clock source (see RCC section)
  - External clock source over LPTIM input (working with no embedded oscillator running, used by Pulse Counter application)
- 16 bit ARR autoreload register
- 16 bit compare register
- Continuous/One-shot mode
- Selectable software/hardware input trigger
- Programmable Digital Glitch filter
- Configurable output: Pulse, PWM
- Configurable I/O polarity
- Encoder mode

## 26.3 LPTIM implementation

*Table 163* describes LPTIM implementation on STM32F75xxx and STM32F74xxx devices: the full set of features is implemented in LPTIM1.

**Table 163. STM32F75xxx and STM32F74xxx LPTIM features**

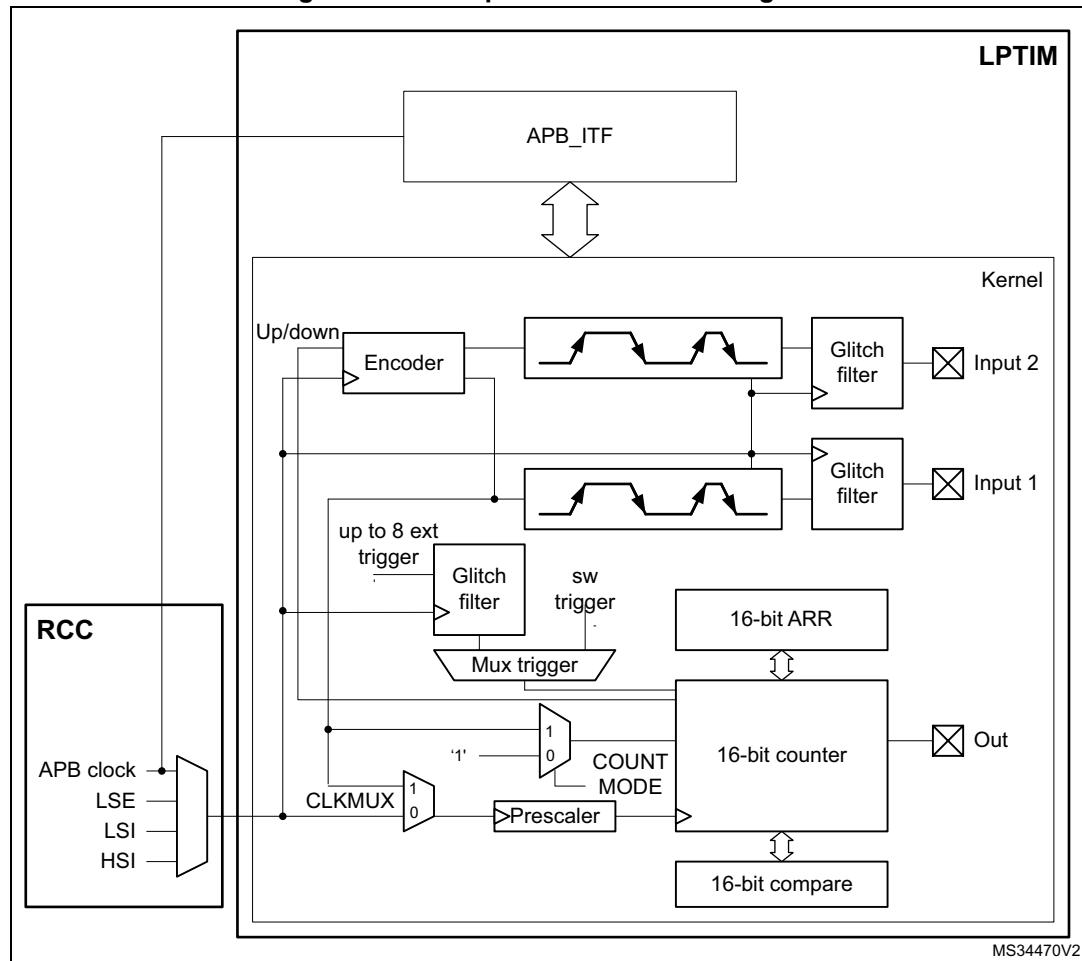
LPTIM modes/features <sup>(1)</sup>	LPTIM1
Encoder mode	X

1. X = supported.

## 26.4 LPTIM functional description

### 26.4.1 LPTIM block diagram

**Figure 288. Low-power timer block diagram**



MS34470V2

### 26.4.2 LPTIM trigger mapping

The LPTIM external trigger connections are detailed hereafter:

**Table 164. LPTIM1 external trigger connection**

TRIGSEL	External trigger
lptim_ext_trig0	GPIO
lptim_ext_trig1	RTC_ALARMA
lptim_ext_trig2	RTC_ALARMB
lptim_ext_trig3	RTC_TAMP1_OUT
lptim_ext_trig4	RTC_TAMP2_OUT
lptim_ext_trig5	RTC_TAMP3_OUT
lptim_ext_trig6	Reserved
lptim_ext_trig7	Reserved

### 26.4.3 LPTIM reset and clocks

The LPTIM can be clocked using several clock sources. It can be clocked using an internal clock signal which can be any configurable internal clock source selectable through the RCC (see RCC section for more details). Also, the LPTIM can be clocked using an external clock signal injected on its external Input1. When clocked with an external clock source, the LPTIM may run in one of these two possible configurations:

- The first configuration is when the LPTIM is clocked by an external signal but in the same time an internal clock signal is provided to the LPTIM from configurable internal clock source (see RCC section).
- The second configuration is when the LPTIM is solely clocked by an external clock source through its external Input1. This configuration is the one used to realize Timeout function or Pulse counter function when all the embedded oscillators are turned off after entering a low-power mode.

Programming the CKSEL and COUNTMODE bits allows controlling whether the LPTIM will use an external clock source or an internal one.

When configured to use an external clock source, the CKPOL bits are used to select the external clock signal active edge. If both edges are configured to be active ones, an internal clock signal should also be provided (first configuration). In this case, the internal clock signal frequency should be at least four times higher than the external clock signal frequency.

### 26.4.4 Glitch filter

The LPTIM inputs, either external (mapped to GPIOs) or internal (mapped on the chip-level to other embedded peripherals), are protected with digital filters that prevent any glitches and noise perturbations to propagate inside the LPTIM. This is in order to prevent spurious counts or triggers.

Before activating the digital filters, an internal clock source should first be provided to the LPTIM. This is necessary to guarantee the proper operation of the filters.

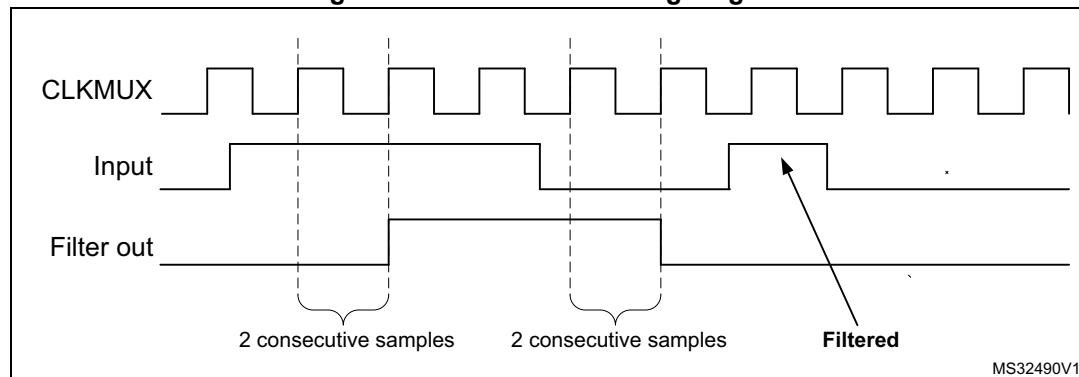
The digital filters are divided into two groups:

- The first group of digital filters protects the LPTIM external inputs. The digital filters sensitivity is controlled by the CKFLT bits
- The second group of digital filters protects the LPTIM internal trigger inputs. The digital filters sensitivity is controlled by the TRGFLT bits.

**Note:** *The digital filters sensitivity is controlled by groups. It is not possible to configure each digital filter sensitivity separately inside the same group.*

The filter sensitivity acts on the number of consecutive equal samples that should be detected on one of the LPTIM inputs to consider a signal level change as a valid transition. [Figure 289](#) shows an example of glitch filter behavior in case of a 2 consecutive samples programmed.

**Figure 289. Glitch filter timing diagram**



**Note:** *In case no internal clock signal is provided, the digital filter must be deactivated by setting the CKFLT and TRGFLT bits to '0'. In that case, an external analog filter may be used to protect the LPTIM external inputs against glitches.*

#### 26.4.5 Prescaler

The LPTIM 16-bit counter is preceded by a configurable power-of-2 prescaler. The prescaler division ratio is controlled by the PRESC[2:0] 3-bit field. The table below lists all the possible division ratios:

**Table 165. Prescaler division ratios**

programming	dividing factor
000	/1
001	/2
010	/4
011	/8
100	/16
101	/32
110	/64
111	/128

### 26.4.6 Trigger multiplexer

The LPTIM counter may be started either by software or after the detection of an active edge on one of the 8 trigger inputs.

TRIGEN[1:0] is used to determine the LPTIM trigger source:

- When TRIGEN[1:0] equals '00', The LPTIM counter is started as soon as one of the CNTSTRT or the SNGSTRT bits is set by software. The three remaining possible values for the TRIGEN[1:0] are used to configure the active edge used by the trigger inputs. The LPTIM counter starts as soon as an active edge is detected.
- When TRIGEN[1:0] is different than '00', TRIGSEL[2:0] is used to select which of the 8 trigger inputs is used to start the counter.

The external triggers are considered asynchronous signals for the LPTIM. So after a trigger detection, a two-counter-clock period latency is needed before the timer starts running due to the synchronization.

If a new trigger event occurs when the timer is already started it will be ignored (unless timeout function is enabled).

**Note:** *The timer must be enabled before setting the SNGSTRT/CNTSTRT bits. Any write on these bits when the timer is disabled will be discarded by hardware.*

**Note:** *When starting the counter by software (TRIGEN[1:0] = 00), there is a delay of 3 kernel clock cycles between the LPTIM\_CR register update (set one of SNGSTRT or CNTSTRT bits) and the effective start of the counter.*

### 26.4.7 Operating mode

The LPTIM features two operating modes:

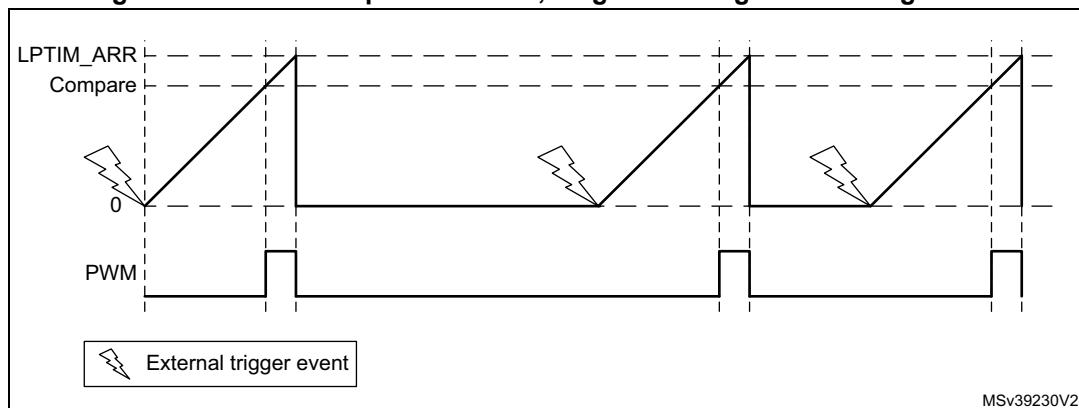
- The Continuous mode: the timer is free running, the timer is started from a trigger event and never stops until the timer is disabled
- One-shot mode: the timer is started from a trigger event and stops when reaching the ARR value.

#### One-shot mode

To enable the one-shot counting, the SNGSTRT bit must be set.

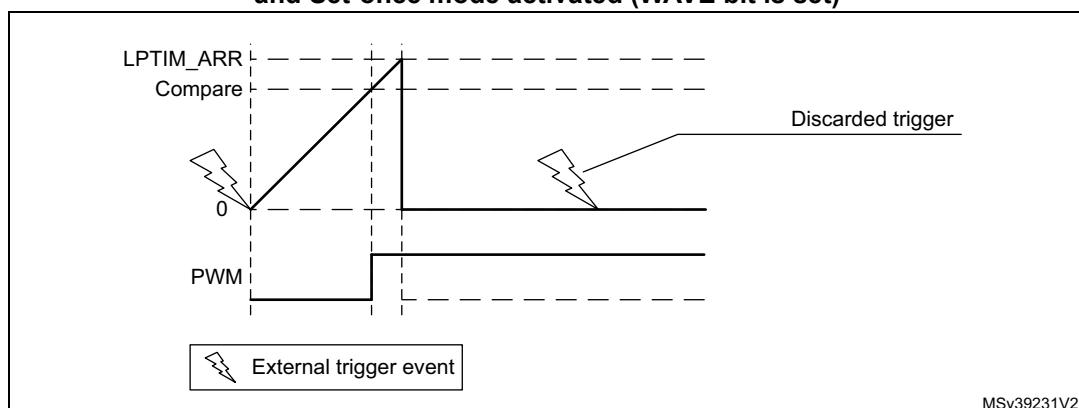
A new trigger event will re-start the timer. Any trigger event occurring after the counter starts and before the counter reaches ARR will be discarded.

In case an external trigger is selected, each external trigger event arriving after the SNGSTRT bit is set, and after the counter register has stopped (contains zero value), will start the counter for a new one-shot counting cycle as shown in [Figure 290](#).

**Figure 290. LPTIM output waveform, single counting mode configuration**

Set-once mode activated:

It should be noted that when the WAVE bit-field in the LPTIM\_CFGR register is set, the Set-once mode is activated. In this case, the counter is only started once following the first trigger, and any subsequent trigger event is discarded as shown in [Figure 291](#).

**Figure 291. LPTIM output waveform, Single counting mode configuration and Set-once mode activated (WAVE bit is set)**

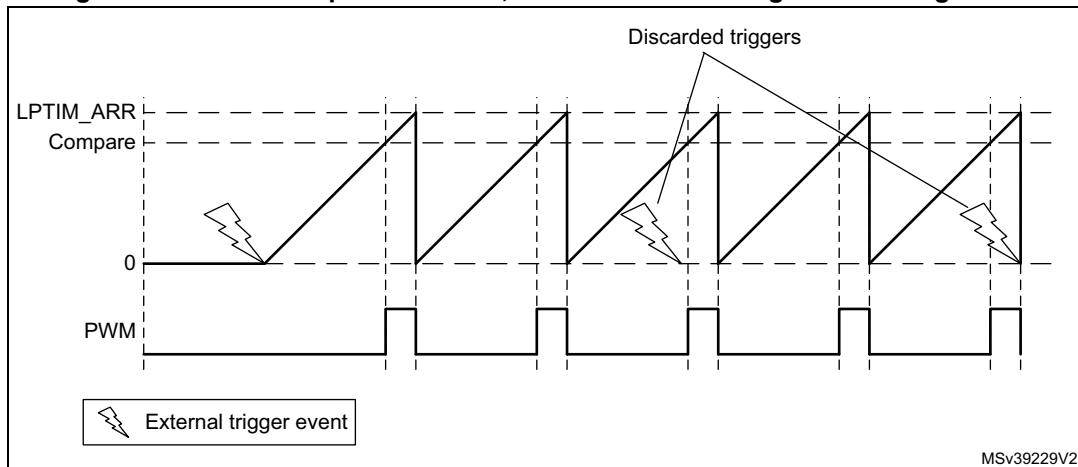
In case of software start (TRIGEN[1:0] = '00'), the SNGSTRT setting will start the counter for one-shot counting.

### Continuous mode

To enable the continuous counting, the CNTSTART bit must be set.

In case an external trigger is selected, an external trigger event arriving after CNTSTART is set will start the counter for continuous counting. Any subsequent external trigger event will be discarded as shown in [Figure 292](#).

In case of software start (TRIGEN[1:0] = '00'), setting CNTSTART will start the counter for continuous counting.

**Figure 292. LPTIM output waveform, Continuous counting mode configuration**

SNGSTRT and CNTSTRT bits can only be set when the timer is enabled (The ENABLE bit is set to '1'). It is possible to change "on the fly" from One-shot mode to Continuous mode.

If the Continuous mode was previously selected, setting SNGSTRT will switch the LPTIM to the One-shot mode. The counter (if active) will stop as soon as it reaches ARR.

If the One-shot mode was previously selected, setting CNTSTRT will switch the LPTIM to the Continuous mode. The counter (if active) will restart as soon as it reaches ARR.

#### 26.4.8 Timeout function

The detection of an active edge on one selected trigger input can be used to reset the LPTIM counter. This feature is controlled through the TIMEOUT bit.

The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.

A low-power timeout function can be realized. The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.

#### 26.4.9 Waveform generation

Two 16-bit registers, the LPTIM\_ARR (autoreload register) and LPTIM\_CMP (compare register), are used to generate several different waveforms on LPTIM output

The timer can generate the following waveforms:

- The PWM mode: the LPTIM output is set as soon as the counter value in LPTIM\_CNT exceeds the compare value in LPTIM\_CMP. The LPTIM output is reset as soon as a match occurs between the LPTIM\_ARR and the LPTIM\_CNT registers.
- The One-pulse mode: the output waveform is similar to the one of the PWM mode for the first pulse, then the output is permanently reset
- The Set-once mode: the output waveform is similar to the One-pulse mode except that the output is kept to the last signal level (depends on the output configured polarity).

The above described modes require that the LPTIM\_ARR register value be strictly greater than the LPTIM\_CMP register value.

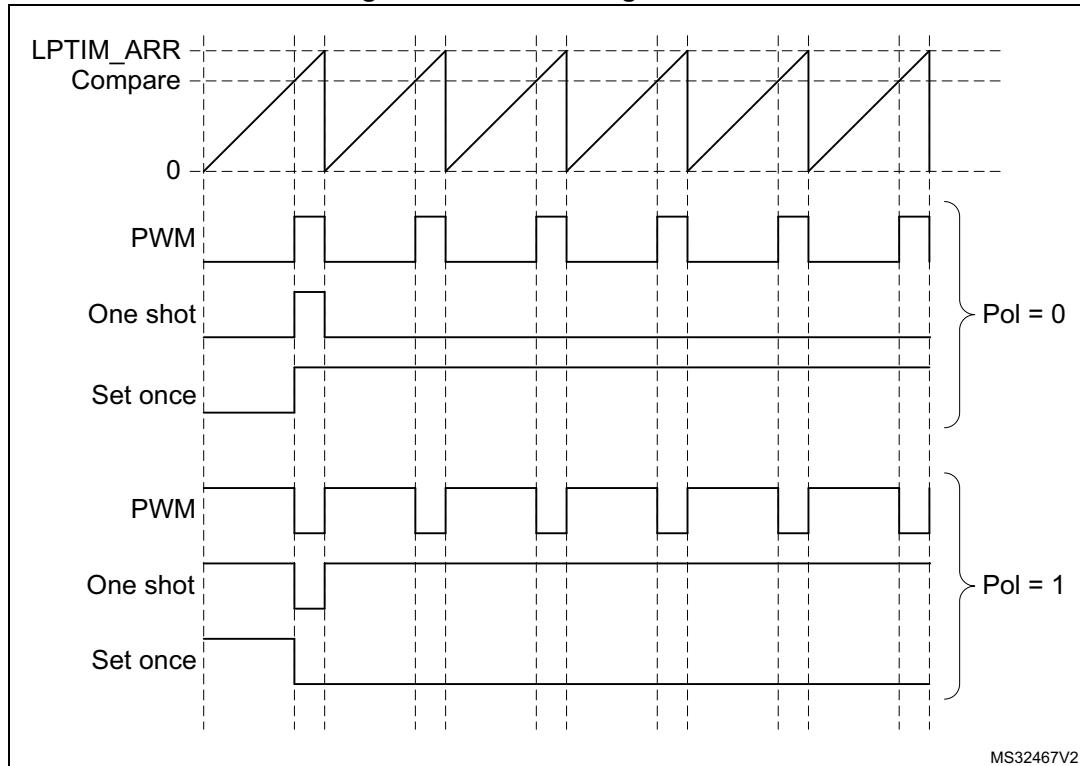
The LPTIM output waveform can be configured through the WAVE bit as follow:

- Resetting the WAVE bit to '0' forces the LPTIM to generate either a PWM waveform or a One pulse waveform depending on which bit is set: CNTSTART or SNGSTART.
- Setting the WAVE bit to '1' forces the LPTIM to generate a Set-once mode waveform.

The WAVPOL bit controls the LPTIM output polarity. The change takes effect immediately, so the output default value will change immediately after the polarity is re-configured, even before the timer is enabled.

Signals with frequencies up to the LPTIM clock frequency divided by 2 can be generated. [Figure 293](#) below shows the three possible waveforms that can be generated on the LPTIM output. Also, it shows the effect of the polarity change using the WAVPOL bit.

**Figure 293. Waveform generation**



#### 26.4.10 Register update

The LPTIM\_ARR register and LPTIM\_CMP register are updated immediately after the APB bus write operation, or at the end of the current period if the timer is already started.

The PRELOAD bit controls how the LPTIM\_ARR and the LPTIM\_CMP registers are updated:

- When the PRELOAD bit is reset to '0', the LPTIM\_ARR and the LPTIM\_CMP registers are immediately updated after any write access.
- When the PRELOAD bit is set to '1', the LPTIM\_ARR and the LPTIM\_CMP registers are updated at the end of the current period, if the timer has been already started.

The LPTIM APB interface and the LPTIM kernel logic use different clocks, so there is some latency between the APB write and the moment when these values are available to the

counter comparator. Within this latency period, any additional write into these registers must be avoided.

The ARROK flag and the CMPOK flag in the LPTIM\_ISR register indicate when the write operation is completed to respectively the LPTIM\_ARR register and the LPTIM\_CMP register.

After a write to the LPTIM\_ARR register or the LPTIM\_CMP register, a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before respectively the ARROK flag or the CMPOK flag be set, will lead to unpredictable results.

#### 26.4.11 Counter mode

The LPTIM counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. The CKSEL and COUNTMODE bits control which source will be used for updating the counter.

In case the LPTIM is configured to count external events on Input1, the counter can be updated following a rising edge, falling edge or both edges depending on the value written to the CKPOL[1:0] bits.

The count modes below can be selected, depending on CKSEL and COUNTMODE values:

- CKSEL = 0: the LPTIM is clocked by an internal clock source
  - COUNTMODE = 0  
The LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be updated following each internal clock pulse.
  - COUNTMODE = 1  
The LPTIM external Input1 is sampled with the internal clock provided to the LPTIM.

Consequently, in order not to miss any event, the frequency of the changes on the external Input1 signal should never exceed the frequency of the internal clock provided to the LPTIM. Also, the internal clock provided to the LPTIM must not be prescaled (PRESC[2:0] = 000).

- CKSEL = 1: the LPTIM is clocked by an external clock source  
COUNTMODE value is don't care.

In this configuration, the LPTIM has no need for an internal clock source (except if the glitch filters are enabled). The signal injected on the LPTIM external Input1 is used as system clock for the LPTIM. This configuration is suitable for operation modes where no embedded oscillator is enabled.

For this configuration, the LPTIM counter can be updated either on rising edges or falling edges of the input1 clock signal but not on both rising and falling edges.

Since the signal injected on the LPTIM external Input1 is also used to clock the LPTIM kernel logic, there is some initial latency (after the LPTIM is enabled) before the counter is incremented. More precisely, the first five active edges on the LPTIM external Input1 (after LPTIM is enable) are lost.

#### 26.4.12 Timer enable

The ENABLE bit located in the LPTIM\_CR register is used to enable/disable the LPTIM kernel logic. After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM is actually enabled.

The LPTIM\_CFGR and LPTIM\_IER registers must be modified only when the LPTIM is disabled.

### 26.4.13 Encoder mode

This mode allows handling signals from quadrature encoders used to detect angular position of rotary elements. Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value programmed into the LPTIM\_ARR register (0 up to ARR or ARR down to 0 depending on the direction). Therefore LPTIM\_ARR must be configured before starting the counter. From the two external input signals, Input1 and Input2, a clock signal is generated to clock the LPTIM counter. The phase between those two signals determines the counting direction.

The Encoder mode is only available when the LPTIM is clocked by an internal clock source. The signals frequency on both Input1 and Input2 inputs must not exceed the LPTIM internal clock frequency divided by 4. This is mandatory in order to guarantee a proper operation of the LPTIM.

Direction change is signalized by the two Down and Up flags in the LPTIM\_ISR register. Also, an interrupt can be generated for both direction change events if enabled through the DOWNIE bit.

To activate the Encoder mode the ENC bit has to be set to '1'. The LPTIM must first be configured in Continuous mode.

When Encoder mode is active, the LPTIM counter is modified automatically following the speed and the direction of the incremental encoder. Therefore, its content always represents the encoder's position. The count direction, signaled by the Up and Down flags, correspond to the rotation direction of the encoder rotor.

According to the edge sensitivity configured using the CKPOL[1:0] bits, different counting scenarios are possible. The following table summarizes the possible combinations, assuming that Input1 and Input2 do not switch at the same time.

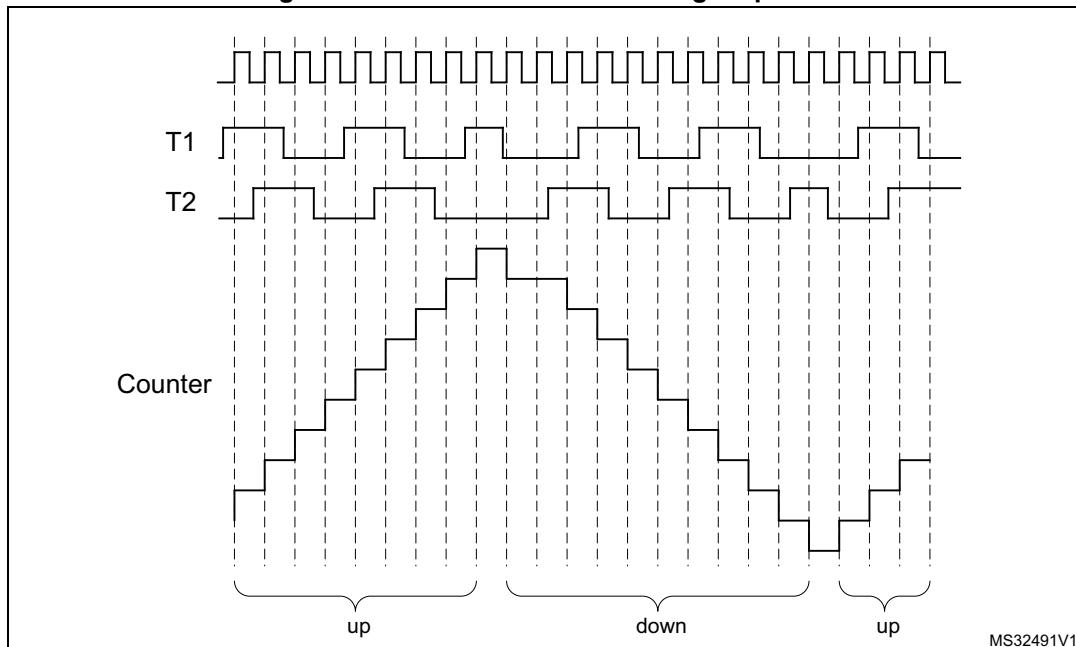
**Table 166. Encoder counting scenarios**

Active edge	Level on opposite signal (Input1 for Input2, Input2 for Input1)	Input1 signal		Input2 signal	
		Rising	Falling	Rising	Falling
Rising Edge	High	Down	No count	Up	No count
	Low	Up	No count	Down	No count
Falling Edge	High	No count	Up	No count	Down
	Low	No count	Down	No count	Up
Both Edges	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

The following figure shows a counting sequence for Encoder mode where both-edge sensitivity is configured.

**Caution:** In this mode the LPTIM must be clocked by an internal clock source, so the CKSEL bit must be maintained to its reset value which is equal to '0'. Also, the prescaler division ratio must be equal to its reset value which is 1 (PRESC[2:0] bits must be '000').

Figure 294. Encoder mode counting sequence



#### 26.4.14 Debug mode

When the microcontroller enters debug mode (core halted), the LPTIM counter either continues to work normally or stops, depending on the DBG\_LPTIM\_STOP configuration bit in the DBG module.

### 26.5 LPTIM low-power modes

Table 167. Effect of low-power modes on the LPTIM

Mode	Description
Sleep	No effect. LPTIM interrupts cause the device to exit Sleep mode.
Stop	The LPTIM peripheral is active when it is clocked by LSE or LSI. LPTIM interrupts cause the device to exit Stop mode
Standby	The LPTIM peripheral is powered down and must be reinitialized after exiting Standby mode.

## 26.6 LPTIM interrupts

The following events generate an interrupt/wake-up event, if they are enabled through the LPTIM\_IER register:

- Compare match
- Auto-reload match (whatever the direction if encoder mode)
- External trigger event
- Autoreload register write completed
- Compare register write completed
- Direction change (encoder mode), programmable (up / down / both).

**Note:** *If any bit in the LPTIM\_IER register (Interrupt Enable Register) is set after that its corresponding flag in the LPTIM\_ISR register (Status Register) is set, the interrupt is not asserted.*

**Table 168. Interrupt events**

Interrupt event	Description
Compare match	Interrupt flag is raised when the content of the Counter register (LPTIM_CNT) matches the content of the compare register (LPTIM_CMP).
Auto-reload match	Interrupt flag is raised when the content of the Counter register (LPTIM_CNT) matches the content of the Auto-reload register (LPTIM_ARR).
External trigger event	Interrupt flag is raised when an external trigger event is detected
Auto-reload register update OK	Interrupt flag is raised when the write operation to the LPTIM_ARR register is complete.
Compare register update OK	Interrupt flag is raised when the write operation to the LPTIM_CMP register is complete.
Direction change	Used in Encoder mode. Two interrupt flags are embedded to signal direction change: – UP flag signals up-counting direction change – DOWN flag signals down-counting direction change.

## 26.7 LPTIM registers

Refer to [Section 1.2: List of abbreviations for registers](#) for a list of abbreviations used in register descriptions.

The peripheral registers can only be accessed by words (32-bit).

### 26.7.1 LPTIM interrupt and status register (LPTIM\_ISR)

Address offset: 0x000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DOWN	UP	ARR OK	CMP OK	EXT TRIG	ARRM	CMPM								
								r	r	r	r	r	r	r	r

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **DOWN**: Counter direction change up to down

In Encoder mode, DOWN bit is set by hardware to inform application that the counter direction has changed from up to down. DOWN flag can be cleared by writing 1 to the DOWNCF bit in the LPTIM\_ICR register.

*Note: If the LPTIM does not support encoder mode feature, this bit is reserved. Please refer to [Section 26.3: LPTIM implementation](#).*

Bit 5 **UP**: Counter direction change down to up

In Encoder mode, UP bit is set by hardware to inform application that the counter direction has changed from down to up. UP flag can be cleared by writing 1 to the UPCF bit in the LPTIM\_ICR register.

*Note: If the LPTIM does not support encoder mode feature, this bit is reserved. Please refer to [Section 26.3: LPTIM implementation](#).*

Bit 4 **ARROK**: Autoreload register update OK

ARROK is set by hardware to inform application that the APB bus write operation to the LPTIM\_ARR register has been successfully completed. ARROK flag can be cleared by writing 1 to the ARROKCF bit in the LPTIM\_ICR register.

Bit 3 **CMPOK**: Compare register update OK

CMPOK is set by hardware to inform application that the APB bus write operation to the LPTIM\_CMP register has been successfully completed. CMPOK flag can be cleared by writing 1 to the CMPOKCF bit in the LPTIM\_ICR register.

Bit 2 **EXTTRIG**: External trigger edge event

EXTTRIG is set by hardware to inform application that a valid edge on the selected external trigger input has occurred. If the trigger is ignored because the timer has already started, then this flag is not set. EXTTRIG flag can be cleared by writing 1 to the EXTTRIGCF bit in the LPTIM\_ICR register.

Bit 1 **ARRM**: Autoreload match

ARRM is set by hardware to inform application that LPTIM\_CNT register's value reached the LPTIM\_ARR register's value. ARRM flag can be cleared by writing 1 to the ARRMCF bit in the LPTIM\_ICR register.

Bit 0 **CMPM**: Compare match

The CMPM bit is set by hardware to inform application that LPTIM\_CNT register value reached the LPTIM\_CMP register's value. CMPM flag can be cleared by writing 1 to the CMPMCF bit in the LPTIM\_ICR register.

## 26.7.2 LPTIM interrupt clear register (LPTIM\_ICR)

Address offset: 0x004

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DOWNCF	UPCF	ARROKCF	CMPOKCF	EXTTRIGCF	ARRMCF	CMPMCF								
								w	w	w	w	w	w	w	w

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **DOWNCF**: Direction change to down clear flag

Writing 1 to this bit clear the DOWN flag in the LPTIM\_ISR register.

*Note: If the LPTIM does not support encoder mode feature, this bit is reserved. Please refer to Section 26.3: LPTIM implementation.*

Bit 5 **UPCF**: Direction change to UP clear flag

Writing 1 to this bit clear the UP flag in the LPTIM\_ISR register.

*Note: If the LPTIM does not support encoder mode feature, this bit is reserved. Please refer to Section 26.3: LPTIM implementation.*

Bit 4 **ARROKCF**: Autoreload register update OK clear flag

Writing 1 to this bit clears the ARROK flag in the LPTIM\_ISR register

Bit 3 **CMPOKCF**: Compare register update OK clear flag

Writing 1 to this bit clears the CMPOK flag in the LPTIM\_ISR register

Bit 2 **EXTTRIGCF**: External trigger valid edge clear flag

Writing 1 to this bit clears the EXTTRIG flag in the LPTIM\_ISR register

Bit 1 **ARRMCF**: Autoreload match clear flag

Writing 1 to this bit clears the ARRM flag in the LPTIM\_ISR register

Bit 0 **CMPMCF**: Compare match clear flag

Writing 1 to this bit clears the CMPM flag in the LPTIM\_ISR register

## 26.7.3 LPTIM interrupt enable register (LPTIM\_IER)

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DownIE	UPIE	ARROKIE	CMPOKIE	EXTTRIGIE	ARRMIE	CMPMIE								
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **DOWNIE**: Direction change to down Interrupt Enable

- 0: DOWN interrupt disabled
- 1: DOWN interrupt enabled

*Note: If the LPTIM does not support encoder mode feature, this bit is reserved. Please refer to Section 26.3: LPTIM implementation.*

Bit 5 **UPIE**: Direction change to UP Interrupt Enable

- 0: UP interrupt disabled
- 1: UP interrupt enabled

*Note: If the LPTIM does not support encoder mode feature, this bit is reserved. Please refer to Section 26.3: LPTIM implementation.*

Bit 4 **ARROKIE**: Autoreload register update OK Interrupt Enable

- 0: ARROK interrupt disabled
- 1: ARROK interrupt enabled

Bit 3 **CMPOKIE**: Compare register update OK Interrupt Enable

- 0: CMPOK interrupt disabled
- 1: CMPOK interrupt enabled

Bit 2 **EXTTRIGIE**: External trigger valid edge Interrupt Enable

- 0: EXTTRIG interrupt disabled
- 1: EXTTRIG interrupt enabled

Bit 1 **ARRMIE**: Autoreload match Interrupt Enable

- 0: ARRM interrupt disabled
- 1: ARRM interrupt enabled

Bit 0 **CMPMIE**: Compare match Interrupt Enable

- 0: CMPM interrupt disabled
- 1: CMPM interrupt enabled

**Caution:** The LPTIM\_IER register must only be modified when the LPTIM is disabled (ENABLE bit reset to '0')

#### 26.7.4 LPTIM configuration register (LPTIM\_CFGR)

Address offset: 0x00C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	ENC	COUNT MODE	PRELOAD	WAVPOL	WAVE	TIMOUT	TRIGEN[1:0]	Res.	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRIGSEL[2:0]			Res.	PRESC[2:0]			Res.	TRGFLT[1:0]		Res.	CKFLT[1:0]		CKPOL[1:0]		CKSEL
rw	rw	rw		rw	rw	rw		rw	rw		rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 Reserved, must be kept at reset value.

Bits 28:25 Reserved, must be kept at reset value.

**Bit 24 ENC:** Encoder mode enable

The ENC bit controls the Encoder mode

- 0: Encoder mode disabled
- 1: Encoder mode enabled

*Note: If the LPTIM does not support encoder mode feature, this bit is reserved. Please refer to Section 26.3: LPTIM implementation.*

**Bit 23 COUNTMODE:** counter mode enabled

The COUNTMODE bit selects which clock source is used by the LPTIM to clock the counter:

- 0: the counter is incremented following each internal clock pulse
- 1: the counter is incremented following each valid clock pulse on the LPTIM external Input1

**Bit 22 PRELOAD:** Registers update mode

The PRELOAD bit controls the LPTIM\_ARR and the LPTIM\_CMP registers update modality

- 0: Registers are updated after each APB bus write access
- 1: Registers are updated at the end of the current LPTIM period

**Bit 21 WAVPOL:** Waveform shape polarity

The WAVEPOL bit controls the output polarity

- 0: The LPTIM output reflects the compare results between LPTIM\_CNT and LPTIM\_CMP registers
- 1: The LPTIM output reflects the inverse of the compare results between LPTIM\_CNT and LPTIM\_CMP registers

**Bit 20 WAVE:** Waveform shape

The WAVE bit controls the output shape

- 0: Deactivate Set-once mode
- 1: Activate the Set-once mode

**Bit 19 TIMOUT:** Timeout enable

The TIMOUT bit controls the Timeout feature

- 0: A trigger event arriving when the timer is already started will be ignored
- 1: A trigger event arriving when the timer is already started will reset and restart the counter

**Bits 18:17 TRIGEN[1:0]:** Trigger enable and polarity

The TRIGEN bits controls whether the LPTIM counter is started by an external trigger or not. If the external trigger option is selected, three configurations are possible for the trigger active edge:

- 00: software trigger (counting start is initiated by software)
- 01: rising edge is the active edge
- 10: falling edge is the active edge
- 11: both edges are active edges

**Bit 16** Reserved, must be kept at reset value.

**Bits 15:13 TRIGSEL[2:0]: Trigger selector**

The TRIGSEL bits select the trigger source that will serve as a trigger event for the LPTIM among the below 8 available sources:

- 000: lptim\_ext\_trig0
- 001: lptim\_ext\_trig1
- 010: lptim\_ext\_trig2
- 011: lptim\_ext\_trig3
- 100: lptim\_ext\_trig4
- 101: lptim\_ext\_trig5
- 110: lptim\_ext\_trig6
- 111: lptim\_ext\_trig7

See [Section 26.4.2: LPTIM trigger mapping](#) for details.

Bit 12 Reserved, must be kept at reset value.

**Bits 11:9 PRESC[2:0]: Clock prescaler**

The PRESC bits configure the prescaler division factor. It can be one among the following division factors:

- 000: /1
- 001: /2
- 010: /4
- 011: /8
- 100: /16
- 101: /32
- 110: /64
- 111: /128

Bit 8 Reserved, must be kept at reset value.

**Bits 7:6 TRGFLT[1:0]: Configurable digital filter for trigger**

The TRGFLT value sets the number of consecutive equal samples that should be detected when a level change occurs on an internal trigger before it is considered as a valid level transition. An internal clock source must be present to use this feature

- 00: any trigger active level change is considered as a valid trigger
- 01: trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger.
- 10: trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger.
- 11: trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger.

Bit 5 Reserved, must be kept at reset value.

Bits 4:3 **CKFLT[1:0]**: Configurable digital filter for external clock

The CKFLT value sets the number of consecutive equal samples that should be detected when a level change occurs on an external clock signal before it is considered as a valid level transition. An internal clock source must be present to use this feature

00: any external clock signal level change is considered as a valid transition

01: external clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition.

10: external clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition.

11: external clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition.

Bits 2:1 **CKPOL[1:0]**: Clock polarity

If LPTIM is clocked by an external clock source:

When the LPTIM is clocked by an external clock source, CKPOL bits is used to configure the active edge or edges used by the counter:

00:the rising edge is the active edge used for counting.

If the LPTIM is configured in Encoder mode (ENC bit is set), the encoder sub-mode 1 is active.

01:the falling edge is the active edge used for counting

If the LPTIM is configured in Encoder mode (ENC bit is set), the encoder sub-mode 2 is active.

10:both edges are active edges. When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency.

If the LPTIM is configured in Encoder mode (ENC bit is set), the encoder sub-mode 3 is active.

11:not allowed

Refer to [Section 26.4.13: Encoder mode](#) for more details about Encoder mode sub-modes.

Bit 0 **CKSEL**: Clock selector

The CKSEL bit selects which clock source the LPTIM will use:

0: LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)

1: LPTIM is clocked by an external clock source through the LPTIM external Input1

**Caution:** The LPTIM\_CFG register must only be modified when the LPTIM is disabled (ENABLE bit reset to '0').

## 26.7.5 LPTIM control register (LPTIM\_CR)

Address offset: 0x010

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CNT STRT	SNG STRT	ENABLE												

Bits 31:3 Reserved, must be kept at reset value.

**Bit 2 CNTSTRT:** Timer start in Continuous mode

This bit is set by software and cleared by hardware.

In case of software start (TRIGEN[1:0] = '00'), setting this bit starts the LPTIM in Continuous mode.

If the software start is disabled (TRIGEN[1:0] different than '00'), setting this bit starts the timer in Continuous mode as soon as an external trigger is detected.

If this bit is set when a single pulse mode counting is ongoing, then the timer will not stop at the next match between the LPTIM\_ARR and LPTIM\_CNT registers and the LPTIM counter keeps counting in Continuous mode.

This bit can be set only when the LPTIM is enabled. It will be automatically reset by hardware.

**Bit 1 SNGSTRT:** LPTIM start in Single mode

This bit is set by software and cleared by hardware.

In case of software start (TRIGEN[1:0] = '00'), setting this bit starts the LPTIM in single pulse mode.

If the software start is disabled (TRIGEN[1:0] different than '00'), setting this bit starts the LPTIM in single pulse mode as soon as an external trigger is detected.

If this bit is set when the LPTIM is in continuous counting mode, then the LPTIM will stop at the following match between LPTIM\_ARR and LPTIM\_CNT registers.

This bit can only be set when the LPTIM is enabled. It will be automatically reset by hardware.

**Bit 0 ENABLE:** LPTIM enable

The ENABLE bit is set and cleared by software.

0:LPTIM is disabled

1:LPTIM is enabled

## 26.7.6 LPTIM compare register (LPTIM\_CMP)

Address offset: 0x014

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMP[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

**Bits 15:0 CMP[15:0]:** Compare value

CMP is the compare value used by the LPTIM.

**Caution:** The LPTIM\_CMP register must only be modified when the LPTIM is enabled (ENABLE bit set to '1').

### 26.7.7 LPTIM autoreload register (LPTIM\_ARR)

Address offset: 0x018

Reset value: 0x0000 0001

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ARR[15:0]: Auto reload value**

ARR is the autoreload value for the LPTIM.

This value must be strictly greater than the CMP[15:0] value.

**Caution:** The LPTIM\_ARR register must only be modified when the LPTIM is enabled (ENABLE bit set to '1').

### 26.7.8 LPTIM counter register (LPTIM\_CNT)

Address offset: 0x01C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]: Counter value**

When the LPTIM is running with an asynchronous clock, reading the LPTIM\_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.

It should be noted that for a reliable LPTIM\_CNT register read access, two consecutive read accesses must be performed and compared. A read access can be considered reliable when the values of the two consecutive read accesses are equal.

### **26.7.9 LPTIM register map**

The following table summarizes the LPTIM registers.

**Table 169. LPTIM register map and reset values**

Offset	Register name	31	30
0x000	LPTIM_ISR	Res.	Res.
	Reset value	Res.	Res.
0x004	LPTIM_ICR	Res.	Res.
	Reset value	Res.	Res.
0x008	LPTIM_IER	Res.	Res.
	Reset value	Res.	Res.
0x00C	LPTIM_CFGR	Res.	Res.
	Reset value	Res.	Res.
0x010	LPTIM_CR	Res.	Res.
	Reset value	Res.	Res.
0x014	LPTIM_CMP	CMP[15:0]	
	Reset value	Res.	Res.
0x018	LPTIM_ARR	ARR[15:0]	
	Reset value	Res.	Res.
0x01C	LPTIM_CNT	CNT[15:0]	
	Reset value	Res.	Res.
0x020	LPTIM1_OR	Res.	Res.
	Reset value	Res.	Res.

- If LPTIM does not support encoder mode feature, this bit is reserved. Please refer to [Section 26.3: LPTIM implementation](#).

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 27 Independent watchdog (IWDG)

### 27.1 Introduction

The devices feature an embedded watchdog peripheral (IWDG) that offers a combination of high safety level, timing accuracy, and flexibility of use. This peripheral detects and solves malfunctions due to software failure, and triggers a system reset when the counter reaches a given timeout value.

The independent watchdog is clocked by its own dedicated low-speed clock (LSI), and stays active even if the main clock fails.

The IWDG is best suited for applications that require the watchdog to run as a totally independent process outside the main application, but have lower timing accuracy constraints. For further information on the window watchdog, refer to [Section 28: System window watchdog \(WWDG\)](#).

### 27.2 IWDG main features

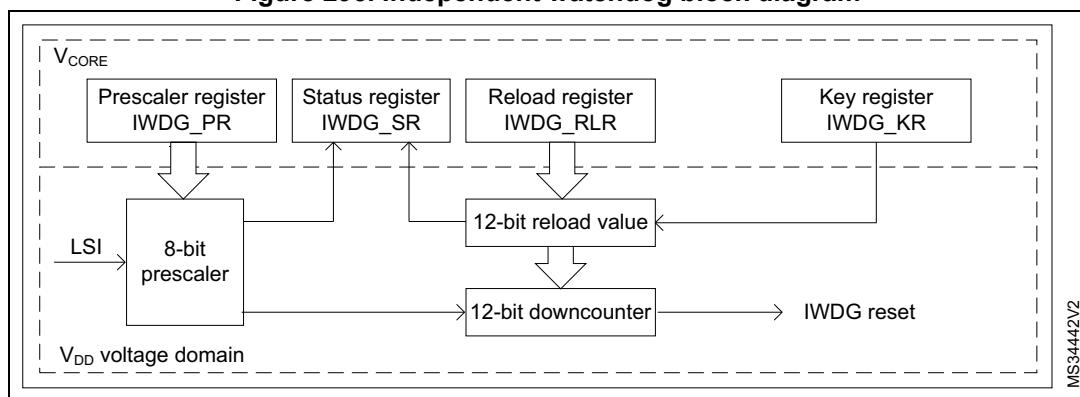
- Free-running downcounter
- Clocked from an independent RC oscillator (can operate in Standby and Stop modes)
- Conditional reset
  - Reset (if watchdog is activated) when the downcounter value becomes lower than 0x000
  - Reset (if watchdog is activated) if the downcounter is reloaded outside the window

### 27.3 IWDG functional description

#### 27.3.1 IWDG block diagram

[Figure 295](#) shows the functional blocks of the independent watchdog module.

**Figure 295. Independent watchdog block diagram**



1. The register interface is located in the  $V_{CORE}$  voltage domain. The watchdog function is located in the  $V_{DD}$  voltage domain, still functional in Stop and Standby modes.

When the independent watchdog is started by writing the value 0x0000 CCCC in the *IWDG key register (IWDG\_KR)*, the counter starts counting down from the reset value of 0xFFFF. When it reaches the end of count value (0x000), a reset signal is generated (IWDG reset).

Whenever the key value 0x0000 AAAA is written in the *IWDG key register (IWDG\_KR)*, the IWDG\_RLR value is reloaded in the counter, and the watchdog reset is prevented.

Once running, the IWDG cannot be stopped.

### 27.3.2 Window option

The IWDG can also work as a window watchdog by setting the appropriate window in the *IWDG window register (IWDG\_WINR)*.

If the reload operation is performed while the counter is greater than the value stored in the *IWDG window register (IWDG\_WINR)*, a reset is provided.

The default value of the *IWDG window register (IWDG\_WINR)* is 0x0000 0FFF, so if it is not updated, the window option is disabled.

As soon as the window value is changed, a reload operation is performed to reset the downcounter to the *IWDG reload register (IWDG\_RLR)* value, and to ease the cycle number calculation to generate the next reload.

#### Configuring the IWDG when the window option is enabled

1. Enable the IWDG by writing 0x0000 CCCC in the *IWDG key register (IWDG\_KR)*.
2. Enable register access by writing 0x0000 5555 in the *IWDG key register (IWDG\_KR)*.
3. Write the IWDG prescaler by programming *IWDG prescaler register (IWDG\_PR)* from 0 to 7.
4. Write the *IWDG reload register (IWDG\_RLR)*.
5. Wait for the registers to be updated (IWDG\_SR = 0x0000 0000).
6. Write to the *IWDG window register (IWDG\_WINR)*. This automatically refreshes the counter value in the *IWDG reload register (IWDG\_RLR)*.

*Note:* Writing the window value allows the counter value to be refreshed by the RLR when the *IWDG status register (IWDG\_SR)* is set to 0x0000 0000.

#### Configuring the IWDG when the window option is disabled

When the window option is not used, the IWDG can be configured as follows:

1. Enable the IWDG by writing 0x0000 CCCC in the *IWDG key register (IWDG\_KR)*.
2. Enable register access by writing 0x0000 5555 in the *IWDG key register (IWDG\_KR)*.
3. Write the prescaler by programming the *IWDG prescaler register (IWDG\_PR)* from 0 to 7.
4. Write the *IWDG reload register (IWDG\_RLR)*.
5. Wait for the registers to be updated (IWDG\_SR = 0x0000 0000).
6. Refresh the counter value with IWDG\_RLR (IWDG\_KR = 0x0000 AAAA).

### 27.3.3 Hardware watchdog

If this feature is enabled through the device option bits, the watchdog is automatically enabled at power-on, and generates a reset unless the [IWDG key register \(IWDG\\_KR\)](#) is written by the software before the counter reaches the end of count, and if the downcounter is lower than the window value (WIN[11:0]).

### 27.3.4 Low-power freeze

Depending on the IWDG\_STOP and IWDG\_STBY options configuration, the IWDG can continue counting or not during the Stop mode and the Standby mode, respectively. If the IWDG is kept running during Stop or Standby modes, it can wake up the device from this mode. Refer to [User and read protection option bytes](#) for more details.

### 27.3.5 Register access protection

Write access to [IWDG prescaler register \(IWDG\\_PR\)](#), [IWDG reload register \(IWDG\\_RLR\)](#), and [IWDG window register \(IWDG\\_WINR\)](#) is protected. To modify them, first write the code 0x0000 5555 in the [IWDG key register \(IWDG\\_KR\)](#). A write access to this register with a different value breaks the sequence, and register access is protected again. This is the case of the reload operation (writing 0x0000 AAAA).

A status register is available to indicate that an update of the prescaler, or of the downcounter reload value, or of the window value, is ongoing.

### 27.3.6 Debug mode

When the device enters Debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on the configuration of the corresponding bit in DBGMCU freeze register.

## 27.4 IWDG registers

Refer to [Section 1.2 on page 63](#) for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

### 27.4.1 IWDG key register (IWDG\_KR)

Address offset: 0x00

Reset value: 0x0000 0000 (reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KEY[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **KEY[15:0]**: Key value (write only, read 0x0000)

These bits must be written by software at regular intervals with the key value 0xAAAA, otherwise the watchdog generates a reset when the counter reaches 0.

Writing the key value 0x5555 to enable access to the IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers (see [Section 27.3.5: Register access protection](#))

Writing the key value 0xCCCC starts the watchdog (except if the hardware watchdog option is selected)

## 27.4.2 IWDG prescaler register (IWDG\_PR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PR[2:0]														
														rw	rw

Bits 31:3 Reserved, must be kept at reset value.

Bits 2:0 **PR[2:0]**: Prescaler divider

These bits are write access protected see [Section 27.3.5: Register access protection](#). They are written by software to select the prescaler divider feeding the counter clock. PVU bit of the [IWDG status register \(IWDG\\_SR\)](#) must be reset in order to be able to change the prescaler divider.

- 000: divider /4
- 001: divider /8
- 010: divider /16
- 011: divider /32
- 100: divider /64
- 101: divider /128
- 110: divider /256
- 111: divider /256

*Note: Reading this register returns the prescaler value from the V<sub>DD</sub> voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the PVU bit in the [IWDG status register \(IWDG\\_SR\)](#) is reset.*

### 27.4.3 IWDG reload register (IWDG\_RLR)

Address offset: 0x08

Reset value: 0x0000 0FFF (reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.												RL[11:0]
				rw											

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **RL[11:0]**: Watchdog counter reload value

These bits are write access protected see [Register access protection](#). They are written by software to define the value to be loaded in the watchdog counter each time the value 0xAAAA is written in the [IWDG key register \(IWDG\\_KR\)](#). The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to the datasheet for the timeout information.

The RVU bit in the [IWDG status register \(IWDG\\_SR\)](#) must be reset to be able to change the reload value.

*Note: Reading this register returns the reload value from the  $V_{DD}$  voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on it. For this reason the value read from this register is valid only when the RVU bit in the [IWDG status register \(IWDG\\_SR\)](#) is reset.*

## 27.4.4 IWDG status register (IWDG\_SR)

Address offset: 0x0C

Reset value: 0x0000 0000 (not reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	WVU	RVU	PVU												
													r	r	r

Bits 31:3 Reserved, must be kept at reset value.

Bit 2 **WVU**: Watchdog counter window value update

This bit is set by hardware to indicate that an update of the window value is ongoing. It is reset by hardware when the reload value update operation is completed in the V<sub>DD</sub> voltage domain (takes up to five RC 40 kHz cycles).

Window value can be updated only when WVU bit is reset.

Bit 1 **RVU**: Watchdog counter reload value update

This bit is set by hardware to indicate that an update of the reload value is ongoing. It is reset by hardware when the reload value update operation is completed in the V<sub>DD</sub> voltage domain (takes up to five RC 40 kHz cycles).

Reload value can be updated only when RVU bit is reset.

Bit 0 **PVU**: Watchdog prescaler value update

This bit is set by hardware to indicate that an update of the prescaler value is ongoing. It is reset by hardware when the prescaler update operation is completed in the V<sub>DD</sub> voltage domain (takes up to five RC 40 kHz cycles).

Prescaler value can be updated only when PVU bit is reset.

Note:

*If several reload, prescaler, or window values are used by the application, it is mandatory to wait until RVU bit is reset before changing the reload value, to wait until PVU bit is reset before changing the prescaler value, and to wait until WVU bit is reset before changing the window value. However, after updating the prescaler and/or the reload/window value it is not necessary to wait until RVU or PVU or WVU is reset before continuing code execution except in case of low-power mode entry.*

### 27.4.5 IWDG window register (IWDG\_WINR)

Address offset: 0x10

Reset value: 0x0000 0FFF (reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	WIN[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **WIN[11:0]**: Watchdog counter window value

These bits are write access protected, see [Section 27.3.5](#), they contain the high limit of the window value to be compared with the downcounter.

To prevent a reset, the downcounter must be reloaded when its value is lower than the window register value and greater than 0x0

The WVU bit in the [IWDG status register \(IWDG\\_SR\)](#) must be reset in order to be able to change the reload value.

*Note: Reading this register returns the reload value from the  $V_{DD}$  voltage domain. This value may not be valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the WVU bit in the [IWDG status register \(IWDG\\_SR\)](#) is reset.*

## **27.4.6 IWDG register map**

The following table gives the IWDG register map and reset values.

**Table 170. IWDG register map and reset values**

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 28 System window watchdog (WWDG)

### 28.1 Introduction

The system window watchdog (WWDG) is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence.

The watchdog circuit generates an MCU reset on expiry of a programmed time period, unless the program refreshes the contents of the down-counter before the T6 bit is cleared. An MCU reset is also generated if the 7-bit down-counter value (in the control register) is refreshed before the down-counter reaches the window register value. This implies that the counter must be refreshed in a limited window.

The WWDG clock is prescaled from the APB clock and has a configurable time window that can be programmed to detect abnormally late or early application behavior.

The WWDG is best suited for applications requiring the watchdog to react within an accurate timing window.

### 28.2 WWDG main features

- Programmable free-running down-counter
- Conditional reset
  - Reset (if watchdog activated) when the down-counter value becomes lower than 0x40
  - Reset (if watchdog activated) if the down-counter is reloaded outside the window (see [Figure 297](#))
- Early wake-up interrupt (EWI): triggered (if enabled and the watchdog activated) when the down-counter is equal to 0x40

### 28.3 WWDG functional description

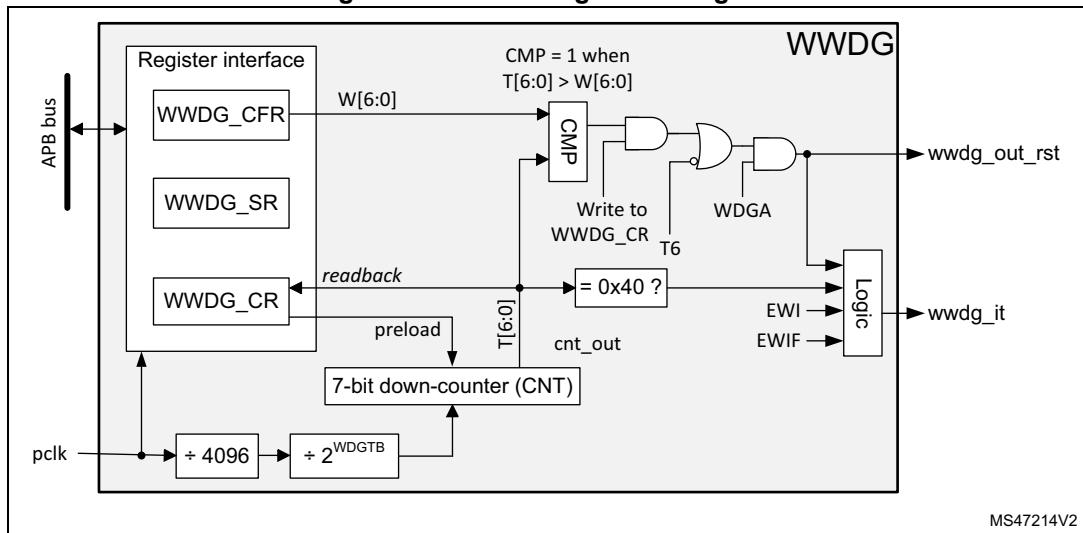
If the watchdog is activated (the WDGA bit is set in the WWDG\_CR register), and when the 7-bit down-counter (T[6:0] bits) is decremented from 0x40 to 0x3F (T6 becomes cleared), it initiates a reset. If the software reloads the counter while the counter is greater than the value stored in the window register, then a reset is generated.

The application program must write in the WWDG\_CR register at regular intervals during normal operation to prevent an MCU reset. This operation can take place only when the counter value is lower than or equal to the window register value, and higher than 0x3F. The value to be stored in the WWDG\_CR register must be between 0xFF and 0xC0.

Refer to [Figure 296](#) for the WWDG block diagram.

### 28.3.1 WWDG block diagram

Figure 296. Watchdog block diagram



### 28.3.2 Enabling the watchdog

When the user option WWDG\_SW selects “Software window watchdog”, the watchdog is always disabled after a reset. It is enabled by setting the WDGA bit in the WWDG\_CR register, then it cannot be disabled again, except by a reset.

When the user option WWDG\_SW selects “Hardware window watchdog”, the watchdog is always enabled after a reset, it cannot be disabled.

### 28.3.3 Controlling the down-counter

This down-counter is free-running, counting down even if the watchdog is disabled. When the watchdog is enabled, the T6 bit must be set to prevent generating an immediate reset.

The T[5:0] bits contain the number of increments that represent the time delay before the watchdog produces a reset. The timing varies between a minimum and a maximum value, due to the unknown status of the prescaler when writing to the WWDG\_CR register (see [Figure 297](#)). The [WWDG configuration register \(WWDG\\_CFR\)](#) contains the high limit of the window: to prevent a reset, the down-counter must be reloaded when its value is lower than or equal to the window register value, and greater than 0x3F. [Figure 297](#) describes the window watchdog process.

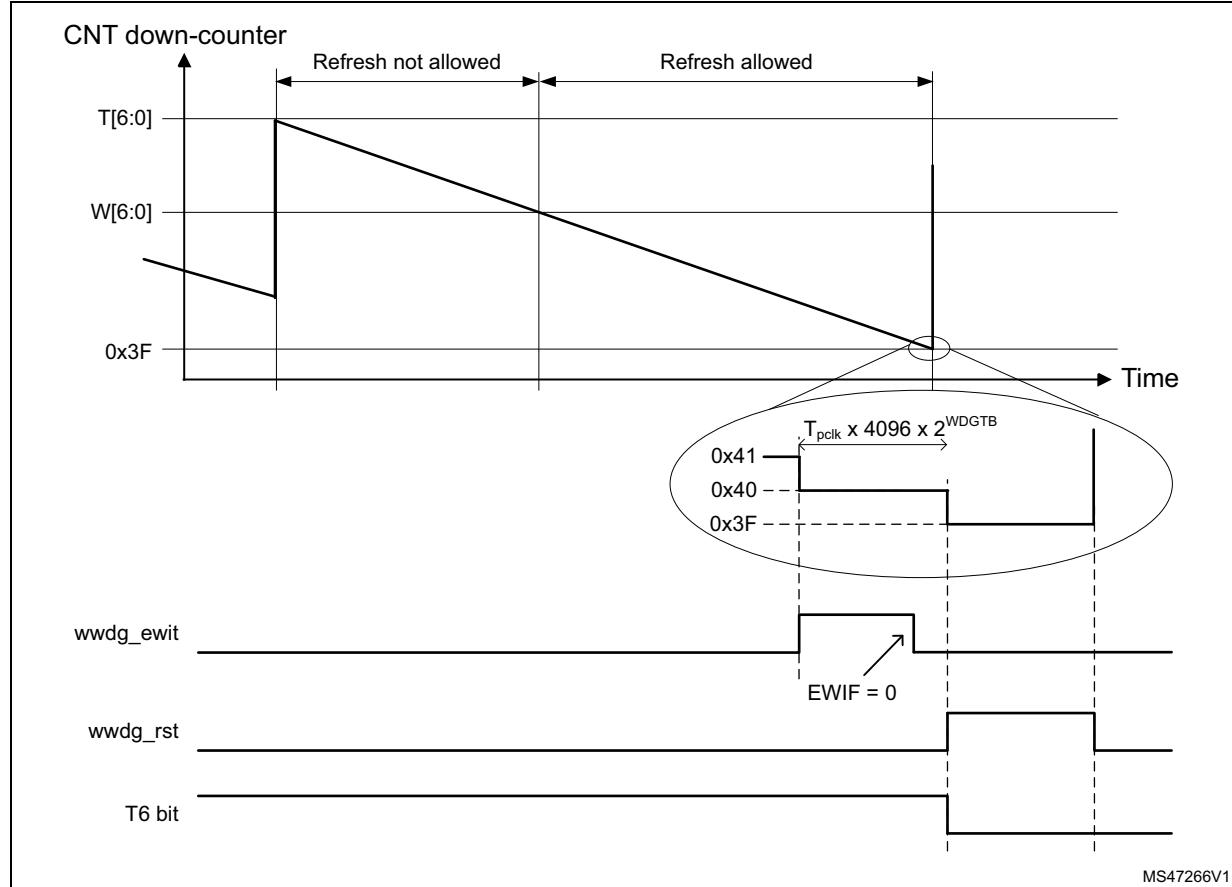
**Note:** The T6 bit can be used to generate a software reset (the WDGA bit is set and the T6 bit is cleared).

### 28.3.4 How to program the watchdog timeout

Use the formula in [Figure 297](#) 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 297. Window watchdog timing diagram



The formula to calculate the timeout value is given by:

$$t_{\text{WWDG}} = t_{\text{PCLK}} \times 4096 \times 2^{\text{WDGTB}[1:0]} \times (T[5:0] + 1) \quad (\text{ms})$$

where:

- $t_{\text{WWDG}}$ : WWDG timeout
- $t_{\text{PCLK}}$ : APB clock period measured in ms
- 4096: value corresponding to internal divider

As an example, if APB frequency is 48 MHz, WDGTB1:0] is set to 3, and T[5:0] is set to 63:

$$t_{\text{WWDG}} = (1 / 48000) \times 4096 \times 2^3 \times (63 + 1) = 43.69\text{ms}$$

Refer to the datasheet for the minimum and maximum values of  $t_{\text{WWDG}}$ .

### 28.3.5 Debug mode

When the device enters debug mode (processor halted), the WWDG counter either continues to work normally or stops, depending on the configuration bit in DBG module. For more details, refer to [Section 40.16.2: Debug support for timers, watchdog, bxCAN and I<sup>2</sup>C](#).

## 28.4 WWDG interrupts

The early wake-up interrupt (EWI) can be used if specific safety operations or data logging must be performed before the reset is generated. To enable the early wake-up interrupt, the application must:

- Write EWIF bit of WWDG\_SR register to 0, to clear unwanted pending interrupt
- Write EWI bit of WWDG\_CFR register to 1, to enable interrupt

When the down-counter reaches the value 0x40, a watchdog interrupt is generated, and the corresponding interrupt service routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

In some applications, the EWI interrupt can be used to manage a software system check and/or system recovery/graceful degradation, without generating a WWDG reset. In this case the corresponding ISR must reload the WWDG counter to avoid the WWDG reset, then trigger the required actions.

The watchdog interrupt is cleared by writing 0 to the EWIF bit in the WWDG\_SR register.

**Note:** *When the watchdog interrupt cannot be served (for example due to a system lock in a higher priority task), the WWDG reset is eventually generated.*

## 28.5 WWDG registers

Refer to [Section 1.2: List of abbreviations for registers](#) for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by halfwords (16-bit) or words (32-bit).

### 28.5.1 WWDG control register (WWDG\_CR)

Address offset: 0x000

Reset value: 0x0000 007F

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	WDGA	T[6:0]													
								rs	rw	rw	rw	rw	rw	rw	rw

Bits 31:8 Reserved, must be kept at reset value.

Bit 7 **WDGA:** Activation bit

This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

- 0: Watchdog disabled
- 1: Watchdog enabled

Bits 6:0 **T[6:0]:** 7-bit counter (MSB to LSB)

These bits contain the value of the watchdog counter, decremented every  $(4096 \times 2^{\text{WDGTB}1:0})$  PCLK cycles. A reset is produced when it is decremented from 0x40 to 0x3F (T6 becomes cleared).

**28.5.2 WWDG configuration register (WWDG\_CFR)**

Address offset: 0x004

Reset value: 0x0000 007F

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	EWI	WDGTB[1:0]	W[6:0]							
						rs	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:10 Reserved, must be kept at reset value.

Bit 9 **EWI:** Early wake-up interrupt enable

Set by software and cleared by hardware after a reset. When set, an interrupt occurs whenever the counter reaches the value 0x40.

Bits 8:7 **WDGTB[1:0]:** Timer base

The time base of the prescaler can be modified as follows:

- 00: CK counter clock (PCLK div 4096) div 1
- 01: CK counter clock (PCLK div 4096) div 2
- 10: CK counter clock (PCLK div 4096) div 4
- 11: CK counter clock (PCLK div 4096) div 8

Bits 6:0 **W[6:0]:** 7-bit window value

These bits contain the window value to be compared with the down-counter.

**28.5.3 WWDG status register (WWDG\_SR)**

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	EWIF														
															rc_w0

Bits 31:1 Reserved, must be kept at reset value.

Bit 0 **EWIF**: Early wake-up interrupt flag

This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. Writing 1 has no effect. This bit is also set if the interrupt is not enabled.

## 28.5.4 WWDG register map

The following table gives the WWDG register map and reset values.

**Table 171. WWDG register map and reset values**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	WWDG_CR	Res.																															
	Reset value																																
0x004	WWDG_CFR	Res.																															
	Reset value																																
0x008	WWDG_SR	Res.																															
	Reset value																																

Refer to [Section 2.2: Memory organization](#) for the register boundary addresses.

## 29 Real-time clock (RTC)

### 29.1 Introduction

The RTC provides an automatic wake-up to manage all low-power modes.

The real-time clock (RTC) is an independent BCD timer/counter. The RTC provides a time-of-day clock/calendar with programmable alarm interrupts.

The RTC includes also a periodic programmable wake-up flag with interrupt capability.

Two 32-bit registers contain the seconds, minutes, hours (12- or 24-hour format), day (day of week), date (day of month), month, and year, expressed in binary coded decimal format (BCD). The sub-seconds value is also available in binary format.

Compensations for 28-, 29- (leap year), 30-, and 31-day months are performed automatically. Daylight saving time compensation can also be performed.

Additional 32-bit registers contain the programmable alarm subseconds, seconds, minutes, hours, day, and date.

A digital calibration feature is available to compensate for any deviation in crystal oscillator accuracy.

After 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).

## 29.2 RTC main features

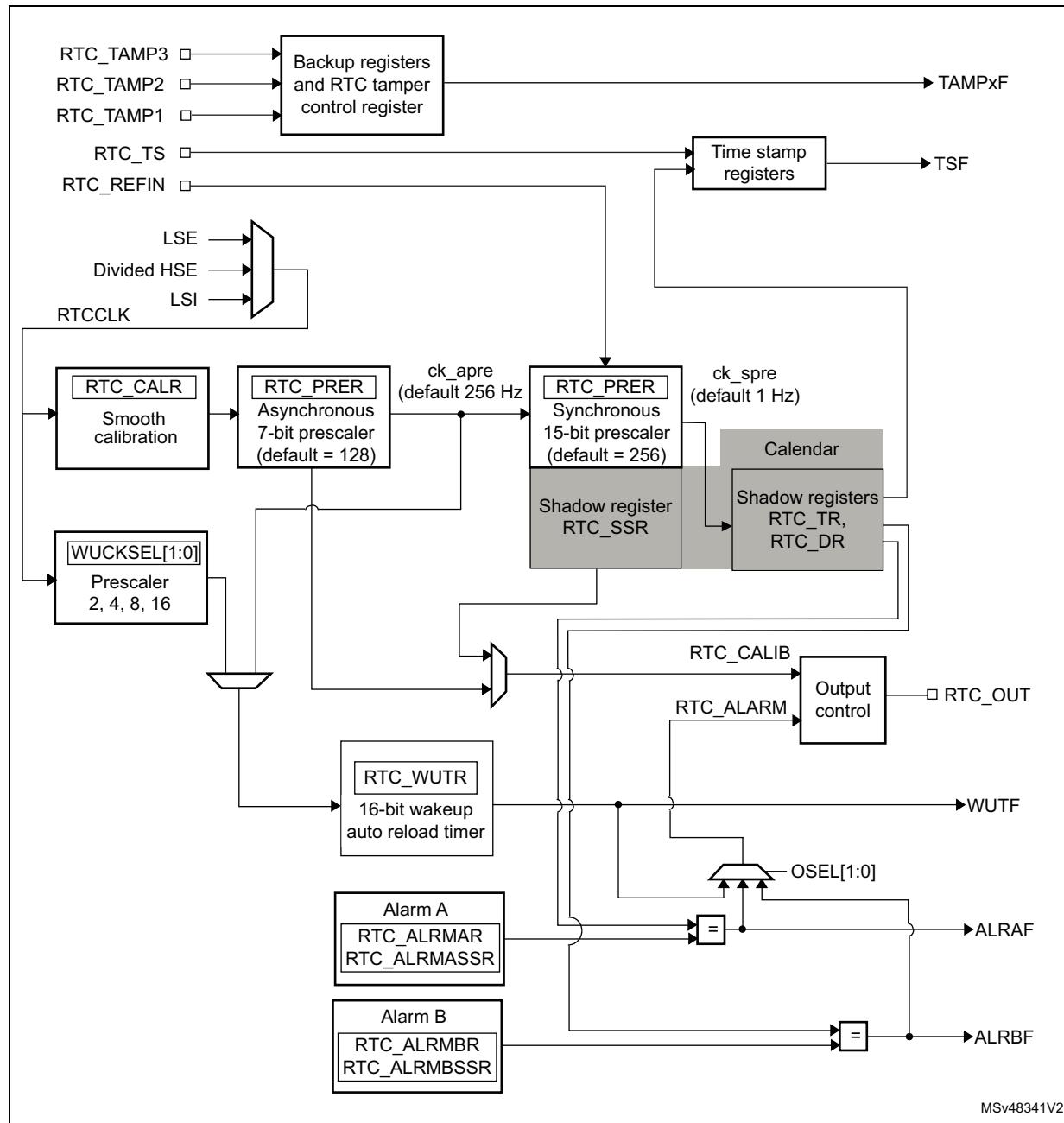
The RTC unit main features are the following (see [Figure 298: RTC block diagram](#)):

- Calendar with subseconds, seconds, minutes, hours (12 or 24 format), day (day of week), date (day of month), month, and year.
- Daylight saving compensation programmable by software.
- Programmable alarm with interrupt function. The alarm can be triggered by any combination of the calendar fields.
- Automatic wake-up unit generating a periodic flag that triggers an automatic wake-up interrupt.
- Reference clock detection: a more precise second source clock (50 or 60 Hz) can be used to enhance the calendar precision.
- Accurate synchronization with an external clock using the subsecond shift feature.
- Digital calibration circuit (periodic counter correction): 0.95 ppm accuracy, obtained in a calibration window of several seconds
- Time-stamp function for event saving
- Tamper detection event with configurable filter and internal pull-up
- Maskable interrupts/events:
  - Alarm A
  - Alarm B
  - Wake-up interrupt
  - Time-stamp
  - Tamper detection
- 32 backup registers.

## 29.3 RTC functional description

### 29.3.1 RTC block diagram

Figure 298. RTC block diagram



The RTC includes:

- Two alarms
- Three tamper events from I/Os
  - Tamper detection erases the backup registers.
- One timestamp event from I/O
- Tamper event detection can generate a timestamp event
- Timestamp can be generated when a switch to V<sub>BAT</sub> occurs
- 32 x 32-bit backup registers
  - The backup registers (RTC\_BKPxR) are implemented in the RTC domain that remains powered-on by VBAT when the VDD power is switched off.
- Output functions: RTC\_OUT which selects one of the following two outputs:
  - RTC\_CALIB: 512 Hz or 1Hz clock output (with an LSE frequency of 32.768 kHz). This output is enabled by setting the COE bit in the RTC\_CR register.
  - RTC\_ALARM: This output is enabled by configuring the OSEL[1:0] bits in the RTC\_CR register which select the Alarm A, Alarm B or Wakeup outputs.
- Input functions:
  - RTC\_TS: timestamp event
  - RTC\_TAMP1: tamper1 event detection
  - RTC\_TAMP2: tamper2 event detection
  - RTC\_TAMP3: tamper3 event detection
  - RTC\_REFIN: 50 or 60 Hz reference clock input

### 29.3.2 GPIOs controlled by the RTC

RTC\_OUT, RTC\_TS and RTC\_TAMP1 are mapped on the same pin (PC13). PC13 pin configuration is controlled by the RTC, whatever the PC13 GPIO configuration, except for the RTC\_ALARM output open-drain mode. The RTC functions mapped on PC13 are available in all low-power modes and in VBAT mode.

The output mechanism follows the priority order shown in [Table 172](#).

**Table 172. RTC pin PC13 configuration<sup>(1)</sup>**

PC13 Pin configuration and function	OSEL[1:0] bits (RTC_ALARM output enable)	COE bit (RTC_CALIB output enable)	RTC_ALARM_TYPE bit	TAMP1E bit (RTC_TAMP1 input enable)	TSE bit (RTC_TS input enable)	TSINSEL bits
RTC_ALARM output OD	01 or 10 or 11	Don't care	0	Don't care	Don't care	Don't care
RTC_ALARM output PP	01 or 10 or 11	Don't care	1	Don't care	Don't care	Don't care
RTC_CALIB output PP	00	1	Don't care	Don't care	Don't care	Don't care

Table 172. RTC pin PC13 configuration<sup>(1)</sup> (continued)

PC13 Pin configuration and function	OSEL[1:0] bits (RTC_ALARM output enable)	COE bit (RTC_CALIB output enable)	RTC_ALARM_TYPE bit	TAMP1E bit (RTC_TAMP1 input enable)	TSE bit (RTC_TS input enable)	TSINSEL bits
RTC_TAMP1 input floating	00	0	Don't care	1	0	Don't care
	00	1				
	01 or 10 or 11	0				
RTC_TS and RTC_TAMP1 input floating	00	0	Don't care	1	1	00
	00	1				
	01 or 10 or 11	0				
RTC_TS input floating	00	0	Don't care	0	1	00
	00	1				
	01 or 10 or 11	0				
Wakeup pin or Standard GPIO	00	0	Don't care	0	0	Don't care

1. OD: open drain; PP: push-pull.

RTC\_TAMP2 and RTC\_TS are mapped on the same pin (PI8). PI8 configuration is controlled by the RTC, whatever the PI8 GPIO configuration. The RTC functions mapped on PI8 are available in all low-power modes and in VBAT mode.

The output mechanism follows the priority order shown in [Table 173](#).

Table 173. RTC pin PI8 configuration

PI8 pin configuration and function	TAMP2E bit (RTC_TAMP2 input enable)	TSE bit (RTC_TS input enable)	TSINSEL bit (Timestamp pin selection)
RTC_TAMP2 input floating	1	0	Don't care
RTC_TS and RTC_TAMP2 input floating	1	1	01
RTC_TS input floating	0	1	01
Wakeup pin or Standard GPIO	0	0	Don't care

RTC\_TAMP3 and RTC\_TS are mapped on the same pin (PC1). PC1 configuration is controlled by the RTC, whatever the PC1 GPIO configuration. The RTC functions mapped on PC1 are available in all low-power modes, but are not available in VBAT mode.

**Table 174. RTC pin PC2 configuration**

PC2 pin configuration and function	TAMP2E bit (RTC_TAMP3 input enable)	TSE bit (RTC_TS input enable)	TSINSEL bit (Timestamp pin selection)
RTC_TAMP3 input floating	1	0	Don't care
RTC_TS and RTC_TAMP3 input floating	1	1	10 or 11
RTC_TS input floating	0	1	01
Wakeup pin or Standard GPIO	0	0	Don't care

RTC\_REFIN is mapped on PB15. PB15 must be configured in alternate function mode to allow RTC\_REFIN function. RTC\_REFIN is not available in VBAT and in Standby mode.

The table below summarizes the RTC pins and functions capabilities in all modes.

**Table 175. RTC functions over modes**

Pin	RTC functions	Functional in all low-power modes except Standby modes	Functional in Standby mode	Functional in VBAT mode
PC13	RTC_TAMP1 RTC_TS RTC_OUT	YES	YES	YES
PI8	RTC_TAMP2 RTC_TS	YES	YES	YES
PC1	RTC_TAMP3 RTC_TS	YES	YES	NO
PB15	RTC_REFIN	YES	NO	NO

### 29.3.3 Clock and prescalers

The RTC clock source (RTCCCLK) is selected through the clock controller among the LSE clock, the LSI oscillator clock, and the HSE clock. For more information on the RTC clock source configuration, refer to [Section 5: Reset and clock control \(RCC\)](#).

A programmable prescaler stage generates a 1 Hz clock which is used to update the calendar. To minimize power consumption, the prescaler is split into 2 programmable prescalers (see [Figure 298: 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 wake-up auto-reload timer. To obtain short timeout periods, the 16-bit wake-up auto-reload timer can also run with the RTCCLK divided by the programmable 4-bit asynchronous prescaler (see [Section 29.3.6: Periodic auto-wake-up](#) for details).

#### 29.3.4 Real-time clock and calendar

The RTC calendar time and date registers are accessed through shadow registers which are synchronized with PCLK (APB clock). They can also be accessed directly in order to avoid waiting for the synchronization duration.

- RTC\_SSR for the subseconds
- RTC\_TR for the time
- RTC\_DR for the date

Every RTCCLK period, the current calendar value is copied into the shadow registers, and the RSF bit of RTC\_ISR register is set (see [Section 29.6.4: RTC initialization and status register \(RTC\\_ISR\)](#)). The copy is not performed in Stop and Standby mode. When exiting these modes, the shadow registers are updated after up to 1 RTCCLK period.

When the application reads the calendar registers, it accesses the content of the shadow registers. It is possible to make a direct access to the calendar registers by setting the BYPSHAD control bit in the RTC\_CR register. By default, this bit is cleared, and the user accesses the shadow registers.

When reading the RTC\_SSR, RTC\_TR or RTC\_DR registers in BYPSHAD=0 mode, the frequency of the APB clock ( $f_{APB}$ ) must be at least 7 times the frequency of the RTC clock ( $f_{RTCCLK}$ ).

The shadow registers are reset by system reset.

#### 29.3.5 Programmable alarms

The RTC unit provides programmable alarm: Alarm A and Alarm B. The description below is given for Alarm A, but can be translated in the same way for Alarm B.

The programmable alarm function is enabled through the ALRAE bit in the RTC\_CR register. The ALRAF is set to 1 if the calendar subseconds, seconds, minutes, hours, date or day match the values programmed in the alarm registers RTC\_ALRMASSR and RTC\_ALRMAR. Each calendar field can be independently selected through the MSKx bits of the RTC\_ALRMAR register, and through the MASKSx bits of the RTC\_ALRMASSR register. The alarm interrupt is enabled through the ALRAIE bit in the RTC\_CR register.

**Caution:** If the seconds field is selected (MSK1 bit reset in RTC\_ALRMAR), the synchronous prescaler division factor set in the RTC\_PRER register must be at least 3 to ensure correct behavior.

Alarm A and Alarm B (if enabled by bits OSEL[1:0] in RTC\_CR register) can be routed to the RTC\_ALARM output. RTC\_ALARM output polarity can be configured through bit POL the RTC\_CR register.

### 29.3.6 Periodic auto-wake-up

The periodic wake-up flag is generated by a 16-bit programmable auto-reload down-counter. The wake-up timer range can be extended to 17 bits.

The wake-up function is enabled through the WUTE bit in the RTC\_CR register.

The wake-up timer clock input can be:

- RTC clock (RTCCLK) divided by 2, 4, 8, or 16.  
When RTCCLK is LSE(32.768 kHz), this allows to configure the wake-up interrupt period from 122 µs to 32 s, with a resolution down to 61 µs.
- ck\_spre (usually 1 Hz internal clock)  
When ck\_spre frequency is 1Hz, this allows to achieve a wake-up time from 1 s to around 36 hours with one-second resolution. This large programmable time range is divided in 2 parts:
  - from 1s to 18 hours when WUCKSEL [2:1] = 10
  - and from around 18h to 36h when WUCKSEL[2:1] = 11. In this last case 2<sup>16</sup> is added to the 16-bit counter current value. When the initialization sequence is complete (see [Programming the wake-up timer on page 911](#)), the timer starts counting down. When the wake-up function is enabled, the down-counting remains active in low-power modes. In addition, when it reaches 0, the WUTF flag is set in the RTC\_ISR register, and the wake-up counter is automatically reloaded with its reload value (RTC\_WUTR register value).

The WUTF flag must then be cleared by software.

When the periodic wake-up interrupt is enabled by setting the WUTIE bit in the RTC\_CR register, it can exit the device from low-power modes.

The periodic wake-up flag can be routed to the RTC\_ALARM output provided it has been enabled through bits OSEL[1:0] of RTC\_CR register. RTC\_ALARM output polarity can be configured through the POL bit in the RTC\_CR register.

System reset, as well as low-power modes (Sleep, Stop and Standby) have no influence on the wake-up timer.

### 29.3.7 RTC initialization and configuration

#### RTC register access

The RTC registers are 32-bit registers. The APB interface introduces two wait states in RTC register accesses except on read accesses to calendar shadow registers when BYPSHAD = 0.

#### RTC register write protection

After system reset, the RTC registers are protected against parasitic write access by clearing the DBP bit in the PWR\_CR1 register (refer to the power control section). DBP bit must be set in order to enable RTC registers write access.

After Backup domain reset, all the RTC registers are write-protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC\_WPR.

The following steps are required to unlock the write protection on all the RTC registers except for RTC\_TAMPCR, RTC\_BKPxR, RTC\_OR and RTC\_ISR[31:8].

1. Write '0xCA' into the RTC\_WPR register.
2. Write '0x53' into the RTC\_WPR register.

Writing a wrong key reactivates the write protection.

The protection mechanism is not affected by system reset.

#### Calendar initialization and configuration

To program the initial time and date calendar values, including the time format and the prescaler configuration, the following sequence is required:

1. Set INIT bit to 1 in the RTC\_ISR register to enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated.
2. Poll INITF bit of in the RTC\_ISR register. The initialization phase mode is entered when INITF is set to 1. It takes around 2 RTCCLK clock cycles (due to clock synchronization).
3. To generate a 1 Hz clock for the calendar counter, program both the prescaler factors in RTC\_PRER register.
4. Load the initial time and date values in the shadow registers (RTC\_TR and RTC\_DR), and configure the time format (12 or 24 hours) through the FMT bit in the RTC\_CR register.
5. Exit the initialization mode by clearing the INIT bit. The actual calendar counter value is then automatically loaded and the counting restarts after 4 RTCCLK clock cycles.

When the initialization sequence is complete, the calendar starts counting.

Note:

*After a system reset, the application can read the INITS flag in the RTC\_ISR register to check if the calendar has been initialized or not. If this flag equals 0, the calendar has not been initialized since the year field is set at its Backup domain reset default value (0x00).*

*To read the calendar after initialization, the software must first check that the RSF flag is set in the RTC\_ISR register.*

## Daylight saving time

The daylight saving time management is performed through bits SUB1H, ADD1H, and BKP of the RTC\_CR register.

Using SUB1H or ADD1H, the software can subtract or add one hour to the calendar in one single operation without going through the initialization procedure.

In addition, the software can use the BKP bit to memorize this operation.

## Programming the alarm

A similar procedure must be followed to program or update the programmable alarms. The procedure below is given for Alarm A but can be translated in the same way for Alarm B.

1. Clear ALRAE in RTC\_CR to disable Alarm A.
2. Program the Alarm A registers (RTC\_ALRMASSR/RTC\_ALRMAR).
3. Set ALRAE in the RTC\_CR register to enable Alarm A again.

*Note:*

*Each change of the RTC\_CR register is taken into account after around 2 RTCCLK clock cycles due to clock synchronization.*

## Programming the wake-up timer

The following sequence is required to configure or change the wake-up timer auto-reload value (WUT[15:0] in RTC\_WUTR):

1. Clear WUTE in RTC\_CR to disable the wake-up timer.
2. Poll WUTWF until it is set in RTC\_ISR to make sure the access to wake-up auto-reload counter and to WUCKSEL[2:0] bits is allowed. It takes around 2 RTCCLK clock cycles (due to clock synchronization).
3. Program the wake-up auto-reload value WUT[15:0], and the wake-up clock selection (WUCKSEL[2:0] bits in RTC\_CR). Set WUTE in RTC\_CR to enable the timer again. The wake-up timer restarts down-counting. The WUTWF bit is cleared up to 2 RTCCLK clock cycles after WUTE is cleared, due to clock synchronization.

### 29.3.8 Reading the calendar

#### When BYPSHAD control bit is cleared in the RTC\_CR register

To read the RTC calendar registers (RTC\_SSR, RTC\_TR and RTC\_DR) properly, the APB clock frequency ( $f_{PCLK}$ ) must be equal to or greater than seven times the RTC clock frequency ( $f_{RTCCLK}$ ). This ensures a secure behavior of the synchronization mechanism.

If the APB clock frequency is less than seven times the RTC clock frequency, the software must read the calendar time and date registers twice. If the second read of the RTC\_TR gives the same result as the first read, this ensures that the data is correct. Otherwise a third read access must be done. In any case the APB clock frequency must never be lower than the RTC clock frequency.

The RSF bit is set in RTC\_ISR register each time the calendar registers are copied into the RTC\_SSR, RTC\_TR and RTC\_DR shadow registers. The copy is performed every RTCCLK cycle. To ensure consistency between the 3 values, reading either RTC\_SSR or RTC\_TR locks the values in the higher-order calendar shadow registers until RTC\_DR is read. In case the software makes read accesses to the calendar in a time interval smaller than 1 RTCCLK period: RSF must be cleared by software after the first calendar read, and

then the software must wait until RSF is set before reading again the RTC\_SSR, RTC\_TR and RTC\_DR registers.

After waking up from low-power mode (Stop or Standby), RSF must be cleared by software. The software must then wait until it is set again before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers.

The RSF bit must be cleared after wake-up and not before entering low-power mode.

After a system reset, the software must wait until RSF is set before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers. Indeed, a system reset resets the shadow registers to their default values.

After an initialization (refer to [Calendar initialization and configuration on page 910](#)): the software must wait until RSF is set before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers.

After synchronization (refer to [Section 29.3.10: RTC synchronization](#)): the software must wait until RSF is set before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers.

### When the BYPSHAD control bit is set in the RTC\_CR register (bypass shadow registers)

Reading the calendar registers gives the values from the calendar counters directly, thus eliminating the need to wait for the RSF bit to be set. This is especially useful after exiting from low-power modes (STOP or Standby), since the shadow registers are not updated during these modes.

When the BYPSHAD bit is set to 1, the results of the different registers might not be coherent with each other if an RTCCLK edge occurs between two read accesses to the registers. Additionally, the value of one of the registers may be incorrect if an RTCCLK edge occurs during the read operation. The software must read all the registers twice, and then compare the results to confirm that the data is coherent and correct. Alternatively, the software can just compare the two results of the least-significant calendar register.

*Note:* While BYPSHAD=1, instructions which read the calendar registers require one extra APB cycle to complete.

### 29.3.9 Resetting the RTC

The calendar shadow registers (RTC\_SSR, RTC\_TR and RTC\_DR) and some bits of the RTC status register (RTC\_ISR) are reset to their default values by all available system reset sources.

On the contrary, the following registers are reset to their default values by a Backup domain reset and are not affected by a system reset: the RTC current calendar registers, the RTC control register (RTC\_CR), the prescaler register (RTC\_PRER), the RTC calibration register (RTC\_CALR), the RTC shift register (RTC\_SHIFTR), the RTC timestamp registers (RTC\_TSSSR, RTC\_TSTR and RTC\_TSDDR), the RTC tamper configuration register (RTC\_TAMPCCR), the RTC backup registers (RTC\_BKPxR), the wake-up timer register (RTC\_WUTR), the Alarm A and Alarm B registers (RTC\_ALRMASSR/RTC\_ALRMAR and RTC\_ALRMBSSR/RTC\_ALRMBR), and the Option register (RTC\_OR).

In addition, when it is clocked by the LSE, the RTC keeps on running under system reset if the reset source is different from the Backup domain reset one (refer to the RTC clock section of the Reset and clock controller for details on the list of RTC clock sources not

affected by system reset). When a Backup domain reset occurs, the RTC is stopped and all the RTC registers are set to their reset values.

### 29.3.10 RTC synchronization

The RTC can be synchronized to a remote clock with a high degree of precision. After reading the sub-second field (RTC\_SSR or RTC\_TSSSR), a calculation can be made of the precise offset between the times being maintained by the remote clock and the RTC. The RTC can then be adjusted to eliminate this offset by “shifting” its clock by a fraction of a second using RTC\_SHIFTR.

RTC\_SSR contains the value of the synchronous prescaler counter. This allows one to calculate the exact time being maintained by the RTC down to a resolution of  $1 / (\text{PREDIV\_S} + 1)$  seconds. As a consequence, the resolution can be improved by increasing the synchronous prescaler value (PREDIV\_S[14:0]. The maximum resolution allowed (30.52 µs with a 32768 Hz clock) is obtained with PREDIV\_S set to 0x7FFF.

However, increasing PREDIV\_S means that PREDIV\_A must be decreased in order to maintain the synchronous prescaler output at 1 Hz. In this way, the frequency of the asynchronous prescaler output increases, which may increase the RTC dynamic consumption.

The RTC can be finely adjusted using the RTC shift control register (RTC\_SHIFTR). Writing to RTC\_SHIFTR can shift (either delay or advance) the clock by up to a second with a resolution of  $1 / (\text{PREDIV\_S} + 1)$  seconds. The shift operation consists of adding the SUBFS[14:0] value to the synchronous prescaler counter SS[15:0]: this will delay the clock. If at the same time the ADD1S bit is set, this results in adding one second and at the same time subtracting a fraction of second, so this will advance the clock.

**Caution:** Before initiating a shift operation, the user must check that SS[15] = 0 in order to ensure that no overflow will occur.

As soon as a shift operation is initiated by a write to the RTC\_SHIFTR register, the SHPF flag is set by hardware to indicate that a shift operation is pending. This bit is cleared by hardware as soon as the shift operation has completed.

**Caution:** This synchronization feature is not compatible with the reference clock detection feature: firmware must not write to RTC\_SHIFTR when REFCKON=1.

### 29.3.11 RTC reference clock detection

The update of the RTC calendar can be synchronized to a reference clock, RTC\_REFIN, which is usually the mains frequency (50 or 60 Hz). The precision of the RTC\_REFIN reference clock should be higher than the 32.768 kHz LSE clock. When the RTC\_REFIN detection is enabled (REFCKON bit of RTC\_CR set to 1), the calendar is still clocked by the LSE, and RTC\_REFIN is used to compensate for the imprecision of the calendar update frequency (1 Hz).

Each 1 Hz clock edge is compared to the nearest RTC\_REFIN clock edge (if one is found within a given time window). In most cases, the two clock edges are properly aligned. When the 1 Hz clock becomes misaligned due to the imprecision of the LSE clock, the RTC shifts the 1 Hz clock a bit so that future 1 Hz clock edges are aligned. Thanks to this mechanism, the calendar becomes as precise as the reference clock.

The RTC detects if the reference clock source is present by using the 256 Hz clock (ck\_apre) generated from the 32.768 kHz quartz. The detection is performed during a time

window around each of the calendar updates (every 1 s). The window equals 7 ck\_apre periods when detecting the first reference clock edge. A smaller window of 3 ck\_apre periods is used for subsequent calendar updates.

Each time the reference clock is detected in the window, the synchronous prescaler which outputs the ck\_spre clock is forced to reload. This has no effect when the reference clock and the 1 Hz clock are aligned because the prescaler is being reloaded at the same moment. When the clocks are not aligned, the reload shifts future 1 Hz clock edges a little for them to be aligned with the reference clock.

If the reference clock halts (no reference clock edge occurred during the 3 ck\_apre window), the calendar is updated continuously based solely on the LSE clock. The RTC then waits for the reference clock using a large 7 ck\_apre period detection window centered on the ck\_spre edge.

When the RTC\_REFIN detection is enabled, PREDIV\_A and PREDIV\_S must be set to their default values:

- PREDIV\_A = 0x007F
- PREDIV\_S = 0x00FF

*Note:* RTC\_REFIN clock detection is not available in Standby mode.

### 29.3.12 RTC smooth digital calibration

The RTC frequency can be digitally calibrated with a resolution of about 0.954 ppm with a range from -487.1 ppm to +488.5 ppm. The correction of the frequency is performed using series of small adjustments (adding and/or subtracting individual RTCCLK pulses). These adjustments are fairly well distributed so that the RTC is well calibrated even when observed over short durations of time.

The smooth digital calibration is performed during a cycle of about  $2^{20}$  RTCCLK pulses, or 32 seconds when the input frequency is 32768 Hz. This cycle is maintained by a 20-bit counter, cal\_cnt[19:0], clocked by RTCCLK.

The smooth calibration register (RTC\_CALR) specifies the number of RTCCLK clock cycles to be masked during the 32-second cycle:

- Setting the bit CALM[0] to 1 causes exactly one pulse to be masked during the 32-second cycle.
- Setting CALM[1] to 1 causes two additional cycles to be masked
- Setting CALM[2] to 1 causes four additional cycles to be masked
- and so on up to CALM[8] set to 1 which causes 256 clocks to be masked.

*Note:* CALM[8:0] (RTC\_CALR) specifies the number of RTCCLK pulses to be masked during the 32-second cycle. Setting the bit CALM[0] to '1' causes exactly one pulse to be masked during the 32-second cycle at the moment when cal\_cnt[19:0] is 0x80000; CALM[1]=1 causes two other cycles to be masked (when cal\_cnt is 0x40000 and 0xC0000); CALM[2]=1 causes four other cycles to be masked (cal\_cnt = 0x20000/0x60000/0xA0000/ 0xE0000); and so on up to CALM[8]=1 which causes 256 clocks to be masked (cal\_cnt = 0xXX800).

While CALM allows the RTC frequency to be reduced by up to 487.1 ppm with fine resolution, the bit CALP can be used to increase the frequency by 488.5 ppm. Setting CALP to '1' effectively inserts an extra RTCCLK pulse every  $2^{11}$  RTCCLK cycles, which means that 512 clocks are added during every 32-second cycle.

Using CALM together with CALP, an offset ranging from -511 to +512 RTCCLK cycles can be added during the 32-second cycle, which translates to a calibration range of -487.1 ppm to +488.5 ppm with a resolution of about 0.954 ppm.

The formula to calculate the effective calibrated frequency ( $F_{CAL}$ ) given the input frequency ( $F_{RTCCLK}$ ) is as follows:

$$F_{CAL} = F_{RTCCLK} \times [1 + (CALP \times 512 - CALM) / (2^{20} + CALM - CALP \times 512)]$$

### Calibration when PREDIV\_A<3

The CALP bit can not be set to 1 when the asynchronous prescaler value (PREDIV\_A bits in RTC\_PRER register) is less than 3. If CALP was already set to 1 and PREDIV\_A bits are set to a value less than 3, CALP is ignored and the calibration operates as if CALP was equal to 0.

To perform a calibration with PREDIV\_A less than 3, the synchronous prescaler value (PREDIV\_S) should be reduced so that each second is accelerated by 8 RTCCLK clock cycles, which is equivalent to adding 256 clock cycles every 32 seconds. As a result, between 255 and 256 clock pulses (corresponding to a calibration range from 243.3 to 244.1 ppm) can effectively be added during each 32-second cycle using only the CALM bits.

With a nominal RTCCLK frequency of 32768 Hz, when PREDIV\_A equals 1 (division factor of 2), PREDIV\_S should be set to 16379 rather than 16383 (4 less). The only other interesting case is when PREDIV\_A equals 0, PREDIV\_S should be set to 32759 rather than 32767 (8 less).

If PREDIV\_S is reduced in this way, the formula given the effective frequency of the calibrated input clock is as follows:

$$F_{CAL} = F_{RTCCLK} \times [1 + (256 - CALM) / (2^{20} + CALM - 256)]$$

In this case, CALM[7:0] equals 0x100 (the midpoint of the CALM range) is the correct setting if RTCCLK is exactly 32768.00 Hz.

### Verifying the RTC calibration

RTC precision is ensured by measuring the precise frequency of RTCCLK and calculating the correct CALM value and CALP values. An optional 1 Hz output is provided to allow applications to measure and verify the RTC precision.

Measuring the precise frequency of the RTC over a limited interval can result in a measurement error of up to 2 RTCCLK clock cycles over the measurement period, depending on how the digital calibration cycle is aligned with the measurement period.

However, this measurement error can be eliminated if the measurement period is the same length as the calibration cycle period. In this case, the only error observed is the error due to the resolution of the digital calibration.

- By default, the calibration cycle period is 32 seconds.

Using this mode and measuring the accuracy of the 1 Hz output over exactly 32 seconds guarantees that the measure is within 0.477 ppm (0.5 RTCCLK cycles over 32 seconds, due to the limitation of the calibration resolution).

- CALW16 bit of the RTC\_CALR register can be set to 1 to force a 16- second calibration cycle period.

In this case, the RTC precision can be measured during 16 seconds with a maximum error of 0.954 ppm (0.5 RTCCLK cycles over 16 seconds). However, since the calibration

resolution is reduced, the long term RTC precision is also reduced to 0.954 ppm: CALM[0] bit is stuck at 0 when CALW16 is set to 1.

- CALW8 bit of the RTC\_CALR register can be set to 1 to force a 8- second calibration cycle period.

In this case, the RTC precision can be measured during 8 seconds with a maximum error of 1.907 ppm (0.5 RTCCLK cycles over 8s). The long term RTC precision is also reduced to 1.907 ppm: CALM[1:0] bits are stuck at 00 when CALW8 is set to 1.

### Re-calibration on-the-fly

The calibration register (RTC\_CALR) can be updated on-the-fly while RTC\_ISR/INITF=0, by using the follow process:

1. Poll the RTC\_ISR/RECALPF (re-calibration pending flag).
2. If it is set to 0, write a new value to RTC\_CALR, if necessary. RECALPF is then automatically set to 1
3. Within three ck\_apre cycles after the write operation to RTC\_CALR, the new calibration settings take effect.

#### 29.3.13 Time-stamp function

Time-stamp is enabled by setting the TSE or ITSE bits of RTC\_CR register to 1.

When TSE is set:

The calendar is saved in the time-stamp registers (RTC\_TSSSR, RTC\_TSTR, RTC\_TSDR) when a time-stamp event is detected on the RTC\_TS pin.

When ITSE is set:

The calendar is saved in the time-stamp registers (RTC\_TSSSR, RTC\_TSTR, RTC\_TSDR) when an internal time-stamp event is detected. The internal timestamp event is generated by the switch to the VBAT supply.

When a time-stamp event occurs, due to internal or external event, the time-stamp flag bit (TSF) in RTC\_ISR register is set. In case the event is internal, the ITSF flag is also set in RTC\_ISR register.

By setting the TSIE bit in the RTC\_CR register, an interrupt is generated when a time-stamp event occurs.

If a new time-stamp event is detected while the time-stamp flag (TSF) is already set, the time-stamp overflow flag (TSOVF) flag is set and the time-stamp registers (RTC\_TSTR and RTC\_TSDR) maintain the results of the previous event.

**Note:** *TSF is set 2 ck\_apre cycles after the time-stamp event occurs due to synchronization process.*

*There is no delay in the setting of TSOVF. This means that if two time-stamp events are close together, TSOVF can be seen as '1' while TSF is still '0'. As a consequence, it is recommended to poll TSOVF only after TSF has been set.*

**Caution:** If a time-stamp event occurs immediately after the TSF bit is supposed to be cleared, then both TSF and TSOVF bits are set. To avoid masking a time-stamp event occurring at the same moment, the application must not write '0' into TSF bit unless it has already read it to '1'.

Optionally, a tamper event can cause a time-stamp to be recorded. See the description of the TAMPTS control bit in [Section 29.6.16: RTC tamper configuration register \(RTC\\_TAMPCR\)](#).

### 29.3.14 Tamper detection

The RTC\_TAMPx input events can be configured either for edge detection, or for level detection with filtering.

The tamper detection can be configured for the following purposes:

- erase the RTC backup registers (default configuration)
- generate an interrupt, capable to wake-up from Stop and Standby modes
- generate a hardware trigger for the low-power timers

#### RTC backup registers

The backup registers (RTC\_BKPxR) are not reset by system reset or when the device wakes up from Standby mode.

The backup registers are reset when a tamper detection event occurs (see [Section 29.6.20: RTC backup registers \(RTC\\_BKPxR\)](#) and [Tamper detection initialization on page 917](#)) except if the TAMPxNOERASE bit is set, or if TAMPxF is set in the RTC\_TAMPCR register.

#### Tamper detection initialization

Each input can be enabled by setting the corresponding TAMPxE bits to 1 in the RTC\_TAMPCR register.

Each RTC\_TAMPx tamper detection input is associated with a flag TAMPxF in the RTC\_ISR register.

When TAMPxF is cleared:

The TAMPxF flag is asserted after the tamper event on the pin, with the latency provided below:

- 3 ck\_apre cycles when TAMPFLT differs from 0x0 (Level detection with filtering)
- 3 ck\_apre cycles when TAMPTS=1 (Timestamp on tamper event)
- No latency when TAMPFLT=0x0 (Edge detection) and TAMPTS=0

A new tamper occurring on the same pin during this period and as long as TAMPxF is set cannot be detected.

When TAMPxF is set:

A new tamper occurring on the same pin cannot be detected during the latency described above and 2.5 ck\_RTC additional cycles.

By setting the TAMPIE bit in the RTC\_TAMPCR register, an interrupt is generated when a tamper detection event occurs (when TAMPxF is set). Setting TAMPIE is not allowed when one or more TAMPxF is set.

When TAMPIE is cleared, each tamper pin event interrupt can be individually enabled by setting the corresponding TAMPxIE bit in the RTC\_TAMPCR register. Setting TAMPxIE is not allowed when the corresponding TAMPxF is set.

### Trigger output generation on tamper event

The tamper event detection can be used as trigger input by the low-power timers.

When TAMPxMF bit in cleared in RTC\_TAMPCR register, the TAMPxF flag must be cleared by software in order to allow a new tamper detection on the same pin.

When TAMPxMF bit is set, the TAMPxF flag is masked, and kept cleared in RTC\_ISR register. This configuration allows to trig automatically the low-power timers in Stop mode, without requiring the system wakeup to perform the TAMPxF clearing. In this case, the backup registers are not cleared.

### Timestamp on tamper event

With TAMPTS set to '1', any tamper event causes a timestamp to occur. In this case, either the TSF bit or the TSOVF bit are set in RTC\_ISR, in the same manner as if a normal timestamp event occurs. The affected tamper flag register TAMPxF is set at the same time that TSF or TSOVF is set.

### Edge detection on tamper inputs

If the TAMPFLT bits are "00", the RTC\_TAMPx pins generate tamper detection events when either a rising edge or a falling edge is observed depending on the corresponding TAMPxTRG bit. The internal pull-up resistors on the RTC\_TAMPx inputs are deactivated when edge detection is selected.

**Caution:** When using the edge detection, it is recommended to check by software the tamper pin level just after enabling the tamper detection (by reading the GPIO registers), and before writing sensitive values in the backup registers, to ensure that an active edge did not occur before enabling the tamper event detection.

When TAMPFLT="00" and TAMPxTRG = 0 (rising edge detection), a tamper event may be detected by hardware if the tamper input is already at high level before enabling the tamper detection.

After a tamper event has been detected and cleared, the RTC\_TAMPx should be disabled and then re-enabled (TAMPxE set to 1) before re-programming the backup registers (RTC\_BKPxR). This prevents the application from writing to the backup registers while the RTC\_TAMPx input value still indicates a tamper detection. This is equivalent to a level detection on the RTC\_TAMPx input.

**Note:** *Tamper detection is still active when V<sub>DD</sub> power is switched off. To avoid unwanted resetting of the backup registers, the pin to which the RTC\_TAMPx is mapped should be externally tied to the correct level.*

### Level detection with filtering on RTC\_TAMPx inputs

Level detection with filtering is performed by setting TAMPFLT to a non-zero value. A tamper detection event is generated when either 2, 4, or 8 (depending on TAMPFLT) consecutive samples are observed at the level designated by the TAMPxTRG bits.

The RTC\_TAMPx inputs are precharged through the I/O internal pull-up resistance before its state is sampled, unless disabled by setting TAMPPUDIS to 1. The duration of the precharge is determined by the TAMPPRCH bits, allowing for larger capacitances on the RTC\_TAMPx inputs.

The trade-off between tamper detection latency and power consumption through the pull-up can be optimized by using TAMPFREQ to determine the frequency of the sampling for level detection.

**Note:** Refer to the datasheets for the electrical characteristics of the pull-up resistors.

### 29.3.15 Calibration clock output

When the COE bit is set to 1 in the RTC\_CR register, a reference clock is provided on the RTC\_CALIB device output.

If the COSEL bit in the RTC\_CR register is reset and PREDIV\_A = 0x7F, the RTC\_CALIB frequency is  $f_{RTCCLK}/64$ . This corresponds to a calibration output at 512 Hz for an RTCCLK frequency at 32.768 kHz. The RTC\_CALIB duty cycle is irregular: there is a light jitter on falling edges. It is therefore recommended to use rising edges.

When COSEL is set and “PREDIV\_S+1” is a non-zero multiple of 256 (i.e: PREDIV\_S[7:0] = 0xFF), the RTC\_CALIB frequency is  $f_{RTCCLK}/(256 * (PREDIV_A+1))$ . This corresponds to a calibration output at 1 Hz for prescaler default values (PREDIV\_A = 0x7F, PREDIV\_S = 0xFF), with an RTCCLK frequency at 32.768 kHz. The 1 Hz output is affected when a shift operation is on going and may toggle during the shift operation (SHPF=1).

**Note:** When COSEL bit is cleared, the RTC\_CALIB output is the output of the 6th stage of the asynchronous prescaler.

When COSEL bit is set, the RTC\_CALIB output is the output of the 8th stage of the synchronous prescaler.

### 29.3.16 Alarm output

The OSEL[1:0] control bits in the RTC\_CR register are used to activate the alarm output RTC\_ALARM, and to select the function which is output. These functions reflect the contents of the corresponding flags in the RTC\_ISR register.

The polarity of the output is determined by the POL control bit in RTC\_CR so that the opposite of the selected flag bit is output when POL is set to 1.

#### Alarm output

The RTC\_ALARM pin can be configured in output open drain or output push-pull using RTC\_OR register.

**Note:** Once the RTC\_ALARM output is enabled, it has priority over RTC\_CALIB (COE bit is don't care and must be kept cleared).

## 29.4 RTC low-power modes

Table 176. Effect of low-power modes on RTC

Mode	Description
Sleep	No effect RTC interrupts cause the device to exit the Sleep mode.
Stop	The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC timestamp event, and RTC wake-up cause the device to exit the Stop mode.
Standby	The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC timestamp event, and RTC wake-up cause the device to exit the Standby mode.

## 29.5 RTC interrupts

All RTC interrupts are connected to the EXTI controller. Refer to [Section 11: Extended interrupts and events controller \(EXTI\)](#).

To enable the RTC Alarm interrupt, the following sequence is required:

1. Configure and enable the EXTI line corresponding to the RTC Alarm event in interrupt mode and select the rising edge sensitivity.
2. Configure and enable the RTC\_ALARM IRQ channel in the NVIC.
3. Configure the RTC to generate RTC alarms.

To enable the RTC Tamper interrupt, the following sequence is required:

1. Configure and enable the EXTI line corresponding to the RTC Tamper event in interrupt mode and select the rising edge sensitivity.
2. Configure and Enable the RTC\_TAMP\_STAMP IRQ channel in the NVIC.
3. Configure the RTC to detect the RTC tamper event.

To enable the RTC TimeStamp interrupt, the following sequence is required:

1. Configure and enable the EXTI line corresponding to the RTC TimeStamp event in interrupt mode and select the rising edge sensitivity.
2. Configure and Enable the RTC\_TAMP\_STAMP IRQ channel in the NVIC.
3. Configure the RTC to detect the RTC time-stamp event.

To enable the Wakeup timer interrupt, the following sequence is required:

1. Configure and enable the EXTI line corresponding to the Wakeup timer even in interrupt mode and select the rising edge sensitivity.
2. Configure and Enable the RTC\_WKUP IRQ channel in the NVIC.
3. Configure the RTC to detect the RTC Wakeup timer event.

**Table 177. Interrupt control bits**

Interrupt event	Event flag	Enable control bit	Exit from Sleep mode	Exit from Stop mode	Exit from Standby mode
Alarm A	ALRAF	ALRAIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
Alarm B	ALRBF	ALRBIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
RTC_TS input (timestamp)	TSF	TSIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
RTC_TAMP1 input detection	TAMP1F	TAMPIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
RTC_TAMP2 input detection	TAMP2F	TAMPIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
RTC_TAMP3 input detection	TAMP3F	TAMPIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
Wakeup timer interrupt	WUTF	WUTIE	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.

## 29.6 RTC registers

Refer to [Section 1.2 on page 63](#) of the reference manual for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by words (32-bit).

### 29.6.1 RTC time register (RTC\_TR)

The RTC\_TR is the calendar time shadow register. This register must be written in initialization mode only. Refer to [Calendar initialization and configuration on page 910](#) and [Reading the calendar on page 911](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).

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	rw

Bits 31:23 Reserved, must be kept at reset value.

Bit 22 **PM**: AM/PM notation

- 0: AM or 24-hour format
- 1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format

Bits 19:16 **HU[3:0]**: Hour units in BCD format

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format

Bits 11:8 **MNU[3:0]**: Minute units in BCD format

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **ST[2:0]**: Second tens in BCD format

Bits 3:0 **SU[3:0]**: Second units in BCD format

## 29.6.2 RTC date register (RTC\_DR)

The RTC\_DR is the calendar date shadow register. This register must be written in initialization mode only. Refer to [Calendar initialization and configuration on page 910](#) and [Reading the calendar on page 911](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).

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[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

### 29.6.3 RTC control register (RTC\_CR)

Address offset: 0x08

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	ITSE	COE	OSEL[1:0]	POL	COSEL	BKP	SUB1H	ADD1H	
							rw	rw	rw	rw	rw	rw	rw	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSIE	WUTIE	ALRBIE	ALRAIE	TSE	WUTE	ALRBE	ALRAE	Res.	FMT	BYPS HAD	REFCKON	TSEDGE	WUCKSEL[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:25 Reserved, must be kept at reset value.

Bit 24 **ITSE**: timestamp on internal event enable

0: internal event timestamp disabled  
1: internal event timestamp enabled

Bit 23 **COE**: Calibration output enable

This bit enables the RTC\_CALIB output  
0: Calibration output disabled  
1: Calibration output enabled

Bits 22:21 **OSEL[1:0]**: Output selection

These bits are used to select the flag to be routed to RTC\_ALARM output  
00: Output disabled  
01: Alarm A output enabled  
10: Alarm B output enabled  
11: Wake-up output enabled

Bit 20 **POL**: Output polarity

This bit is used to configure the polarity of RTC\_ALARM output  
0: The pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL[1:0])  
1: The pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL[1:0]).

Bit 19 **COSEL**: Calibration output selection

When COE=1, this bit selects which signal is output on RTC\_CALIB.  
0: Calibration output is 512 Hz (with default prescaler setting)  
1: Calibration output is 1 Hz (with default prescaler setting)

These frequencies are valid for RTCCLK at 32.768 kHz and prescalers at their default values (PREDIV\_A=127 and PREDIV\_S=255). Refer to [Section 29.3.15: Calibration clock output](#)

Bit 18 **BKP**: Backup

This bit can be written by the user to memorize whether the daylight saving time change has been performed or not.

**Bit 17 SUB1H:** Subtract 1 hour (winter time change)

When this bit is set, 1 hour is subtracted to the calendar time if the current hour is not 0. This bit is always read as 0.

Setting this bit has no effect when current hour is 0.

0: No effect

1: Subtracts 1 hour to the current time. This can be used for winter time change outside initialization mode.

**Bit 16 ADD1H:** Add 1 hour (summer time change)

When this bit is set, 1 hour is added to the calendar time. This bit is always read as 0.

0: No effect

1: Adds 1 hour to the current time. This can be used for summer time change outside initialization mode.

**Bit 15 TSIE:** Time-stamp interrupt enable

0: Time-stamp Interrupt disable

1: Time-stamp Interrupt enable

**Bit 14 WUTIE:** Wake-up timer interrupt enable

0: Wake-up timer interrupt disabled

1: Wake-up timer interrupt enabled

**Bit 13 ALRBIE:** Alarm B interrupt enable

0: Alarm B Interrupt disable

1: Alarm B Interrupt enable

**Bit 12 ALRAIE:** Alarm A interrupt enable

0: Alarm A interrupt disabled

1: Alarm A interrupt enabled

**Bit 11 TSE:** timestamp enable

0: timestamp disable

1: timestamp enable

**Bit 10 WUTE:** Wake-up timer enable

0: Wake-up timer disabled

1: Wake-up timer enabled

*Note: When the wake-up timer is disabled, wait for WUTWF=1 before enabling it again.*

**Bit 9 ALRBE:** Alarm B enable

0: Alarm B disabled

1: Alarm B enabled

**Bit 8 ALRAE:** Alarm A enable

0: Alarm A disabled

1: Alarm A enabled

**Bit 7** Reserved, must be kept at reset value.**Bit 6 FMT:** Hour format

0: 24 hour/day format

1: AM/PM hour format

Bit 5 **BYPSHAD**: Bypass the shadow registers

0: Calendar values (when reading from RTC\_SSR, RTC\_TR, and RTC\_DR) are taken from the shadow registers, which are updated once every two RTCCLK cycles.

1: Calendar values (when reading from RTC\_SSR, RTC\_TR, and RTC\_DR) are taken directly from the calendar counters.

*Note: If the frequency of the APB clock is less than seven times the frequency of RTCCLK, BYPShad must be set to '1'.*

Bit 4 **REFCKON**: RTC\_REFIN reference clock detection enable (50 or 60 Hz)

0: RTC\_REFIN detection disabled

1: RTC\_REFIN detection enabled

*Note: PREDIV\_S must be 0x00FF.*

Bit 3 **TSEDGE**: Time-stamp event active edge

0: RTC\_TS input rising edge generates a time-stamp event

1: RTC\_TS input falling edge generates a time-stamp event

TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting.

Bits 2:0 **WUCKSEL[2:0]**: Wake-up clock selection

000: RTC/16 clock is selected

001: RTC/8 clock is selected

010: RTC/4 clock is selected

011: RTC/2 clock is selected

10x: ck\_spre (usually 1 Hz) clock is selected

11x: ck\_spre (usually 1 Hz) clock is selected and  $2^{16}$  is added to the WUT counter value (see note below)

*Note: Bits 7, 6 and 4 of this register can be written in initialization mode only (RTC\_ISR/INITF = 1).*

*WUT = Wake-up unit counter value. WUT = (0x0000 to 0xFFFF) + 0x10000 added when WUCKSEL[2:1 = 11].*

*Bits 2 to 0 of this register can be written only when RTC\_CR WUTE bit = 0 and RTC\_ISR WUTWF bit = 1.*

*It is recommended not to change the hour during the calendar hour increment as it could mask the incrementation of the calendar hour.*

*ADD1H and SUB1H changes are effective in the next second.*

*This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).*

**Caution:** TSE must be reset when TSEDGE is changed to avoid spuriously setting of TSF.

## 29.6.4 RTC initialization and status register (RTC\_ISR)

Address offset: 0x0C

Backup domain reset value: 0x0000 0007

System reset: not affected except INIT, INITF, and RSF bits which are cleared to '0'

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ITSF	RECALPF
															rc_w0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAMP3F	TAMP2F	TAMP1F	TSOVF	TSF	WUTF	ALRBF	ALRAF	INIT	INITF	RSF	INITS	SHPF	WUTWF	ALRBWF	ALRAWF
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rw	r	rc_w0	r	r	r	r	r

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **ITSF**: Internal tTime-stamp flag

This flag is set by hardware when a time-stamp on the internal event occurs.

This flag is cleared by software by writing 0, and must be cleared together with TSF bit by writing 0 in both bits.

Bit 16 **RECALPF**: Recalibration pending Flag

The RECALPF status flag is automatically set to '1' when software writes to the RTC\_CALR register, indicating that the RTC\_CALR register is blocked. When the new calibration settings are taken into account, this bit returns to '0'. Refer to [Re-calibration on-the-fly](#).

Bit 15 **TAMP3F**: RTC\_TAMP3 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC\_TAMP3 input.

It is cleared by software writing 0

Bit 14 **TAMP2F**: RTC\_TAMP2 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC\_TAMP2 input.

It is cleared by software writing 0

Bit 13 **TAMP1F**: RTC\_TAMP1 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC\_TAMP1 input.

It is cleared by software writing 0

Bit 12 **TSOVF**: Time-stamp overflow flag

This flag is set by hardware when a time-stamp event occurs while TSF is already set.

This flag is cleared by software by writing 0. It is recommended to check and then clear TSOVF only after clearing the TSF bit. Otherwise, an overflow might not be noticed if a time-stamp event occurs immediately before the TSF bit is cleared.

Bit 11 **TSF**: Time-stamp flag

This flag is set by hardware when a time-stamp event occurs.

This flag is cleared by software by writing 0. If ITSF flag is set, TSF must be cleared together with ITSF by writing 0 in both bits.

**Bit 10 WUTF:** Wake-up timer flag

This flag is set by hardware when the wake-up auto-reload counter reaches 0.

This flag is cleared by software by writing 0.

This flag must be cleared by software at least 1.5 RTCCLK periods before WUTF is set to 1 again.

**Bit 9 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.

*Note: This bit is write protected. The write access procedure is described in RTC register write protection on page 910.*

**Bit 6 INITF:** Initialization flag

When this bit is set to 1, the RTC is in initialization state, and the time, date and prescaler registers can be updated.

0: Calendar registers update is not allowed

1: Calendar registers update is allowed

**Bit 5 RSF:** Registers synchronization flag

This bit is set by hardware each time the calendar registers are copied into the shadow registers (RTC\_SSR, RTC\_TR and RTC\_DR). This bit is cleared by hardware in initialization mode, while a shift operation is pending (SHPF=1), or when in bypass shadow register mode (BYPSHAD=1). This bit can also be cleared by software.

It is cleared either by software or by hardware in initialization mode.

0: Calendar shadow registers not yet synchronized

1: Calendar shadow registers synchronized

*Note: This bit is write protected. The write access procedure is described in RTC register write protection on page 910.*

**Bit 4 INITS:** Initialization status flag

This bit is set by hardware when the calendar year field is different from 0 (Backup domain reset state).

0: Calendar has not been initialized

1: Calendar has been initialized

**Bit 3 SHPF:** Shift operation pending

0: No shift operation is pending

1: A shift operation is pending

This flag is set by hardware as soon as a shift operation is initiated by a write to the RTC\_SHIFTR register. It is cleared by hardware when the corresponding shift operation has been executed. Writing to the SHPF bit has no effect.

**Bit 2 WUTWF:** Wake-up timer write flag

This bit is set by hardware up to 2 RTCCLK cycles after the WUTE bit has been set to 0 in RTC\_CR, and is cleared up to 2 RTCCLK cycles after the WUTE bit has been set to 1. The wake-up timer values can be changed when WUTE bit is cleared and WUTWF is set.

- 0: Wake-up timer configuration update not allowed
- 1: Wake-up timer configuration update allowed

**Bit 1 ALRBWF:** Alarm B write flag

This bit is set by hardware when Alarm B values can be changed, after the ALRBE bit has been set to 0 in RTC\_CR.

It is cleared by hardware in initialization mode.

- 0: Alarm B update not allowed
- 1: Alarm B update allowed

**Bit 0 ALRAWF:** Alarm A write flag

This bit is set by hardware when Alarm A values can be changed, after the ALRAE bit has been set to 0 in RTC\_CR.

It is cleared by hardware in initialization mode.

- 0: Alarm A update not allowed
- 1: Alarm A update allowed

**Note:** *The bits ALRAF, ALRBF, WUTF and TSF are cleared 2 APB clock cycles after programming them to 0.*

### 29.6.5 RTC prescaler register (RTC\_PRER)

This register must be written in initialization mode only. The initialization must be performed in two separate write accesses. Refer to [Calendar initialization and configuration on page 910](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).

Address offset: 0x10

Backup domain reset value: 0x007F 00FF

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16								
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PREDIV_A[6:0]														
									rw	rw	rw	rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
Res.	PREDIV_S[14:0]																						
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw								

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:16 **PREDIV\_A[6:0]**: Asynchronous prescaler factor

This is the asynchronous division factor:

$$\text{ck_apre frequency} = \text{RTCCLK frequency}/(\text{PREDIV\_A}+1)$$

Bit 15 Reserved, must be kept at reset value.

Bits 14:0 **PREDIV\_S[14:0]**: Synchronous prescaler factor

This is the synchronous division factor:

$$\text{ck_spre frequency} = \text{ck_apre frequency}/(\text{PREDIV\_S}+1)$$

## 29.6.6 RTC wake-up timer register (RTC\_WUTR)

This register can be written only when WUTWF is set to 1 in RTC\_ISR.

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).

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]**: Wake-up auto-reload value bits

When the wake-up timer is enabled (WUTE set to 1), the WUTF flag is set every (WUT[15:0] + 1) ck\_wut cycles. The ck\_wut period is selected through WUCKSEL[2:0] bits of the RTC\_CR register

When WUCKSEL[2] = 1, the wake-up timer becomes 17-bits and WUCKSEL[1] effectively becomes WUT[16] the most-significant bit to be reloaded into the timer.

The first assertion of WUTF occurs (WUT+1) ck\_wut cycles after WUTE is set. Setting WUT[15:0] to 0x0000 with WUCKSEL[2:0] =011 (RTCCLK/2) is forbidden.

### 29.6.7 RTC alarm A register (RTC\_ALRMAR)

This register can be written only when ALRAWF is set to 1 in RTC\_ISR, or in initialization mode.

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).

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.

### 29.6.8 RTC alarm B register (RTC\_ALRMBR)

This register can be written only when ALRBWF is set to 1 in RTC\_ISR, or in initialization mode.

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).

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

### 29.6.9 RTC write protection register (RTC\_WPR)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	KEY[7:0]														
								w	w	w	w	w	w	w	w

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **KEY[7:0]**: Write protection key

This byte is written by software.

Reading this byte always returns 0x00.

Refer to [RTC register write protection](#) for a description of how to unlock RTC register write protection.

### 29.6.10 RTC sub second register (RTC\_SSR)

Address offset: 0x28

Backup domain reset value: 0x0000 0000

System reset: 0x0000 0000 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **SS[15:0]**: Sub second value

SS[15:0] is the value in the synchronous prescaler counter. The fraction of a second is given by the formula below:

Second fraction = (PREDIV\_S - SS) / (PREDIV\_S + 1)

Note: SS can be larger than PREDIV\_S only after a shift operation. In that case, the correct time/date is one second less than as indicated by RTC\_TR/RTC\_DR.

### 29.6.11 RTC shift control register (RTC\_SHIFTR)

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).

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[14:0]**: Subtract a fraction of a second

These bits are write only and is always read as zero. Writing to this bit has no effect when a shift operation is pending (when SHPF=1, in RTC\_ISR).

The value which is written to SUBFS is added to the synchronous prescaler counter. Since this counter counts down, this operation effectively subtracts from (delays) the clock by:

Delay (seconds) = SUBFS / (PREDIV\_S + 1)

A fraction of a second can effectively be added to the clock (advancing the clock) when the ADD1S function is used in conjunction with SUBFS, effectively advancing the clock by:

Advance (seconds) = (1 - (SUBFS / (PREDIV\_S + 1))).

*Note: Writing to SUBFS causes RSF to be cleared. Software can then wait until RSF=1 to be sure that the shadow registers have been updated with the shifted time.*

### 29.6.12 RTC timestamp time register (RTC\_TSTR)

The content of this register is valid only when TSF is set to 1 in RTC\_ISR. It is cleared when TSF bit is reset.

Address offset: 0x30

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PM	HT[1:0]		HU[3:0]			
									r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MNT[2:0]			MNU[3:0]				Res.	ST[2:0]			SU[3:0]			
	r	r	r	r	r	r	r		r	r	r	r	r	r	r

Bits 31:23 Reserved, must be kept at reset value.

Bit 22 **PM**: AM/PM notation

0: AM or 24-hour format

1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format.

Bits 19:16 **HU[3:0]**: Hour units in BCD format.

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format.

Bits 11:8 **MNU[3:0]**: Minute units in BCD format.

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **ST[2:0]**: Second tens in BCD format.

Bits 3:0 **SU[3:0]**: Second units in BCD format.

### 29.6.13 RTC timestamp date register (RTC\_TSDR)

The content of this register is valid only when TSF is set to 1 in RTC\_ISR. It is cleared when TSF bit is reset.

Address offset: 0x34

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDU[2:0]			MT	MU[3:0]				Res.	Res.	DT[1:0]		DU[3:0]			
r	r	r	r	r	r	r	r			r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:13 **WDU[2:0]**: Week day units

Bit 12 **MT**: Month tens in BCD format

Bits 11:8 **MU[3:0]**: Month units in BCD format

Bits 7:6 Reserved, must be kept at reset value.

Bits 5:4 **DT[1:0]**: Date tens in BCD format

Bits 3:0 **DU[3:0]**: Date units in BCD format

### 29.6.14 RTC time-stamp sub second register (RTC\_TSSSR)

The content of this register is valid only when RTC\_ISR/TSF is set. It is cleared when the RTC\_ISR/TSF bit is reset.

Address offset: 0x38

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
SS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **SS[15:0]**: Sub second value

SS[15:0] is the value of the synchronous prescaler counter when the timestamp event occurred.

### 29.6.15 RTC calibration register (RTC\_CALR)

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#).

Address offset: 0x3C

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CALP	CALW8	CALW16	Res.	Res.	Res.	Res.	CALM[8:0]								
rw	rw	rw					rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bit 15 **CALP**: Increase frequency of RTC by 488.5 ppm

0: No RTCCLK pulses are added.

1: One RTCCLK pulse is effectively inserted every  $2^{11}$  pulses (frequency increased by 488.5 ppm).

This feature is intended to be used in conjunction with CALM, which lowers the frequency of the calendar with a fine resolution. If the input frequency is 32768 Hz, the number of RTCCLK pulses added during a 32-second window is calculated as follows:  $(512 * \text{CALP}) - \text{CALM}$ .

Refer to [Section 29.3.12: RTC smooth digital calibration](#).

Bit 14 **CALW8**: Use an 8-second calibration cycle period

When CALW8 is set to '1', the 8-second calibration cycle period is selected.

Note: CALM[1:0] are stuck at "00" when CALW8='1'. Refer to [Section 29.3.12: RTC smooth digital calibration](#).

Bit 13 **CALW16**: Use a 16-second calibration cycle period

When CALW16 is set to '1', the 16-second calibration cycle period is selected. This bit must not be set to '1' if CALW8=1.

Note: CALM[0] is stuck at '0' when CALW16='1'. Refer to [Section 29.3.12: RTC smooth digital calibration](#).

Bits 12:9 Reserved, must be kept at reset value.

Bits 8:0 **CALM[8:0]**: Calibration minus

The frequency of the calendar is reduced by masking CALM out of  $2^{20}$  RTCCLK pulses (32 seconds if the input frequency is 32768 Hz). This decreases the frequency of the calendar with a resolution of 0.9537 ppm.

To increase the frequency of the calendar, this feature should be used in conjunction with CALP. See [Section 29.3.12: RTC smooth digital calibration on page 914](#).

### 29.6.16 RTC tamper configuration register (RTC\_TAMPCCR)

Address offset: 0x40

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TAMP3 MF	TAMP3 NO ERASE	TAMP3 IE	TAMP2 MF	TAMP2 NO ERASE	TAMP2 IE	TAMP1 MF	TAMP1 NO ERASE	TAMP1 IE
							rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAMP PUDIS	TAMPPRCH [1:0]	TAMPFLT[1:0]	TAMPFREQ[2:0]	TAMP TS	TAMP3 TRG	TAMP3 E	TAMP2 TRG	TAMP2 E	TAMP1 TRG	TAMP1 E	TAMP1 E				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:25 Reserved, must be kept at reset value.

Bit 24 **TAMP3MF**: Tamper 3 mask flag

- 0: Tamper 3 event generates a trigger event and TAMP3F must be cleared by software to allow next tamper event detection.
- 1: Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased.

*Note: The Tamper 3 interrupt must not be enabled when TAMP3MF is set.*

Bit 23 **TAMP3NOERASE**: Tamper 3 no erase

- 0: Tamper 3 event erases the backup registers.
- 1: Tamper 3 event does not erase the backup registers.

Bit 22 **TAMP3IE**: Tamper 3 interrupt enable

- 0: Tamper 3 interrupt is disabled if TAMPIE = 0.
- 1: Tamper 3 interrupt enabled.

Bit 21 **TAMP2MF**: Tamper 2 mask flag

- 0: Tamper 2 event generates a trigger event and TAMP2F must be cleared by software to allow next tamper event detection.
- 1: Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.

*Note: The Tamper 2 interrupt must not be enabled when TAMP2MF is set.*

Bit 20 **TAMP2NOERASE**: Tamper 2 no erase

- 0: Tamper 2 event erases the backup registers.
- 1: Tamper 2 event does not erase the backup registers.

Bit 19 **TAMP2IE**: Tamper 2 interrupt enable

- 0: Tamper 2 interrupt is disabled if TAMPIE = 0.
- 1: Tamper 2 interrupt enabled.

Bit 18 **TAMP1MF**: Tamper 1 mask flag

- 0: Tamper 1 event generates a trigger event and TAMP1F must be cleared by software to allow next tamper event detection.
- 1: Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware. The backup registers are not erased.

*Note: The Tamper 1 interrupt must not be enabled when TAMP1MF is set.*

- Bit 17 **TAMP1NOERASE**: Tamper 1 no erase  
 0: Tamper 1 event erases the backup registers.  
 1: Tamper 1 event does not erase the backup registers.
- Bit 16 **TAMP1IE**: Tamper 1 interrupt enable  
 0: Tamper 1 interrupt is disabled if TAMP1IE = 0.  
 1: Tamper 1 interrupt enabled.
- Bit 15 **TAMPPUDIS**: RTC\_TAMPx pull-up disable  
 This bit determines if each of the RTC\_TAMPx pins are precharged before each sample.  
 0: Precharge RTC\_TAMPx pins before sampling (enable internal pull-up)  
 1: Disable precharge of RTC\_TAMPx pins.
- Bits 14:13 **TAMPPRCH[1:0]**: RTC\_TAMPx precharge duration  
 These bit determines the duration of time during which the pull-up/is activated before each sample. TAMPPRCH is valid for each of the RTC\_TAMPx inputs.  
 0x0: 1 RTCCLK cycle  
 0x1: 2 RTCCLK cycles  
 0x2: 4 RTCCLK cycles  
 0x3: 8 RTCCLK cycles
- Bits 12:11 **TAMPFLT[1:0]**: RTC\_TAMPx filter count  
 These bits determines the number of consecutive samples at the specified level (TAMP\*TRG) needed to activate a Tamper event. TAMPFLT is valid for each of the RTC\_TAMPx inputs.  
 0x0: Tamper event is activated on edge of RTC\_TAMPx input transitions to the active level (no internal pull-up on RTC\_TAMPx input).  
 0x1: Tamper event is activated after 2 consecutive samples at the active level.  
 0x2: Tamper event is activated after 4 consecutive samples at the active level.  
 0x3: Tamper event is activated after 8 consecutive samples at the active level.
- Bits 10:8 **TAMPFREQ[2:0]**: Tamper sampling frequency  
 Determines the frequency at which each of the RTC\_TAMPx inputs are sampled.  
 0x0: RTCCLK / 32768 (1 Hz when RTCCLK = 32768 Hz)  
 0x1: RTCCLK / 16384 (2 Hz when RTCCLK = 32768 Hz)  
 0x2: RTCCLK / 8192 (4 Hz when RTCCLK = 32768 Hz)  
 0x3: RTCCLK / 4096 (8 Hz when RTCCLK = 32768 Hz)  
 0x4: RTCCLK / 2048 (16 Hz when RTCCLK = 32768 Hz)  
 0x5: RTCCLK / 1024 (32 Hz when RTCCLK = 32768 Hz)  
 0x6: RTCCLK / 512 (64 Hz when RTCCLK = 32768 Hz)  
 0x7: RTCCLK / 256 (128 Hz when RTCCLK = 32768 Hz)
- Bit 7 **TAMPTS**: Activate timestamp on tamper detection event  
 0: Tamper detection event does not cause a timestamp to be saved  
 1: Save timestamp on tamper detection event  
 TAMPTS is valid even if TSE=0 in the RTC\_CR register.
- Bit 6 **TAMP3TRG**: Active level for RTC\_TAMP3 input  
 if TAMPFLT ≠ 00:  
 0: RTC\_TAMP3 input staying low triggers a tamper detection event.  
 1: RTC\_TAMP3 input staying high triggers a tamper detection event.  
 if TAMPFLT = 00:  
 0: RTC\_TAMP3 input rising edge triggers a tamper detection event.  
 1: RTC\_TAMP3 input falling edge triggers a tamper detection event.

*Note: The Tamper 3 falling edge detection is not allowed when switch to VBAT is used, otherwise a detection would always occurs when entering in Vbat mode.*

Bit 5 **TAMP3E**: RTC\_TAMP3 detection enable

- 0: RTC\_TAMP3 input detection disabled
- 1: RTC\_TAMP3 input detection enabled

Bit 4 **TAMP2TRG**: Active level for RTC\_TAMP2 input

if TAMPFLT != 00:

- 0: RTC\_TAMP2 input staying low triggers a tamper detection event.
  - 1: RTC\_TAMP2 input staying high triggers a tamper detection event.
- if TAMPFLT = 00:
- 0: RTC\_TAMP2 input rising edge triggers a tamper detection event.
  - 1: RTC\_TAMP2 input falling edge triggers a tamper detection event.

Bit 3 **TAMP2E**: RTC\_TAMP2 input detection enable

- 0: RTC\_TAMP2 detection disabled
- 1: RTC\_TAMP2 detection enabled

Bit 2 **TAMPIE**: Tamper interrupt enable

- 0: Tamper interrupt disabled
- 1: Tamper interrupt enabled.

*Note: This bit enables the interrupt for all tamper pins events, whatever TAMPxIE level. If this bit is cleared, each tamper event interrupt can be individually enabled by setting TAMPxIE.*

Bit 1 **TAMP1TRG**: Active level for RTC\_TAMP1 input

If TAMPFLT != 00

- 0: RTC\_TAMP1 input staying low triggers a tamper detection event.
- 1: RTC\_TAMP1 input staying high triggers a tamper detection event.

if TAMPFLT = 00:

- 0: RTC\_TAMP1 input rising edge triggers a tamper detection event.
- 1: RTC\_TAMP1 input falling edge triggers a tamper detection event.

Bit 0 **TAMP1E**: RTC\_TAMP1 input detection enable

- 0: RTC\_TAMP1 detection disabled
- 1: RTC\_TAMP1 detection enabled

**Caution:** When TAMPFLT = 0, TAMPxIE must be reset when TAMPxTRG is changed to avoid spuriously setting TAMxF.

### 29.6.17 RTC alarm A sub second register (RTC\_ALRMASSR)

This register can be written only when ALRAE is reset in RTC\_CR register, or in initialization mode.

This register is write protected. The write access procedure is described in [RTC register write protection on page 910](#)

Address offset: 0x44

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	MASKSS[3:0]				Res.							
				rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SS[14:0]														
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	w	rw	rw

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 MASKSS[3:0]: Mask the most-significant bits starting at this bit

0: No comparison on sub seconds for Alarm A. The alarm is set when the seconds unit is incremented (assuming that the rest of the fields match).

1: SS[14:1] are don't care in Alarm A comparison. Only SS[0] is compared.

2: SS[14:2] are don't care in Alarm A comparison. Only SS[1:0] are compared.

3: SS[14:3] are don't care in Alarm A comparison. Only SS[2:0] are compared.

...

12: SS[14:12] are don't care in Alarm A comparison. SS[11:0] are compared.

13: SS[14:13] are don't care in Alarm A comparison. SS[12:0] are compared.

14: SS[14] is don't care in Alarm A comparison. SS[13:0] are compared.

15: All 15 SS bits are compared and must match to activate alarm.

The overflow bits of the synchronous counter (bits 15) is never compared. This bit can be different from 0 only after a shift operation.

Bits 23:15 Reserved, must be kept at reset value.

Bits 14:0 SS[14:0]: Sub seconds value

This value is compared with the contents of the synchronous prescaler counter to determine if Alarm A is to be activated. Only bits 0 up MASKSS-1 are compared.

### 29.6.18 RTC alarm B sub second register (RTC\_ALRMBSSR)

This register can be written only when ALRBE is reset in RTC\_CR register, or in initialization mode.

This register is write protected. The write access procedure is described in [Section : RTC register write protection](#).

Address offset: 0x48

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	MASKSS[3:0]				Res.							
				rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SS[14:0]														
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	w	rw	rw

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 **MASKSS[3:0]**: Mask the most-significant bits starting at this bit

0x0: No comparison on sub seconds for Alarm B. The alarm is set when the seconds unit is incremented (assuming that the rest of the fields match).

0x1: SS[14:1] are don't care in Alarm B comparison. Only SS[0] is compared.

0x2: SS[14:2] are don't care in Alarm B comparison. Only SS[1:0] are compared.

0x3: SS[14:3] are don't care in Alarm B comparison. Only SS[2:0] are compared.

...

0xC: SS[14:12] are don't care in Alarm B comparison. SS[11:0] are compared.

0xD: SS[14:13] are don't care in Alarm B comparison. SS[12:0] are compared.

0xE: SS[14] is don't care in Alarm B comparison. SS[13:0] are compared.

0xF: All 15 SS bits are compared and must match to activate alarm.

The overflow bits of the synchronous counter (bits 15) is never compared. This bit can be different from 0 only after a shift operation.

Bits 23:15 Reserved, must be kept at reset value.

Bits 14:0 **SS[14:0]**: Sub seconds value

This value is compared with the contents of the synchronous prescaler counter to determine if Alarm B is to be activated. Only bits 0 up to MASKSS-1 are compared.

### 29.6.19 RTC option register (RTC\_OR)

Address offset: 0x4C

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RTC_ALARM_TYPE	TSINSEL[1:0]	Res.												
												rw	rw	rw	

Bits 31:4 Reserved, must be kept at reset value.

Bit 3 **RTC\_ALARM\_TYPE**: RTC\_ALARM on PC13 output type

0: RTC\_ALARM, when mapped on PC13, is open-drain output

1: RTC\_ALARM, when mapped on PC13, is push-pull output

Bits 2:1 **TSINSEL[1:0]**: TIMESTAMP mapping

00: TIMESTAMP is mapped on PC13

01: TIMESTAMP is mapped on PI8

10: TIMESTAMP is mapped on PC1

11: TIMESTAMP is mapped on PC1

Bit 0 Reserved, must be kept at reset value.

### 29.6.20 RTC backup registers (RTC\_BKPxR)

Address offset: 0x50 to 0xCC

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BKP[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BKP[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	w	rw	rw

Bits 31:0 BKP[31:0]

The application can write or read data to and from these registers.

They are powered-on by  $V_{BAT}$  when  $V_{DD}$  is switched off, so that they are not reset by System reset, and their contents remain valid when the device operates in low-power mode.

This register is reset on a tamper detection event, as long as TAMPxF=1.

## 29.6.21 RTC register map

**Table 178. RTC register map and reset values**

**Table 178. RTC register map and reset values (continued)**

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 30 Inter-integrated circuit interface (I2C)

### 30.1 Introduction

The I2C peripheral handles the interface between the device and the serial I<sup>2</sup>C (inter-integrated circuit) bus. It provides multicontroller 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+).

The I2C peripheral is also SMBus (system management bus) and PMBus® (power management bus) compatible.

It can use DMA to reduce the CPU load.

### 30.2 I2C main features

- I<sup>2</sup>C-bus specification rev03 compatibility:
  - Target and controller modes
  - Multicontroller 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 target 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
  - Clock stretching (optional)
- 1-byte buffer with DMA capability
- Programmable analog and digital noise filters
- SMBus specification rev 3.0 compatibility<sup>(a)</sup>:
  - Hardware PEC (packet error checking) generation and verification with ACK control
  - Command and data acknowledge control
  - Address resolution protocol (ARP) support
  - Host and device support
  - SMBus alert
  - Timeouts and idle condition detection
- PMBus rev 1.3 standard compatibility
- Independent clock

---

a. To check the compliance of the GPIOs selected for SMBus with the specified logical levels, refer to the product datasheet.

For information on I2C instantiation, refer to [Section 30.3: I2C implementation](#).

### 30.3 I2C implementation

This section provides an implementation overview with respect to the I2C instantiation.

**Table 179. I2C implementation**

I2C features <sup>(1)</sup>	I2C1	I2C2	I2C3	I2C4
7-bit addressing mode	X	X	X	X
10-bit addressing mode	X	X	X	X
Standard-mode (up to 100 kbit/s)	X	X	X	X
Fast-mode (up to 400 kbit/s)	X	X	X	X
Fast-mode Plus with 20 mA output drive I/Os (up to 1 Mbit/s)	X	X	X	X
Independent clock	X	X	X	X
Wake-up from Stop mode	-	-	-	-
SMBus/PMBus	X	X	X	X

1. X = supported.

### 30.4 I2C functional description

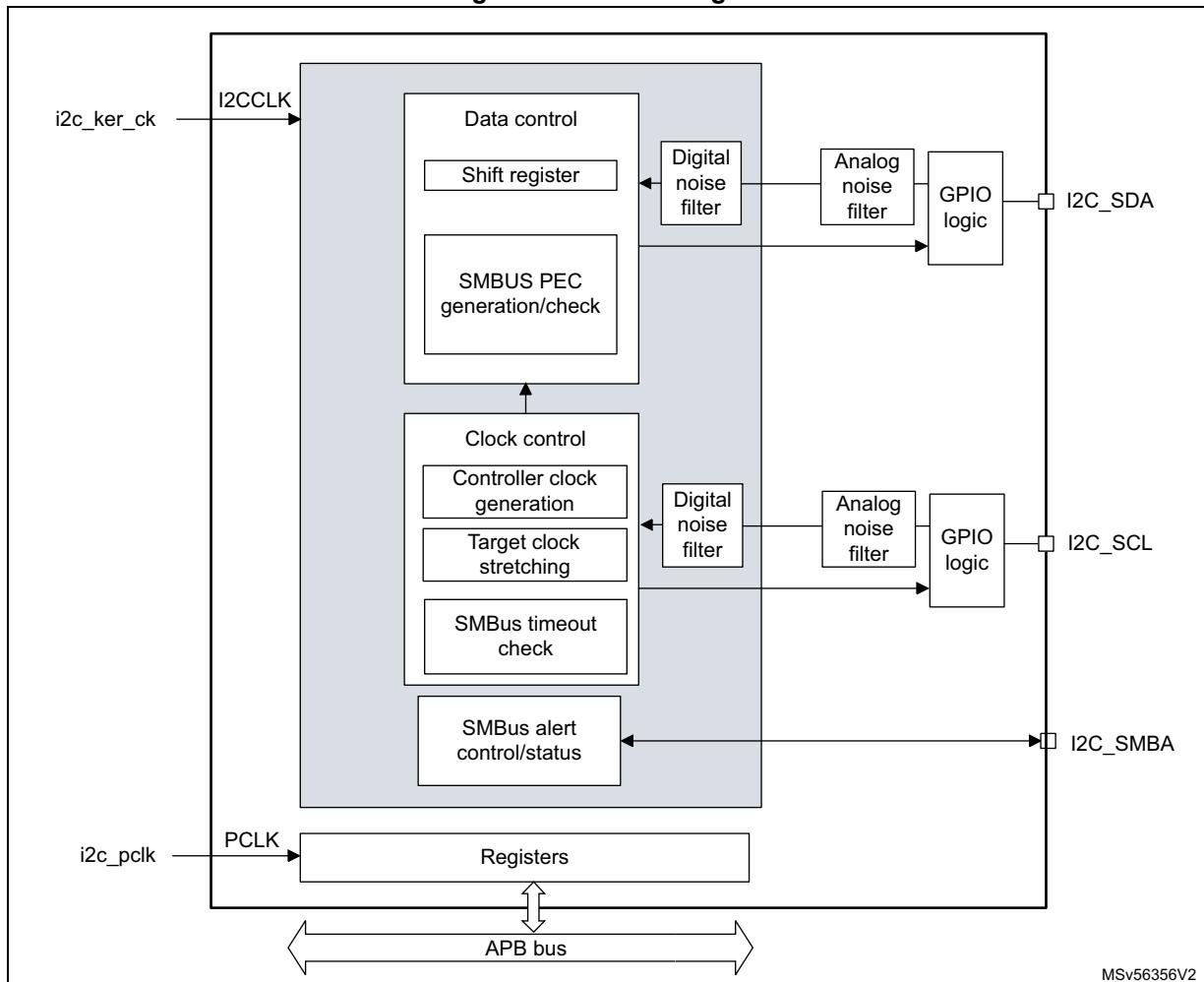
In addition to receiving and transmitting data, the peripheral converts them from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The peripheral is connected to the I<sup>2</sup>C-bus through a data pin (SDA) and a clock pin (SCL). It supports Standard-mode (up to 100 kHz), Fast-mode (up to 400 kHz), and Fast-mode Plus (up to 1 MHz) I<sup>2</sup>C-bus.

The peripheral can also be connected to an SMBus, through the data pin (SDA), the clock pin (SCL), and an optional SMBus alert pin (SMBA).

The independent clock function allows the I2C communication speed to be independent of the PCLK1 frequency.

### 30.4.1 I2C block diagram

Figure 299. Block diagram



### 30.4.2 I2C pins and internal signals

Table 180. I2C input/output pins

Pin name	Signal type	Description
I <sup>2</sup> C_SDA	Bidirectional	I <sup>2</sup> C-bus data
I <sup>2</sup> C_SCL	Bidirectional	I <sup>2</sup> C-bus clock
I <sup>2</sup> C_SMBA	Bidirectional	SMBus alert

Table 181. I2C internal input/output signals

Internal signal name	Signal type	Description
i2c_ker_ck	Input	I <sup>2</sup> C kernel clock, also named I <sup>2</sup> CCLK in this document
i2c_pclk	Input	I <sup>2</sup> C APB clock

**Table 181. I2C internal input/output signals (continued)**

Internal signal name	Signal type	Description
i2c_it	Output	I2C interrupts, refer to <a href="#">Table 195</a> for the list of interrupt sources
i2c_rx_dma	Output	I2C receive data DMA request (I2C_RX)
i2c_tx_dma	Output	I2C transmit data DMA request (I2C_TX)

### 30.4.3 I2C clock requirements

The I2C kernel is clocked by I2CCLK.

The I2CCLK period  $t_{I2CCLK}$  must respect the following conditions:

$$t_{I2CCLK} < (t_{LOW} - t_{filters}) / 4$$

$$t_{I2CCLK} < t_{HIGH}$$

where  $t_{LOW}$  is the SCL low time,  $t_{HIGH}$  is the SCL high time, and  $t_{filters}$  is the sum of the analog and digital filter delays (when enabled).

The digital filter delay is  $DNF[3:0] \times t_{I2CCLK}$ .

The PCLK1 clock period  $t_{PCLK}$  must respect the condition  $t_{PCLK} < 4/3 t_{SCL}$ , where  $t_{SCL}$  is the SCL period.

**Caution:** When the I2C kernel is clocked by PCLK1, this clock must respect the conditions for  $t_{I2CCLK}$ .

### 30.4.4 I2C mode selection

The peripheral can operate as:

- Target transmitter
- Target receiver
- Controller transmitter
- Controller receiver

By default, the peripheral operates in target mode. It automatically switches from target to controller mode upon generating START condition, and from controller to target mode upon arbitration loss or upon generating STOP condition. This allows the use of the I2C peripheral in a multicontroller I2C-bus environment.

#### Communication flow

In controller mode, the I2C peripheral initiates a data transfer and generates the clock signal. Serial data transfers always begin with a START condition and end with a STOP condition. Both START and STOP conditions are generated in controller mode by software.

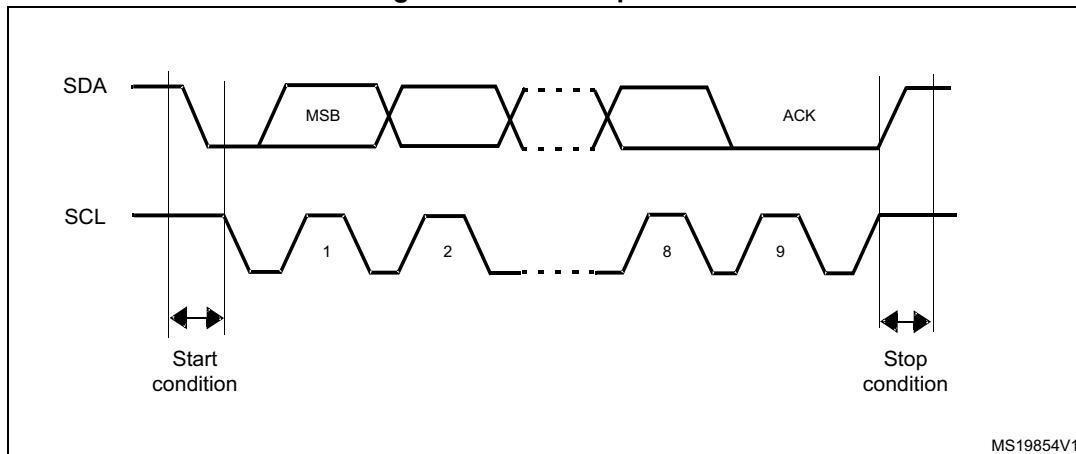
In target mode, the peripheral recognizes its own 7-bit or 10-bit address, and the general call address. The general call address detection can be enabled or disabled by software. The reserved SMBus addresses can also be enabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The address is contained in the first byte (7-bit addressing) or in the first two bytes (10-bit addressing) following the START condition. The address is always transmitted in controller mode.

The following figure shows the transmission of a single byte. The controller generates nine SCL pulses. The transmitter sends the eight data bits to the receiver with the SCL pulses 1

to 8. Then the receiver sends the acknowledge bit to the transmitter with the ninth SCL pulse.

**Figure 300. I<sup>2</sup>C-bus protocol**



The acknowledge can be enabled or disabled by software. The own addresses of the I<sup>2</sup>C peripheral can be selected by software.

### 30.4.5 I<sup>2</sup>C initialization

#### Enabling and disabling the peripheral

Before enabling the I<sup>2</sup>C peripheral, configure and enable its clock through the RCC, and initialize its control registers.

The I<sup>2</sup>C peripheral can then be enabled by setting the PE bit of the I<sup>2</sup>C\_CR1 register.

Disabling the I<sup>2</sup>C peripheral by clearing the PE bit resets the I<sup>2</sup>C peripheral. Refer to [Section 30.4.6](#) for more details.

#### Noise filters

Before enabling the I<sup>2</sup>C peripheral by setting the PE bit of the I<sup>2</sup>C\_CR1 register, the user must configure the analog and/or digital noise filters, as required.

The analog noise filter on the SDA and SCL inputs complies with the I<sup>2</sup>C-bus specification which requires, in Fast-mode and Fast-mode Plus, the suppression of spikes shorter than 50 ns. Enabled by default, it can be disabled by setting the ANFOFF bit.

The digital filter is controlled through the DNF[3:0] bitfield of the I<sup>2</sup>C\_CR1 register. When it is enabled, the internal SCL and SDA signals only take the level of their corresponding I<sup>2</sup>C-bus line when remaining stable for more than DNF[3:0] periods of I<sup>2</sup>CCLK. This allows suppressing spikes shorter than the filtering capacity period programmable from one to fifteen I<sup>2</sup>CCLK periods.

The following table compares the two filters.

**Table 182. Comparison of analog and digital filters**

Item	Analog filter	Digital filter
Filtering capacity <sup>(1)</sup>	$\geq 50$ ns	One to fifteen I <sup>2</sup> CCLK periods (programmable)

1. Maximum duration of spikes that the filter can suppress

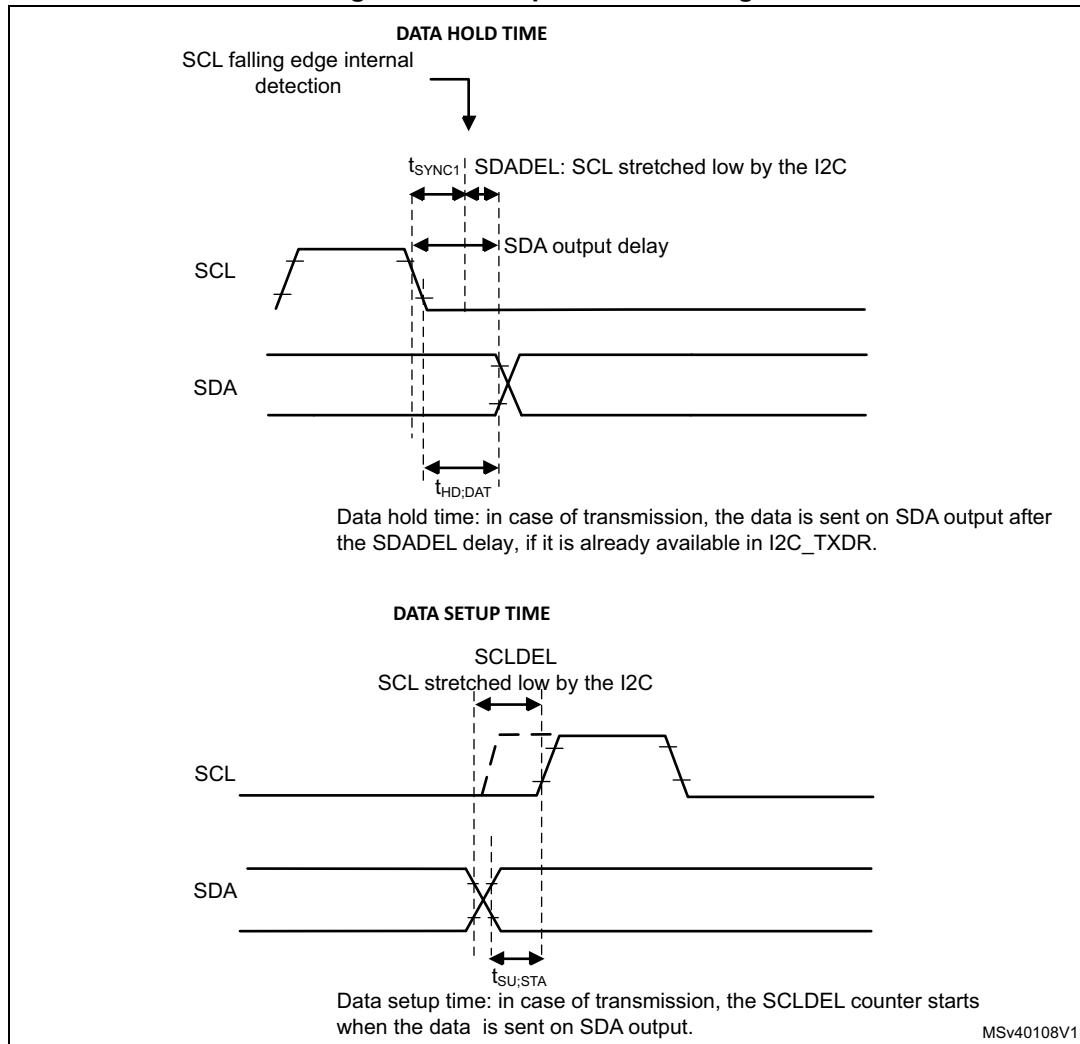
**Caution:** The filter configuration cannot be changed when the I2C peripheral is enabled.

### I2C timings

To ensure correct data hold and setup times, the corresponding timings must be configured through the PRESC[3:0], SCLDEL[3:0], and SDADEL[3:0] bitfields of the I2C\_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C\_TIMINGR content in the *I2C configuration* window.

**Figure 301. Setup and hold timings**



When the SCL falling edge is internally detected, the delay  $t_{SDADEL}$  (impacting the hold time  $t_{HD;DAT}$ ) is inserted before sending SDA output:

$$t_{SDADEL} = SDADEL \times t_{PRESC} + t_{I2CCLK}, \text{ where } t_{PRESC} = (PRESC + 1) \times t_{I2CCLK}.$$

The total SDA output delay is:

$$t_{SYNC1} + \{[SDADEL \times (PRESC + 1) + 1] \times t_{I2CCLK}\}$$

The  $t_{SYNC1}$  duration depends upon:

- SCL falling slope
- input delay  $t_{AF(min)} < t_{AF} < t_{AF(max)}$  introduced by the analog filter (if enabled)
- input delay  $t_{DNF} = DNF \times t_{I2CCLK}$  introduced by the digital filter (if enabled)
- delay due to SCL synchronization to I2CCLK clock (two to three I2CCLK periods)

To bridge the undefined region of the SCL falling edge, the user must set SDADEL[3:0] so as to fulfill the following condition:

$$\{t_f(max) + t_{HD;DAT}(min) - t_{AF(min)} - [(DNF + 3) \times t_{I2CCLK}]\} / \{(PRESC + 1) \times t_{I2CCLK}\} \leq SDADEL$$

$$SDADEL \leq \{t_{HD;DAT}(max) - t_{AF(max)} - [(DNF + 4) \times t_{I2CCLK}]\} / \{(PRESC + 1) \times t_{I2CCLK}\}$$

**Note:**  $t_{AF(min)}$  and  $t_{AF(max)}$  are only part of the condition when the analog filter is enabled. Refer to the device datasheet for  $t_{AF}$  values.

The  $t_{HD;DAT}$  time can at maximum be 3.45  $\mu s$  for Standard-mode, 0.9  $\mu s$  for Fast-mode, and 0.45  $\mu s$  for Fast-mode Plus. It must be lower 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. When it stretches 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. The previous condition then becomes:

$$SDADEL \leq \{t_{VD;DAT}(max) - t_r(max) - t_{AF(max)} - [(DNF + 4) \times t_{I2CCLK}]\} / \{(PRESC + 1) \times t_{I2CCLK}\}$$

**Note:** This condition can be violated when  $NOSTRETCH = 0$ , because the device stretches SCL low to guarantee the set-up time, according to the  $SCLDEL[3:0]$  value.

After  $t_{SDADEL}$ , or after sending SDA output when the target had to stretch the clock because the data was not yet written in I2C\_TXDR register, the 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  $t_{SU;DAT}$ .

To bridge the undefined region of the SDA transition (rising edge usually worst case), the user must program  $SCLDEL[3:0]$  so as to fulfill the following condition:

$$\{[t_r(max) + t_{SU;DAT}(min)] / \{(PRESC + 1) \times t_{I2CCLK}\} - 1\} \leq SCLDEL$$

Refer to the following table for  $t_f$ ,  $t_r$ ,  $t_{HD;DAT}$ ,  $t_{VD;DAT}$ , and  $t_{SU;DAT}$  standard values.

Use the SDA and SCL real transition time values measured in the application to widen the scope of allowed SDADEL[3:0] and  $SCLDEL[3:0]$  values. Use the maximum SDA and SCL transition time values defined in the standard to make the device work reliably regardless of the application.

**Note:** At every clock pulse, after SCL falling edge detection, I2C operating as controller or target stretches SCL low during at least  $[(SDADEL + SCLDEL + 1) \times (PRESC + 1) + 1] \times t_{I2CCLK}$ , in both transmission and reception modes. In transmission mode, if the data is not yet written in I2C\_TXDR when SDA delay elapses, the I2C peripheral keeps 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.

When the NOSTRETCH bit is set in target mode, the SCL is not stretched. The SDADEL[3:0] must then be programmed so that it ensures a sufficient setup time.

Table 183. I<sup>2</sup>C-bus and SMBus specification data setup and hold times

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBus		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
t <sub>HD;DAT</sub>	Data hold time	0	-	0	-	0	-	0.3	-	μs
t <sub>VD;DAT</sub>	Data valid time	-	3.45	-	0.9	-	0.45	-	-	
t <sub>SU;DAT</sub>	Data setup time	250	-	100	-	50	-	250	-	ns
t <sub>r</sub>	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	
t <sub>f</sub>	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	

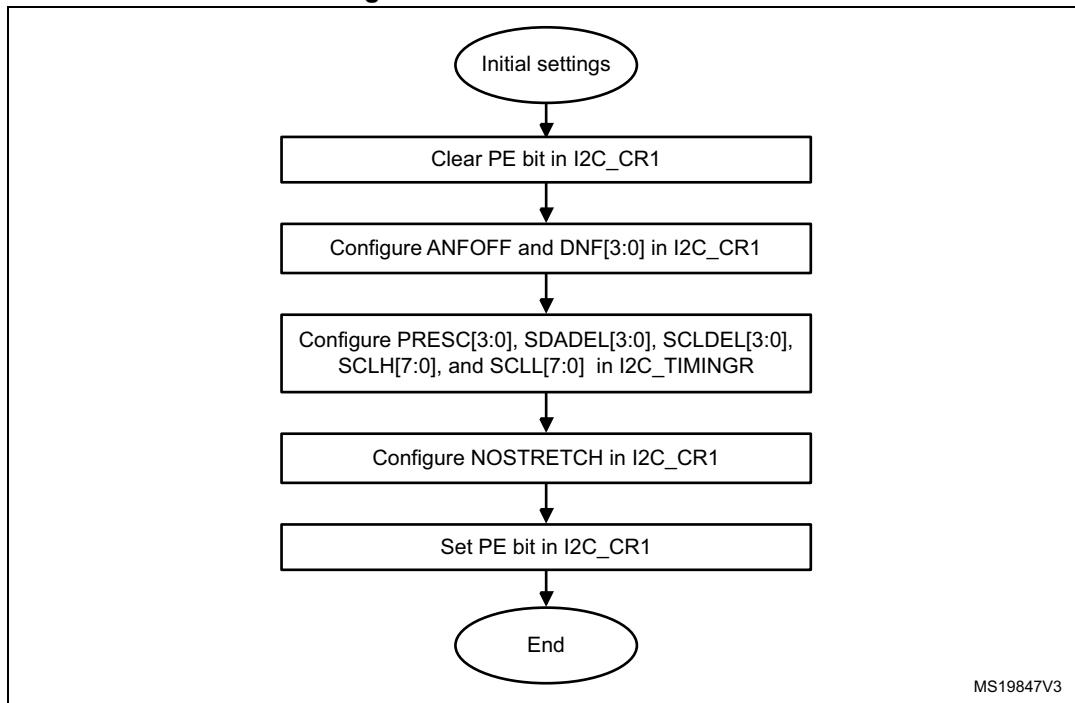
Additionally, in controller mode, the SCL clock high and low levels must be configured by programming the PRESC[3:0], SCLH[7:0], and SCLL[7:0] bitfields of the I2C\_TIMINGR register.

When the SCL falling edge is internally detected, the I2C peripheral releasing the SCL output after the delay  $t_{SCLL} = (SCLL + 1) \times t_{PRESC}$ , where  $t_{PRESC} = (PRESC + 1) \times t_{I2CCLK}$ . The  $t_{SCLL}$  delay impacts the SCL low time  $t_{LOW}$ .

When the SCL rising edge is internally detected, the I2C peripheral forces the SCL output to low level after the delay  $t_{SCLH} = (SCLH + 1) \times t_{PRESC}$ , where  $t_{PRESC} = (PRESC + 1) \times t_{I2CCLK}$ . The  $t_{SCLH}$  impacts the SCL high time  $t_{HIGH}$ .

Refer to [I2C controller initialization](#) for more details.

**Caution:** Changing the timing configuration and the NOSTRETCH configuration is not allowed when the I2C peripheral is enabled. Like the timing settings, the target NOSTRETCH settings must also be done before enabling the peripheral. Refer to [I2C target initialization](#) for more details.

**Figure 302. I2C initialization flow**

### 30.4.6 I2C reset

The reset of the I2C peripheral is performed by clearing the PE bit of the I2C\_CR1 register. It has the effect of releasing the SCL and SDA lines. Internal state machines are reset and the communication control bits and the status bits revert to their reset values. This reset does not impact the configuration registers.

The impacted register bits are:

1. I2C\_CR2 register: START, STOP, PECBYTE, and NACK
2. I2C\_ISR register: BUSY, TXE, TXIS, RXNE, ADDR, NACKF, TCR, TC, STOPF, BERR, ARLO, PECERR, TIMEOUT, ALERT, and OVR

PE must be kept low during at least three APB clock cycles to perform the I2C reset. To ensure this, perform the following software sequence:

1. Write PE = 0
2. Check PE = 0
3. Write PE = 1

### 30.4.7 I2C data transfer

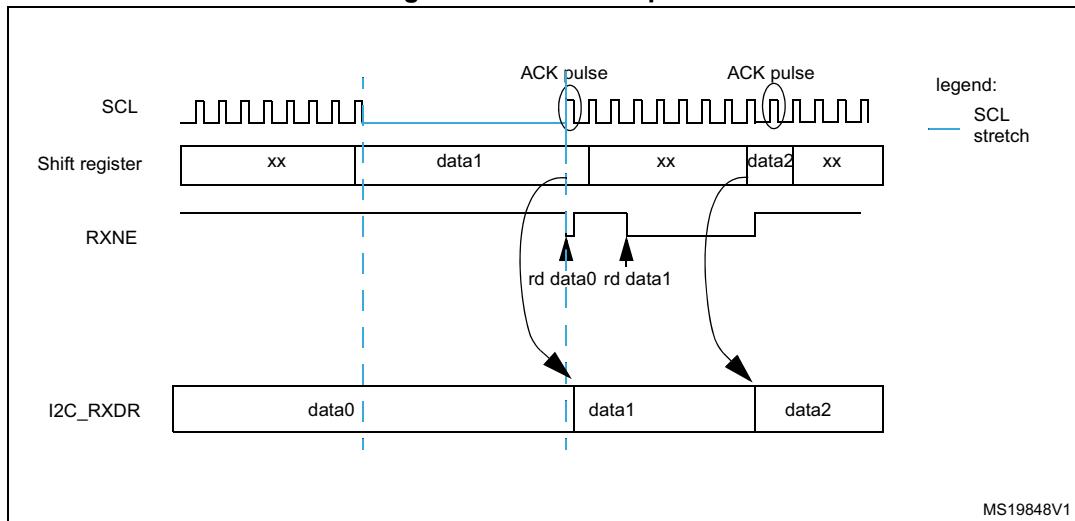
The data transfer is managed through transmit and receive data registers and a shift register.

#### Reception

The SDA input fills the shift register. After the eighth SCL pulse (when the complete data byte is received), the shift register is copied into the I2C\_RXDR register if it is empty (RXNE = 0). If RXNE = 1, which means that the previous received data byte has not yet

been read, the SCL line is stretched low until I2C\_RXDR is read. The stretch occurs between the eighth and the ninth SCL pulse (before the acknowledge pulse).

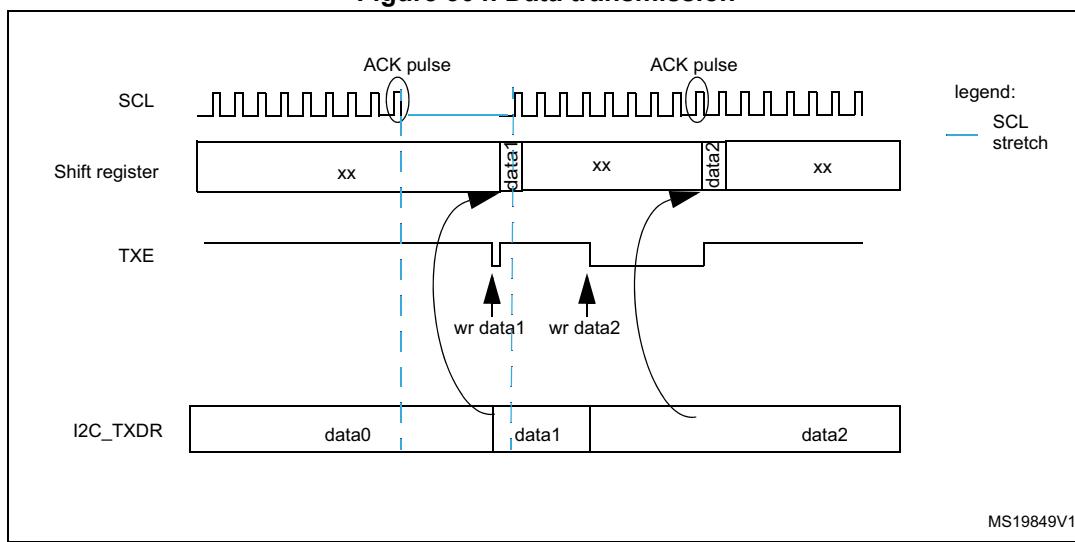
**Figure 303. Data reception**



### Transmission

If the I2C\_TXDR register is not empty (TXE = 0), its content is copied into the shift register after the ninth SCL pulse (the acknowledge pulse). Then the shift register content is shifted out on the SDA line. If TXE = 1, which means that no data is written yet in I2C\_TXDR, the SCL line is stretched low until I2C\_TXDR is written. The stretch starts after the ninth SCL pulse.

**Figure 304. Data transmission**



## Hardware transfer management

The I2C features an embedded byte counter to manage byte transfer and to close the communication in various modes, such as:

- NACK, STOP and ReSTART generation in controller mode
- ACK control in target receiver mode
- PEC generation/checking

In controller mode, the byte counter is always used. By default, it is disabled in target mode. It can be enabled by software, by setting the SBC (target byte control) bit of the I2C\_CR1 register.

The number of bytes to transfer is programmed in the NBYTES[7:0] bitfield of the I2C\_CR2 register. If this number 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 of the I2C\_CR2 register. In this mode, the TCR flag is set when the number of bytes programmed in NBYTES[7:0] is transferred (when the associated counter reaches zero), and an interrupt is generated if TCIE is set. SCL is stretched as long as the TCR flag is set. TCR is cleared by software when NBYTES[7:0] is written to a non-zero value.

When NBYTES[7:0] is reloaded with the last number of bytes to transfer, the RELOAD bit must be cleared.

When RELOAD = 0 in controller mode, the counter can be used in two modes:

- **Automatic end** (AUTOEND = 1 in the I2C\_CR2 register). In this mode, the controller automatically sends a STOP condition once the number of bytes programmed in the NBYTES[7:0] bitfield is transferred.
- **Software end** (AUTOEND = 0 in the I2C\_CR2 register). In this mode, a software action is expected once the number of bytes programmed in the NBYTES[7:0] bitfield is transferred; the TC flag is set and an interrupt is generated if the TCIE bit is set. The SCL signal is stretched as long as the TC flag is set. The TC flag is cleared by software when the START or STOP bit of the I2C\_CR2 register is set. This mode must be used when the controller wants to send a RESTART condition.

**Caution:** The AUTOEND bit has no effect when the RELOAD bit is set.

Table 184. I2C configuration

Function	SBC bit	RELOAD bit	AUTOEND bit
Controller Tx/Rx NBYTES + STOP	X	0	1
Controller Tx/Rx + NBYTES + RESTART	X	0	0
Target Tx/Rx, all received bytes ACKed	0	X	X
Target Rx with ACK control	1	1	X

### 30.4.8 I2C target mode

#### I2C target initialization

To work in target mode, the user must enable at least one target address. The I2C\_OAR1 and I2C\_OAR2 registers are available to program the target own addresses OA1 and OA2, respectively.

OA1 can be configured either in 7-bit (default) or in 10-bit addressing mode, by setting the OA1MODE bit of the I2C\_OAR1 register.

OA1 is enabled by setting the OA1EN bit of the I2C\_OAR1 register.

If an additional target addresses are required, the second target address OA2 can be configured. Up to seven OA2 LSBs can be masked, by configuring the OA2MSK[2:0] bitfield of the I2C\_OAR2 register. Therefore, for OA2MSK[2:0] 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. When OA2MSK[2:0] is other than 0, the address comparator for OA2 excludes the I2C reserved addresses (0000 XXX and 1111 XXX) and they are not acknowledged. If OA2MSK[2:0] = 7, all received 7-bit addresses are acknowledged (except reserved addresses). OA2 is always a 7-bit address.

When enabled through the specific bit, the reserved addresses can be acknowledged if they are programmed in the I2C\_OAR1 or I2C\_OAR2 register with OA2MSK[2:0] = 0.

OA2 is enabled by setting the OA2EN bit of the I2C\_OAR2 register.

The general call address is enabled by setting the GCEN bit of the I2C\_CR1 register.

When the I2C peripheral 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 target uses its clock stretching capability, which means that it stretches the SCL signal at low level when required, to perform software actions. If the controller does not support clock stretching, I2C must be configured with NOSTRETCH = 1 in the I2C\_CR1 register.

After receiving an ADDR interrupt, if several addresses are enabled, the user must read the ADDCODE[6:0] bitfield of the I2C\_ISR register to check which address matched. The DIR flag must also be checked to know the transfer direction.

### Target with clock stretching

As long as the NOSTRETCH bit of the I2C\_CR1 register is zero (default), the I2C peripheral operating as an I<sup>2</sup>C-bus target stretches the SCL signal in the following situations:

- The ADDR flag is set and the received address matches with one of the enabled target addresses.  
The stretch is released when the software clears the ADDR flag by setting the ADDRCF bit.
- In transmission, the previous data transmission is completed and no new data is written in I2C\_TXDR register, or the first data byte is not written when the ADDR flag is cleared (TXE = 1).  
The stretch is released when the data is written to the I2C\_TXDR register.
- In reception, the I2C\_RXDR register is not read yet and a new data reception is completed.  
The stretch is released when I2C\_RXDR is read.
- In target byte control mode (SBC bit set) with reload (RELOAD bit set), the last data byte transfer is finished (TCR bit set).  
The stretch is released when then TCR is cleared by writing a non-zero value in the NBYTES[7:0] bitfield.
- After SCL falling edge detection.  
The stretch is released after [(SDADEL + SCLDEL + 1) x (PRESC+ 1) + 1] x t<sub>I2CCCLK</sub> period.

### Target without clock stretching

As long as the NOSTRETCH bit of the I2C\_CR1 register is set, the I2C peripheral operating as an I<sup>2</sup>C-bus target does not stretch the SCL signal.

The SCL clock is not stretched while the ADDR flag is set.

In transmission, the data must be written in the I2C\_TXDR register before the first SCL pulse corresponding to its transfer occurs. If not, an underrun occurs, the OVR flag is set in the I2C\_ISR register and an interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set. 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, it ensures that the OVR status is provided, even for the first data to be transmitted.

In reception, the data must be read from the I2C\_RXDR register before the ninth SCL pulse (ACK pulse) of the next data byte occurs. If not, an overrun occurs, the OVR flag is set in the I2C\_ISR register, and an interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set.

### Target byte control mode

To allow byte ACK control in target reception mode, the target byte control mode must be enabled, by setting the SBC bit of the I2C\_CR1 register. This is required to comply with SMBus standards.

The reload mode must be selected to allow byte ACK control in target reception mode (RELOAD = 1). To get control of each byte, NBYTES[7:0] must be initialized to 0x1 in the ADDR interrupt subroutine, and reloaded to 0x1 after each received byte. When the byte is received, the TCR bit is set, stretching the SCL signal low between the eighth and the ninth SCL pulse. The user can read the data from the I2C\_RXDR register, and then decide to acknowledge it or not by configuring the ACK bit of the I2C\_CR2 register. The SCL stretch is released by programming NBYTES to a non-zero value: the acknowledge or not-acknowledge is sent and the next byte can be received.

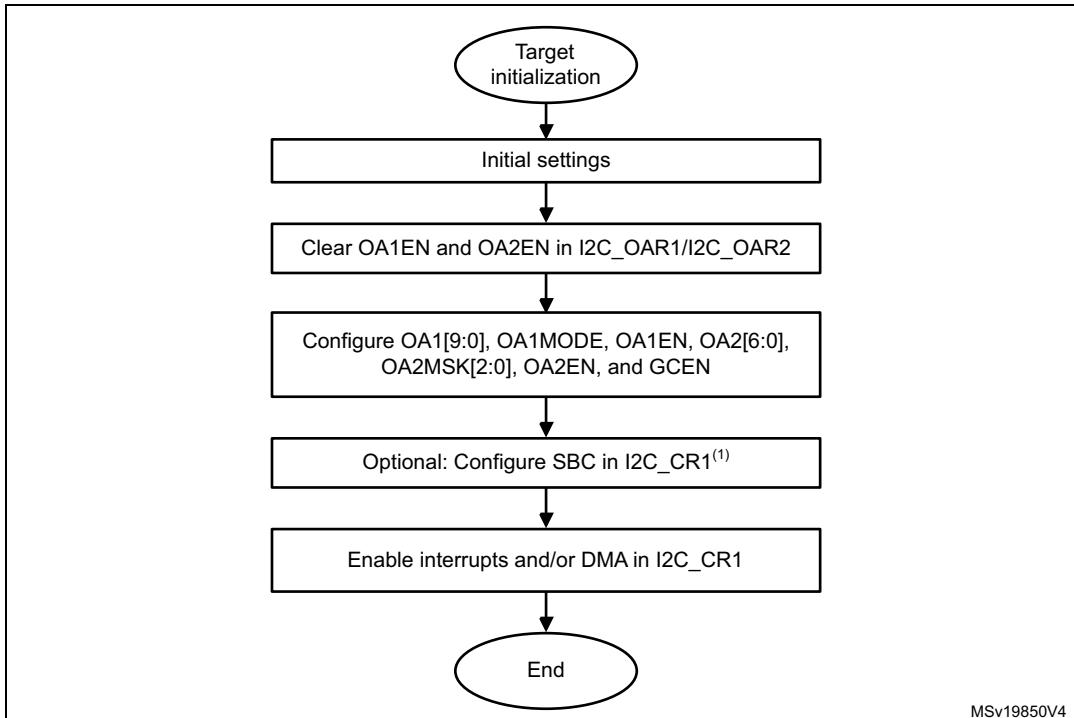
NBYTES[7:0] can be loaded with a value greater than 0x1. Receiving then continues until the corresponding number of bytes are received.

**Note:** *The SBC bit must be configured when the I2C peripheral is disabled, when the target is not addressed, or when ADDR = 1.*

*The RELOAD bit value can be changed when ADDR = 1, or when TCR = 1.*

**Caution:** The target byte control mode is not compatible with NOSTRETCH mode. Setting SBC when NOSTRETCH = 1 is not allowed.

Figure 305. Target initialization flow



1. SBC must be set to support SMBus features.

### Target transmitter

A transmit interrupt status (TXIS) flag is generated when the I2C\_TXDR register becomes empty. An interrupt is generated if the TXIE bit of the I2C\_CR1 register is set.

The TXIS flag is cleared when the I2C\_TXDR register is written with the next data byte to transmit.

When NACK is received, the NACKF flag is set in the I2C\_ISR register and an interrupt is generated if the NACKIE bit of the I2C\_CR1 register is set. The target automatically releases the SCL and SDA lines to let the controller perform a STOP or a RESTART condition. The TXIS bit is not set when a NACK is received.

When STOP is received and the STOPIE bit of the I2C\_CR1 register is set, the STOPF flag of the I2C\_ISR register is set and an interrupt is generated. In most applications, the SBC bit is usually programmed to 0. In this case, if TXE = 0 when the target address is received (ADDR = 1), the user can choose either to send the content of the I2C\_TXDR register as the first data byte, or to flush the I2C\_TXDR register, by setting the TXE bit in order to program a new data byte.

In target byte control mode (SBC = 1), the number of bytes to transmit must be programmed in NBYTES[7:0] 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[7:0].

**Caution:** When NOSTRETCH = 1, the SCL clock is not stretched while the ADDR flag is set, so the user cannot flush the I2C\_TXDR register content in the ADDR subroutine to program the

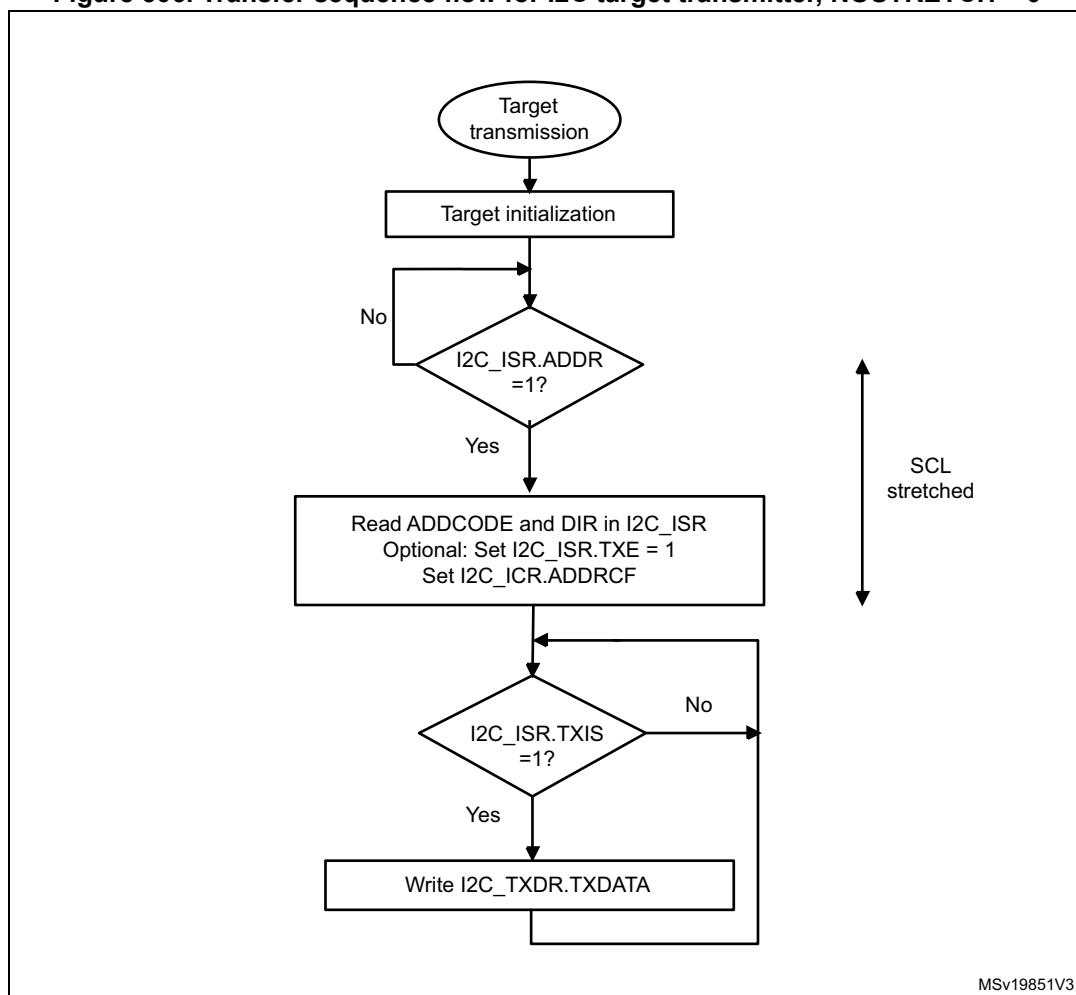
first data byte. The first data byte to send must be previously programmed in the I2C\_TXDR register:

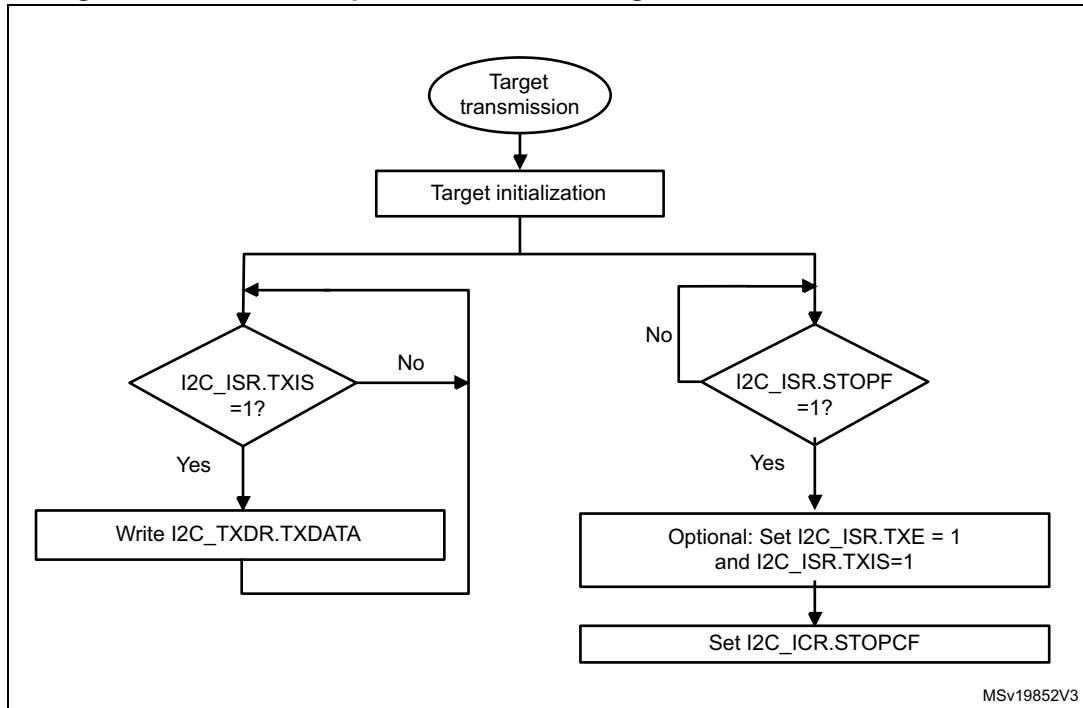
- This data can be the one written in the last TXIS event of the previous transmission message.
- If this data byte is not the one to send, the I2C\_TXDR register can be flushed, by setting the TXE bit, to program a new data byte. The STOPF bit must be cleared only after these actions. This guarantees that they are executed before the first data transmission starts, following the address acknowledge.

If STOPF is still set when the first data transmission starts, an underrun error is generated (the OVR flag is set).

If a TXIS event (transmit interrupt or transmit DMA request) is required, the user must set the TXIS bit in addition to the TXE bit, to generate the event.

**Figure 306. Transfer sequence flow for I2C target transmitter, NOSTRETCH = 0**

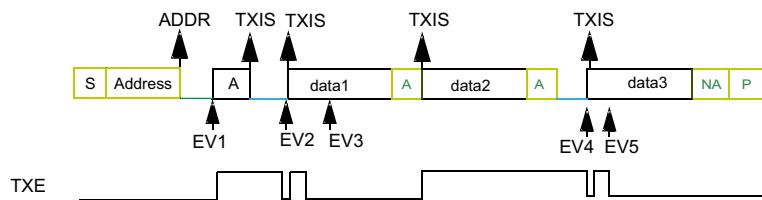


**Figure 307. Transfer sequence flow for I2C target transmitter, NOSTRETCH = 1**

MSv19852V3

**Figure 308. Transfer bus diagrams for I2C target transmitter (mandatory events only)**

Example I2C target transmitter 3 bytes with 1st data flushed,  
NOSTRETCH=0:



legend:

transmission

reception

SCL stretch

EV1: ADDR ISR: check ADDCODE and DIR, set TXE, set ADDRCF

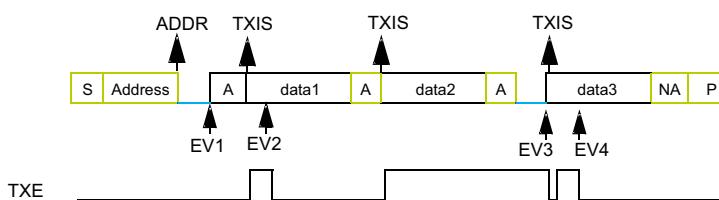
EV2: TXIS ISR: wr data1

EV3: TXIS ISR: wr data2

EV4: TXIS ISR: wr data3

EV5: TXIS ISR: wr data4 (not sent)

Example I2C target transmitter 3 bytes without 1st data flush,  
NOSTRETCH=0:



legend :

transmission

reception

SCL stretch

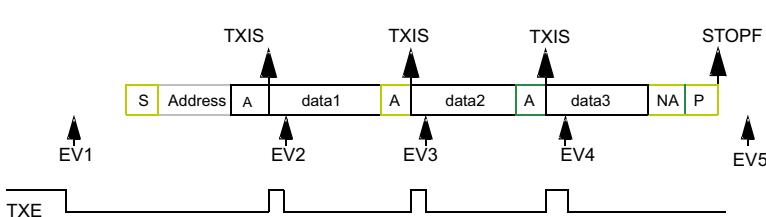
EV1: ADDR ISR: check ADDCODE and DIR, set ADDRCF

EV2: TXIS ISR: wr data2

EV3: TXIS ISR: wr data3

EV4: TXIS ISR: wr data4 (not sent)

Example I2C target transmitter 3 bytes, NOSTRETCH=1:



legend:

transmission

reception

SCL stretch

EV1: wr data1

EV2: TXIS ISR: wr data2

EV3: TXIS ISR: wr data3

EV4: TXIS ISR: wr data4 (not sent)

EV5: STOPF ISR: (optional: set TXE and TXIS), set STOPCF

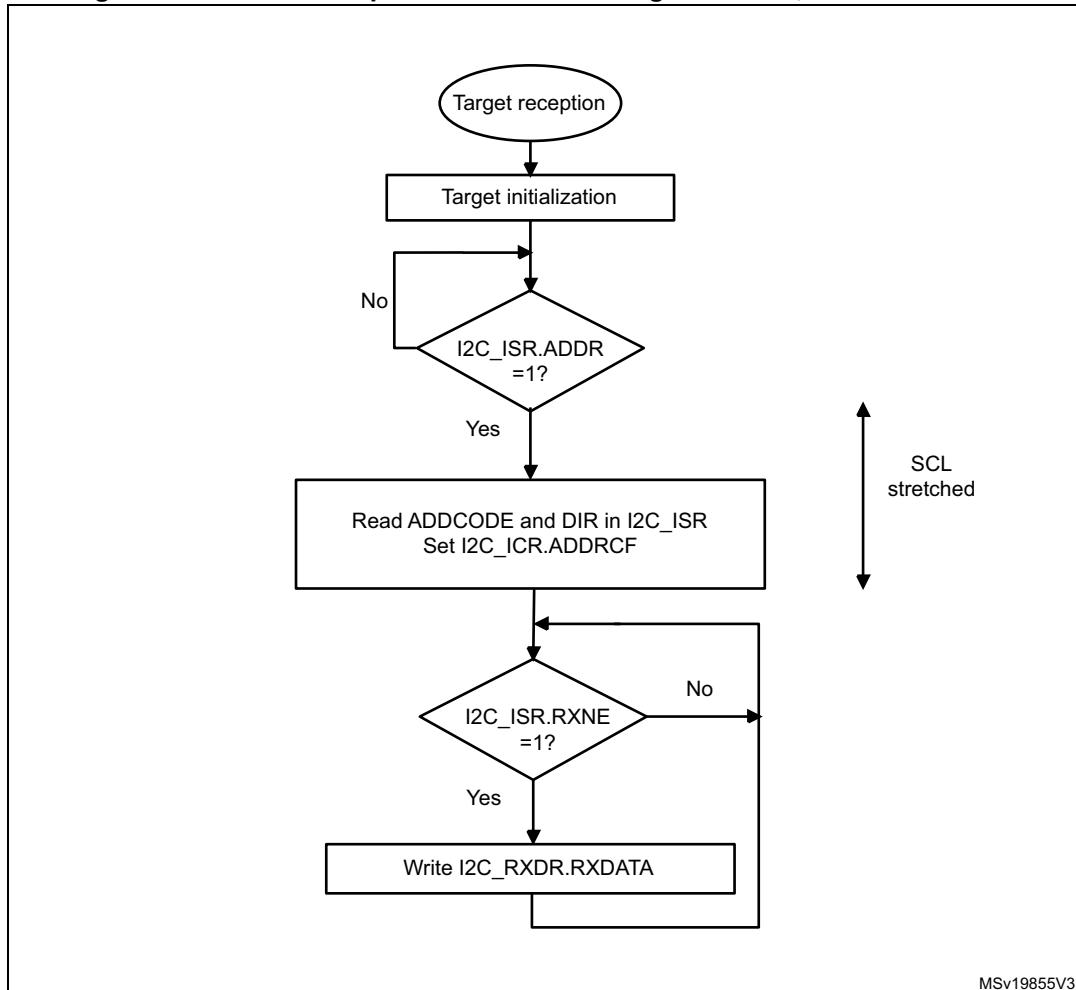
MSv19853V3

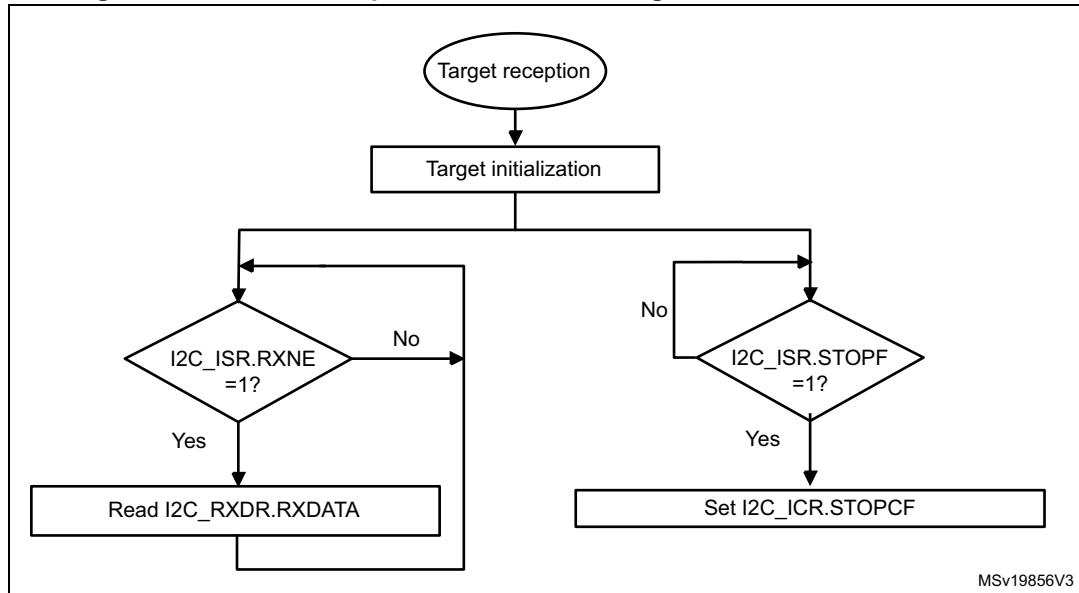
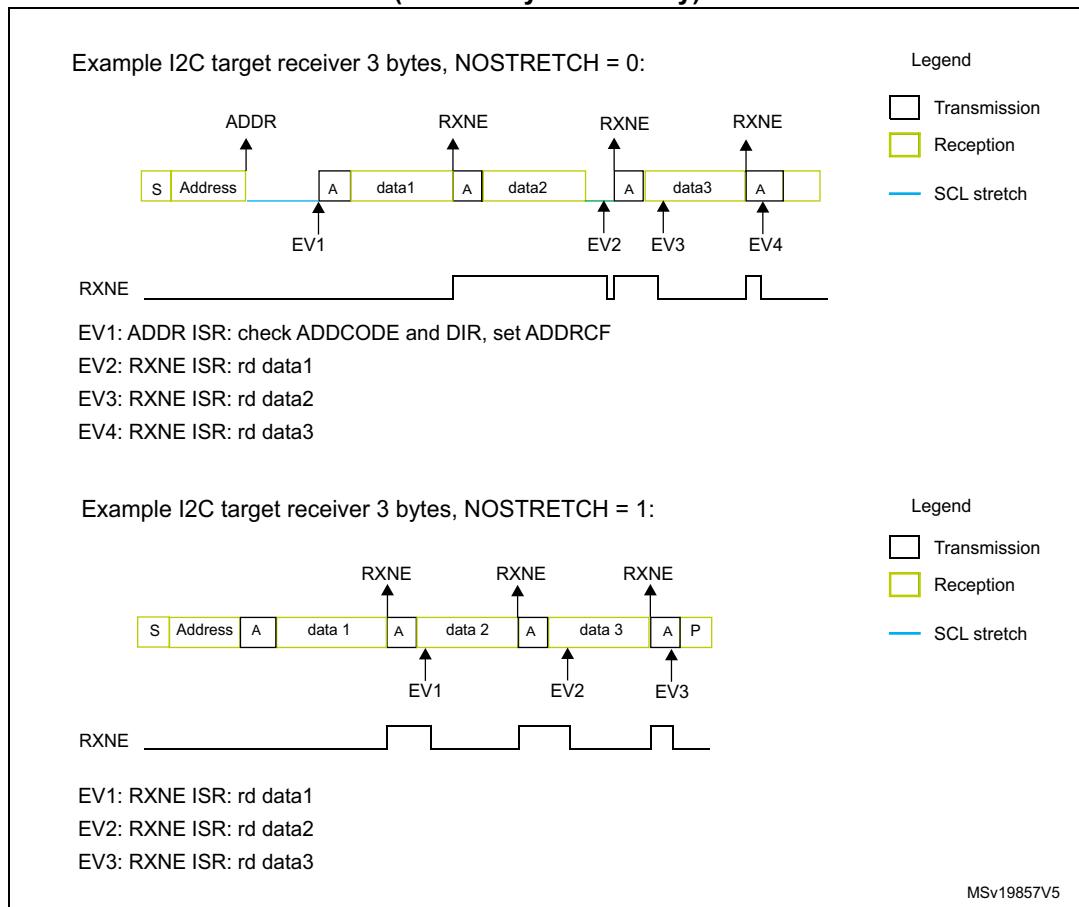
## Target receiver

The RXNE bit of the I2C\_ISR register is set when the I2C\_RXDR is full, which generates an interrupt if the RXIE bit of the I2C\_CR1 register is set. RXNE is cleared when I2C\_RXDR is read.

When STOP condition is received and the STOPIE bit of the I2C\_CR1 register is set, the STOPF flag in the I2C\_ISR register is set and an interrupt is generated.

**Figure 309. Transfer sequence flow for I<sub>2</sub>C target receiver, NOSTRETCH = 0**



**Figure 310. Transfer sequence flow for I2C target receiver, NOSTRETCH = 1****Figure 311. Transfer bus diagrams for I2C target receiver (mandatory events only)**

### 30.4.9 I2C controller mode

#### I2C controller initialization

Before enabling the peripheral, the I2C controller clock must be configured, by setting the SCLH and SCLL bits in the I2C\_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C\_TIMINGR content in the *I2C Configuration* window.

A clock synchronization mechanism is implemented in order to support multicontroller environment and target 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.

I2C detects its own SCL low level after a  $t_{SYNC1}$  delay depending on the SCL falling edge, SCL input noise filters (analog and digital), and SCL synchronization to the I2CxCLK clock. I2C releases SCL to high level once the SCLL counter reaches the value programmed in the SCLL[7:0] bitfield of the I2C\_TIMINGR register.

I2C detects its own SCL high level after a  $t_{SYNC2}$  delay depending on the SCL rising edge, SCL input noise filters (analog and digital), and SCL synchronization to the I2CxCLK clock. I2C ties SCL to low level once the SCLH counter reaches the value programmed in the SCLH[7:0] bitfield of the I2C\_TIMINGR register.

Consequently the controller clock period is:

$$t_{SCL} = t_{SYNC1} + t_{SYNC2} + \{[(SCLH+1) + (SCLL+1)] \times (PRESC+1) \times t_{I2CCLK}\}$$

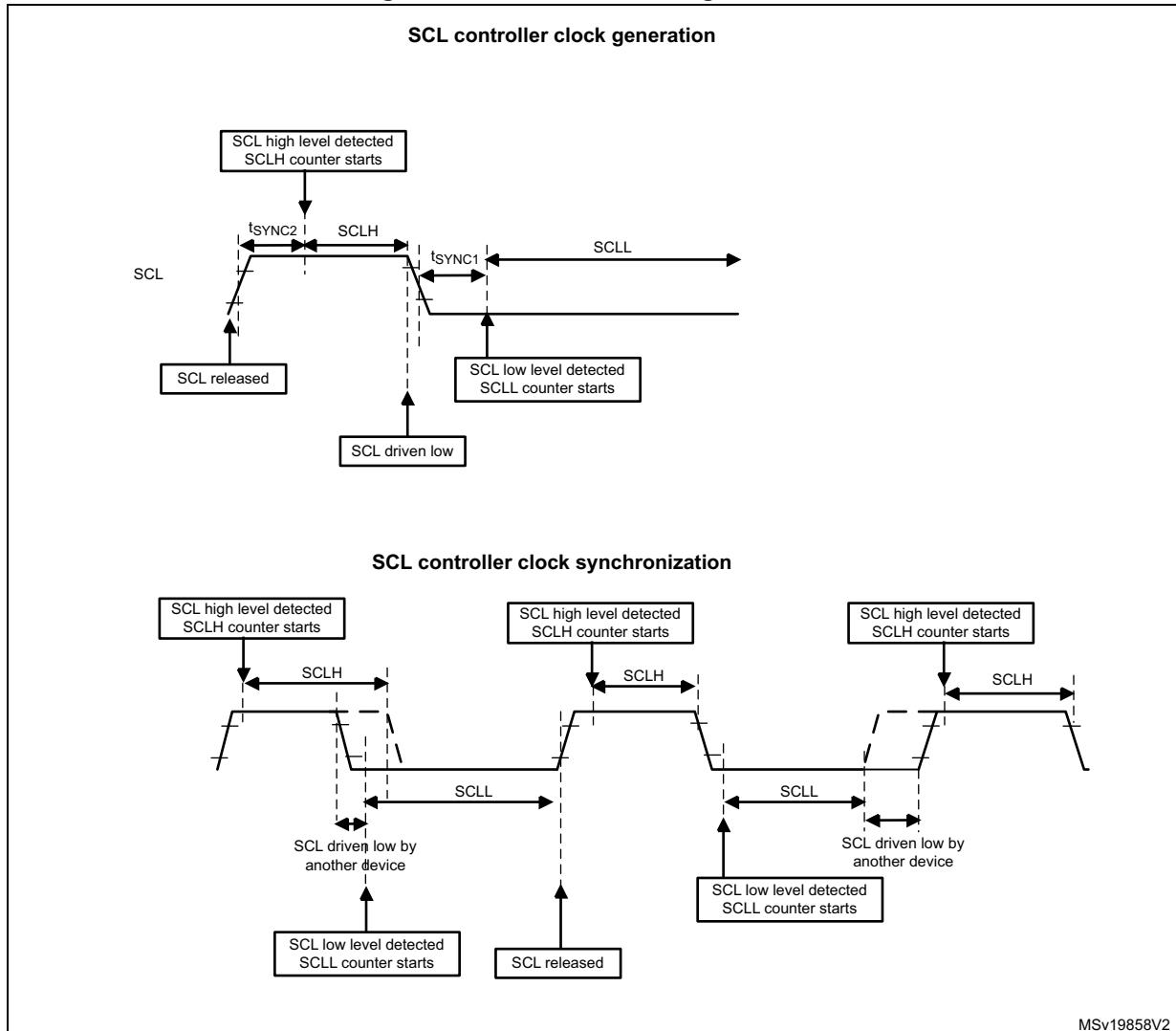
The duration of  $t_{SYNC1}$  depends upon:

- SCL falling slope
- input delay induced by the analog filter (when enabled)
- input delay induced by the digital filter (when enabled): DNF[3:0]  $\times t_{I2CCLK}$
- delay due to SCL synchronization with the I2CCLK clock (two to three I2CCLK periods)

The duration of  $t_{SYNC2}$  depends upon:

- SCL rising slope
- input delay induced by the analog filter (when enabled)
- input delay induced by the digital filter (when enabled): DNF[3:0]  $\times t_{I2CCLK}$
- delay due to SCL synchronization with the I2CCLK clock (two to three I2CCLK periods)

Figure 312. Controller clock generation



**Caution:** For compliance with the I<sup>2</sup>C-bus or SMBus specification, the controller clock must respect the timings in the following table.

Table 185. I<sup>2</sup>C-bus and SMBus specification clock timings

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBus		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
f <sub>SCL</sub>	SCL clock frequency	-	100	-	400	-	1000	-	100	kHz
t <sub>HD:STA</sub>	Hold time (repeated) START condition	4.0	-	0.6	-	0.26	-	4.0	-	μs
t <sub>SU:STA</sub>	Set-up time for a repeated START condition	4.7	-	0.6	-	0.26	-	4.7	-	
t <sub>SU:STO</sub>	Set-up time for STOP condition	4.0	-	0.6	-	0.26	-	4.0	-	
t <sub>BUF</sub>	Bus free time between a STOP and START condition	4.7	-	1.3	-	0.5	-	4.7	-	
t <sub>LOW</sub>	Low period of the SCL clock	4.7	-	1.3	-	0.5	-	4.7	-	
t <sub>HIGH</sub>	High period of the SCL clock	4.0	-	0.6	-	0.26	-	4.0	50	
t <sub>r</sub>	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	ns
t <sub>f</sub>	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	

**Note:** The SCLL[7:0] bitfield also determines the t<sub>BUF</sub> and t<sub>SU:STA</sub> timings and SCLH[7:0] the t<sub>HD:STA</sub> and t<sub>SU:STO</sub> timings.

Refer to [Section 30.4.10](#) for examples of I<sup>2</sup>C\_TIMINGR settings versus the I<sup>2</sup>CCLK frequency.

### Controller communication initialization (address phase)

To initiate the communication with a target to address, set the following bitfields of the I<sup>2</sup>C\_CR2 register:

- ADD10: addressing mode (7-bit or 10-bit)
- SADD[9:0]: target address to send
- RD\_WRN: transfer direction
- HEAD10R: in case of 10-bit address read, this bit determines whether the header only (for direction change) or the complete address sequence is sent.
- NBYTES[7:0]: the number of bytes to transfer; if equal to or greater than 255 bytes, the bitfield must initially be set to 0xFF.

**Note:** Changing these bitfields is not allowed as long as the START bit is set.

Before launching the communication, make sure that the I<sup>2</sup>C-bus is idle. This can be checked using the bus idle detection function or by verifying that the IDR bits of the GPIOs selected as SDA and SCL are set. Any low-level incident on the I<sup>2</sup>C-bus lines that coincides with the START condition asserted by the I<sup>2</sup>C peripheral may cause its deadlock if not filtered out by the input filters. If such incidents cannot be prevented, design the software so that it restores the normal operation of the I<sup>2</sup>C peripheral in case of a deadlock, by toggling the PE bit of the I<sup>2</sup>C\_CR1 register.

To launch the communication, set the START bit of the I2C\_CR2 register. The controller then automatically sends a START condition followed by the target address, either immediately if the BUSY flag is low, or  $t_{BUF}$  time after the BUSY flag transits from high to low state. The BUSY flag is set upon sending the START condition.

In case of an arbitration loss, the controller automatically switches back to target mode and can acknowledge its own address if it is addressed as a target.

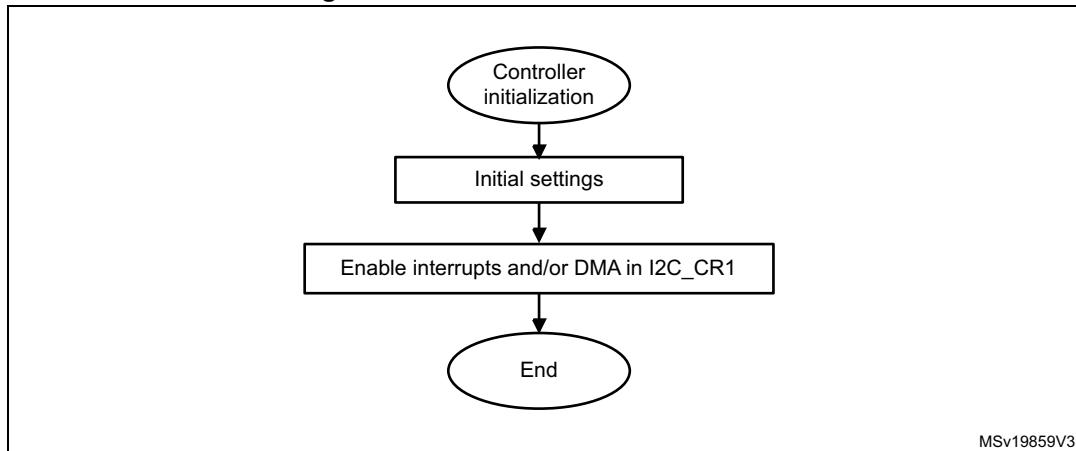
**Note:** *The START bit is reset by hardware when the target address is sent on the bus, whatever the received acknowledge value. The START bit is also reset by hardware upon arbitration loss.*

*In 10-bit addressing mode, the controller automatically keeps resending the target address in a loop until the first address byte (first seven address bits) is acknowledged by the target. Setting the ADDRCF bit makes I2C quit that loop.*

*If the I2C peripheral is addressed as a target (ADDR = 1) while the START bit is set, the I2C peripheral switches to target 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 313. Controller initialization flow**



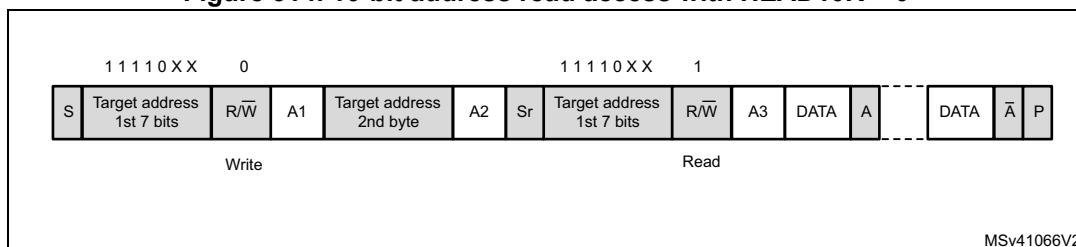
MSv19859V3

### Initialization of a controller receiver addressing a 10-bit address target

If the target address is in 10-bit format, the user can choose to send the complete read sequence, by clearing the HEAD10R bit of the I2C\_CR2 register. In this case, the controller automatically sends the following complete sequence after the START bit is set:

(RE)START + Target address 10-bit header Write + Target address second byte +  
(RE)START + Target address 10-bit header Read.

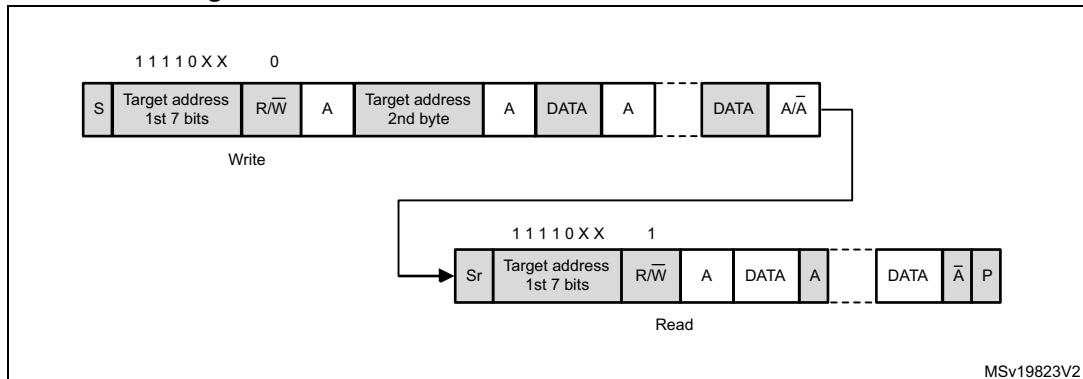
**Figure 314. 10-bit address read access with HEAD10R = 0**



If the controller addresses a 10-bit address target, transmits data to this target and then reads data from the same target, a controller transmission flow must be done first. Then a repeated START is set with the 10-bit target address configured with HEAD10R = 1. In this case, the controller sends this sequence:

RESTART + Target address 10-bit header Read.

**Figure 315. 10-bit address read access with HEAD10R = 1**



MSv19823V2

### Controller transmitter

In the case of a write transfer, the TXIS flag is set after each byte transmission, after the ninth SCL pulse when an ACK is received.

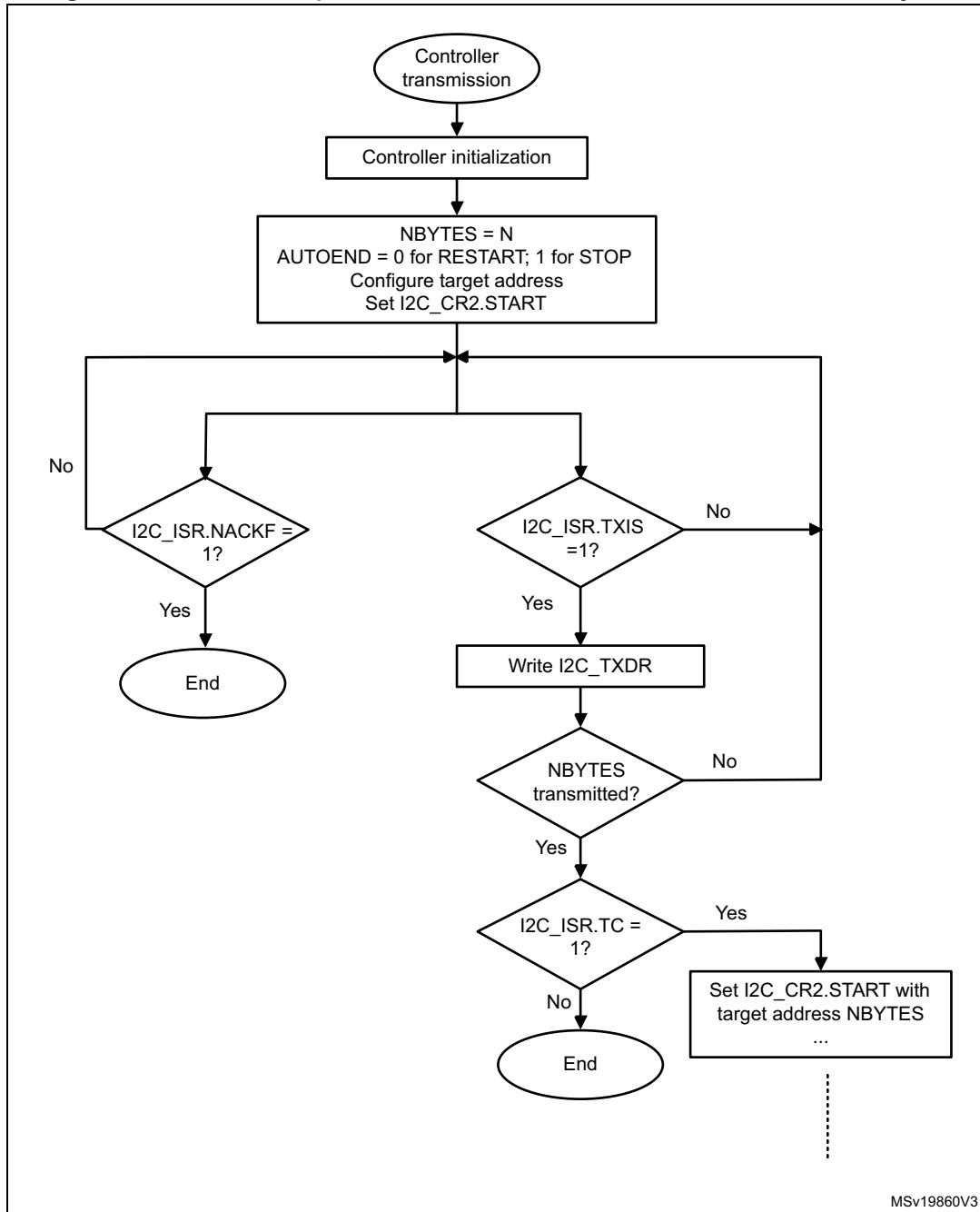
A TXIS event generates an interrupt if the TXIE bit of the I2C\_CR1 register is set. The flag is cleared when the I2C\_TXDR register is written with the next data byte to transmit.

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 transmit is greater than 255, the reload mode must be selected by setting the RELOAD bit in the I2C\_CR2 register. In this case, when the NBYTES[7:0] number of data bytes is transferred, the TCR flag is set and the SCL line is stretched low until NBYTES[7:0] is written with a non-zero value.

When RELOAD = 0 and the number of data bytes defined in NBYTES[7:0] is transferred:

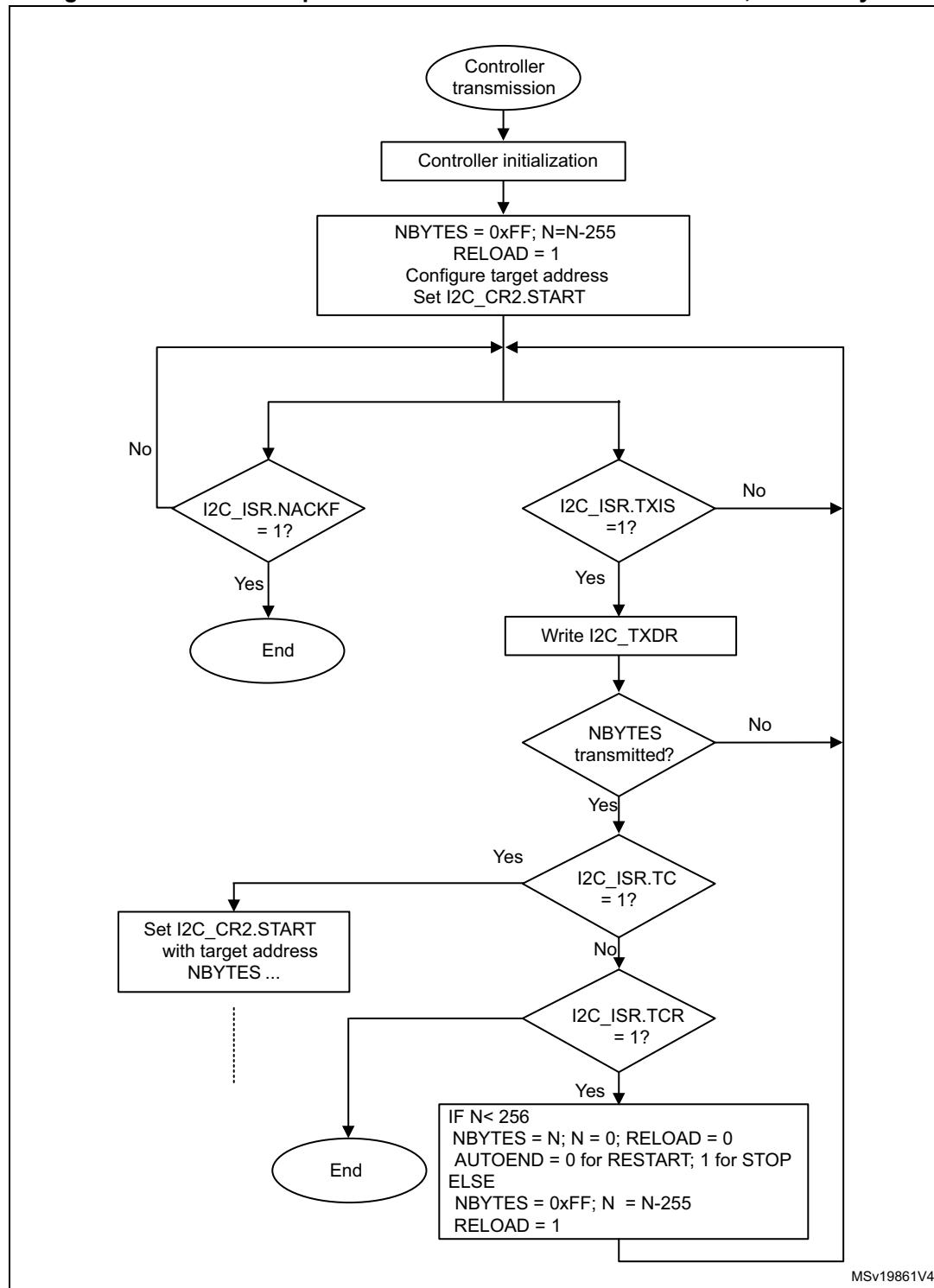
- In automatic end mode (AUTOEND = 1), a STOP condition is automatically sent.
- In software end mode (AUTOEND = 0), the TC flag is set and the SCL line is stretched low, to perform software actions:
  - A RESTART condition can be requested by setting the START bit of the I2C\_CR2 register with the proper target address configuration and the number of bytes to transfer. Setting the START bit clears the TC flag and sends the START condition on the bus.
  - A STOP condition can be requested by setting the STOP bit of the I2C\_CR2 register. This clears the TC flag and sends a STOP condition on the bus.

When a NACK is received, the TXIS flag is not set and a STOP condition is automatically sent. The NACKF flag of the I2C\_ISR register is set. An interrupt is generated if the NACKIE bit is set.

**Figure 316. Transfer sequence flow for I2C controller transmitter,  $N \leq 255$  bytes**

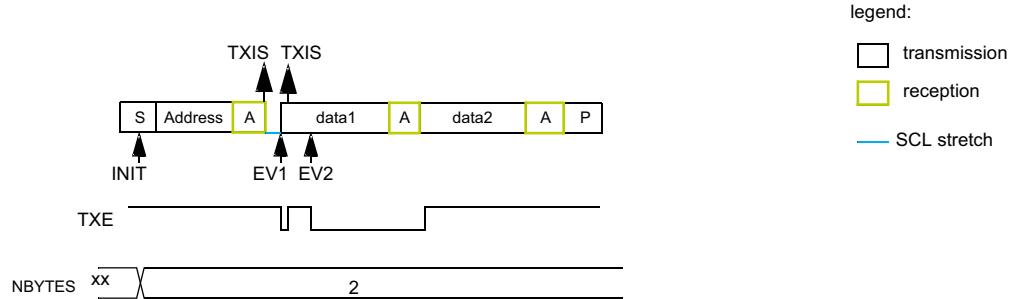
MSv19860V3

Figure 317. Transfer sequence flow for I2C controller transmitter, N &gt; 255 bytes



**Figure 318. Transfer bus diagrams for I2C controller transmitter  
(mandatory events only)**

Example I2C controller transmitter 2 bytes, automatic end mode (STOP)

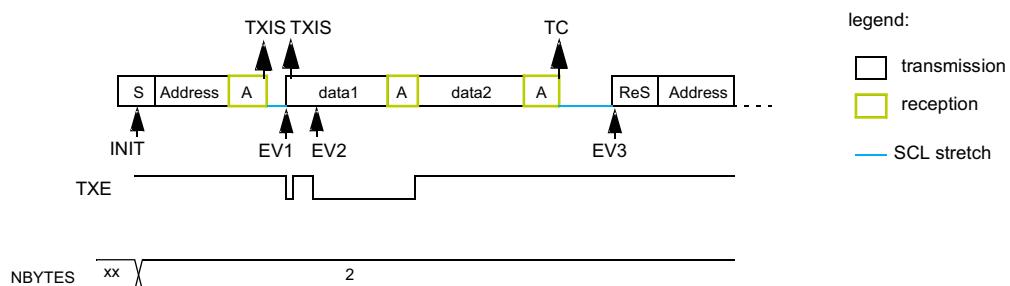


INIT: program target address, program NBYTES = 2, AUTOEND = 1, set START

EV1: TXIS ISR: wr data1

EV2: TXIS ISR: wr data2

Example I2C controller transmitter 2 bytes, software end mode (RESTART)



INIT: program target address, program NBYTES = 2, AUTOEND = 0, set START

EV1: TXIS ISR: wr data1

EV2: TXIS ISR: wr data2

EV3: TC ISR: program target address, program NBYTES = N, set START

MSv19862V3

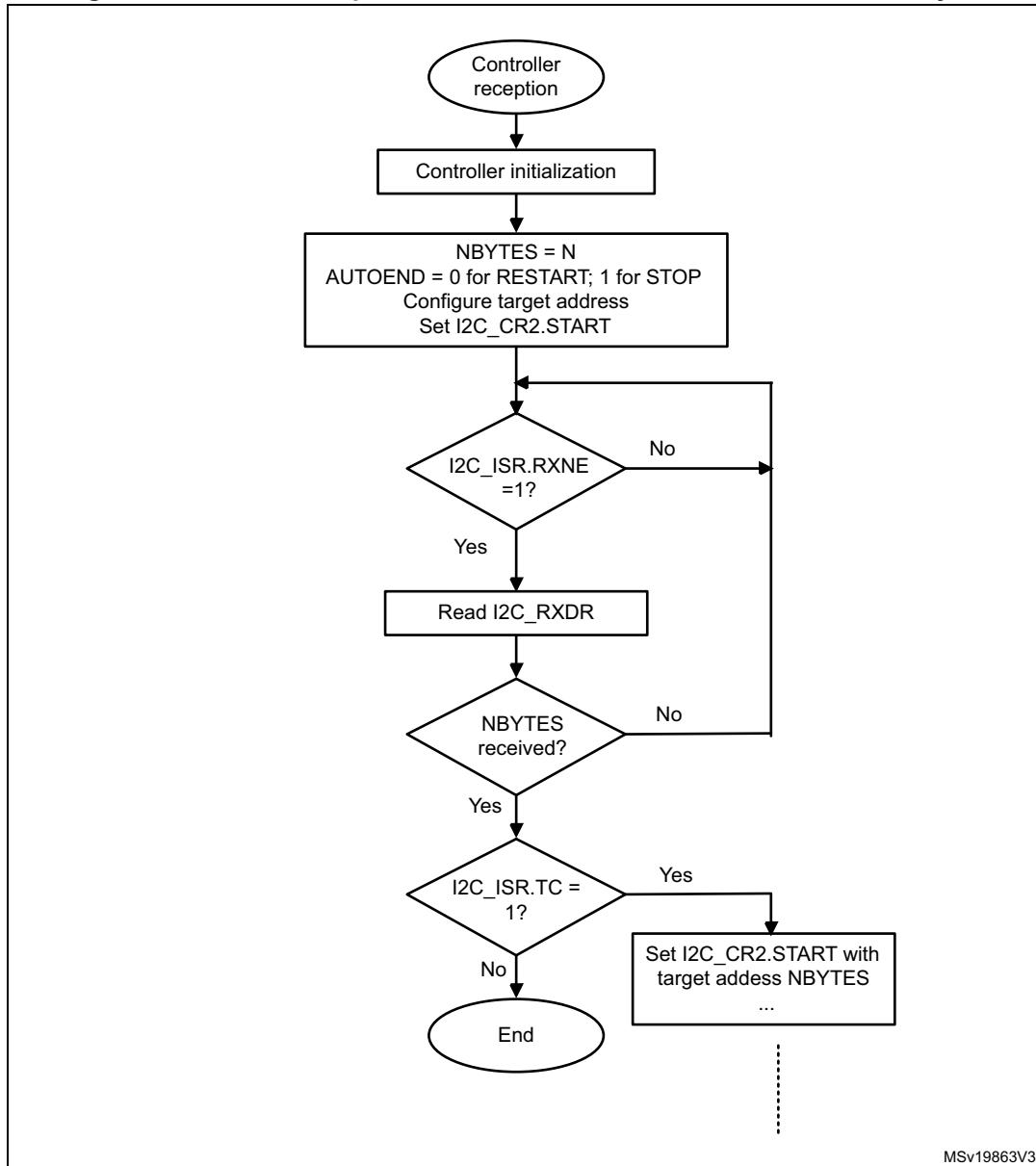
### Controller receiver

In the case of a read transfer, the RXNE flag is set after each byte reception, after the eighth SCL pulse. An RXNE event generates an interrupt if the RXIE bit of the I2C\_CR1 register is set. The flag is cleared when I2C\_RXDR is read.

If the total number of data bytes to receive is greater than 255, select the reload mode, by setting the RELOAD bit of the I2C\_CR2 register. In this case, when the NBYTES[7:0] number of data bytes is transferred, the TCR flag is set and the SCL line is stretched low until NBYTES[7:0] is written with a non-zero value.

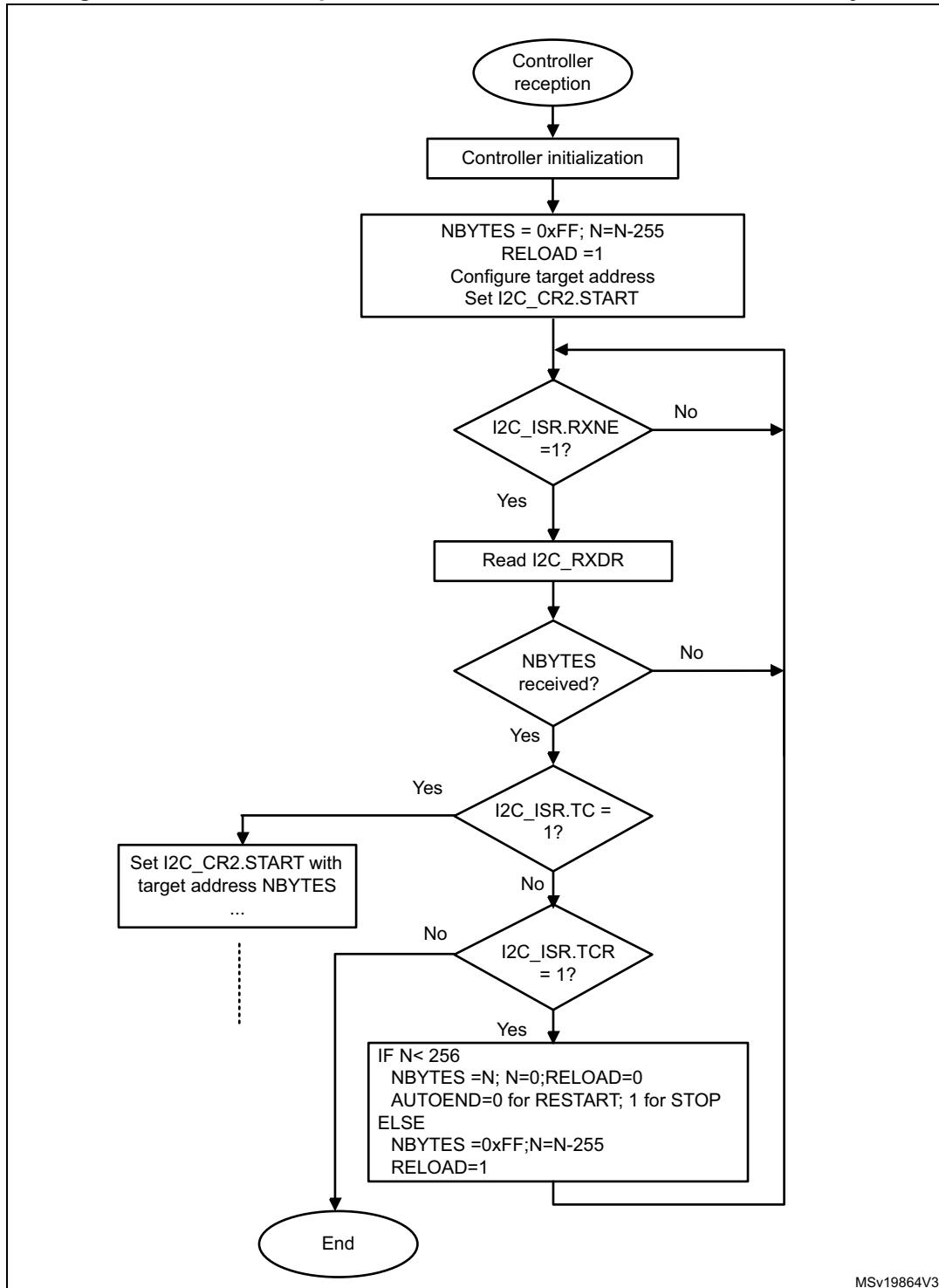
When RELOAD = 0 and the number of data bytes defined in NBYTES[7:0] is transferred:

- 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 of the I2C\_CR2 register, with the proper target address configuration and the number of bytes to transfer. Setting the START bit clears the TC flag and sends the START condition and the target address on the bus.
  - A STOP condition can be requested by setting the STOP bit of the I2C\_CR2 register. This clears the TC flag and sends a STOP condition on the bus.

**Figure 319. Transfer sequence flow for I2C controller receiver,  $N \leq 255$  bytes**

MSv19863V3

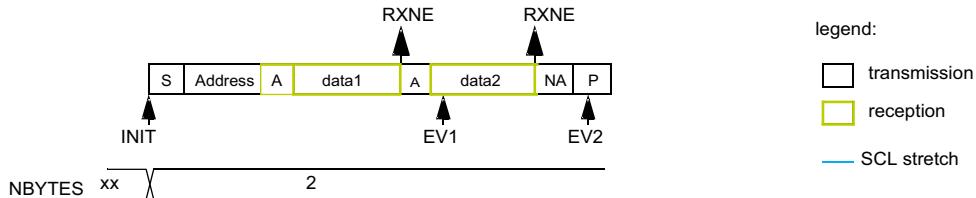
Figure 320. Transfer sequence flow for I2C controller receiver, N &gt; 255 bytes



MSv19864V3

**Figure 321. Transfer bus diagrams for I2C controller receiver  
(mandatory events only)**

Example I2C controller receiver 2 bytes, automatic end mode (STOP)

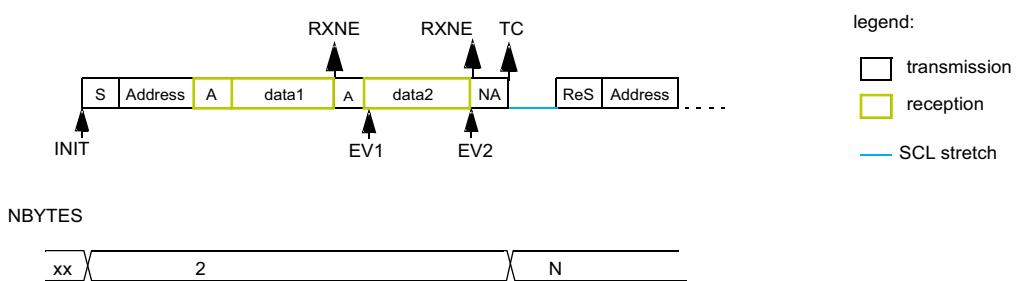


INIT: program target address, program NBYTES = 2, AUTOEND=1, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

Example I2C controller receiver 2 bytes, software end mode (RESTART)



INIT: program target address, program NBYTES = 2, AUTOEND=0, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: read data2

EV3: TC ISR: program target address, program NBYTES = N, set START

MSv19865V2

### 30.4.10 I2C\_TIMINGR register configuration examples

The following tables provide examples of how to program the I2C\_TIMINGR register to obtain timings compliant with the I<sup>2</sup>C-bus specification. To get more accurate configuration values, use the STM32CubeMX tool (*I2C Configuration* window).

**Table 186. Timing settings for  $f_{I2CCLK}$  of 8 MHz**

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	500 kHz
PRESC[3:0]	0x1	0x1	0x0	0x0
SCLL[7:0]	0xC7	0x13	0x9	0x6
$t_{SCLL}$	$200 \times 250 \text{ ns} = 50 \mu\text{s}$	$20 \times 250 \text{ ns} = 5.0 \mu\text{s}$	$10 \times 125 \text{ ns} = 1250 \text{ ns}$	$7 \times 125 \text{ ns} = 875 \text{ ns}$
SCLH[7:0]	0xC3	0xF	0x3	0x3
$t_{SCLH}$	$196 \times 250 \text{ ns} = 49 \mu\text{s}$	$16 \times 250 \text{ ns} = 4.0 \mu\text{s}$	$4 \times 125 \text{ ns} = 500 \text{ ns}$	$4 \times 125 \text{ ns} = 500 \text{ ns}$
$t_{SCL}^{(1)}$	$\sim 100 \mu\text{s}^{(2)}$	$\sim 10 \mu\text{s}^{(2)}$	$\sim 2.5 \mu\text{s}^{(3)}$	$\sim 2.0 \mu\text{s}^{(4)}$
SDADEL[3:0]	0x2	0x2	0x1	0x0
$t_{SDADEL}$	$2 \times 250 \text{ ns} = 500 \text{ ns}$	$2 \times 250 \text{ ns} = 500 \text{ ns}$	$1 \times 125 \text{ ns} = 125 \text{ ns}$	0 ns
SCLDEL[3:0]	0x4	0x4	0x3	0x1
$t_{SCLDEL}$	$5 \times 250 \text{ ns} = 1250 \text{ ns}$	$5 \times 250 \text{ ns} = 1250 \text{ ns}$	$4 \times 125 \text{ ns} = 500 \text{ ns}$	$2 \times 125 \text{ ns} = 250 \text{ ns}$

1.  $t_{SCL}$  is greater than  $t_{SCLL} + t_{SCLH}$  due to SCL internal detection delay. Values provided for  $t_{SCL}$  are examples only.
2.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 500 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 1000 \text{ ns}$ .
3.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 500 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 750 \text{ ns}$ .
4.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 500 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 655 \text{ ns}$ .

**Table 187. Timing settings for  $f_{I2CCLK}$  of 16 MHz**

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	1000 kHz
PRESC[3:0]	0x3	0x3	0x1	0x0
SCLL[7:0]	0xC7	0x13	0x9	0x4
$t_{SCLL}$	$200 \times 250 \text{ ns} = 50 \mu\text{s}$	$20 \times 250 \text{ ns} = 5.0 \mu\text{s}$	$10 \times 125 \text{ ns} = 1250 \text{ ns}$	$5 \times 62.5 \text{ ns} = 312.5 \text{ ns}$
SCLH[7:0]	0xC3	0xF	0x3	0x2
$t_{SCLH}$	$196 \times 250 \text{ ns} = 49 \mu\text{s}$	$16 \times 250 \text{ ns} = 4.0 \mu\text{s}$	$4 \times 125 \text{ ns} = 500 \text{ ns}$	$3 \times 62.5 \text{ ns} = 187.5 \text{ ns}$
$t_{SCL}^{(1)}$	$\sim 100 \mu\text{s}^{(2)}$	$\sim 10 \mu\text{s}^{(2)}$	$\sim 2.5 \mu\text{s}^{(3)}$	$\sim 1.0 \mu\text{s}^{(4)}$
SDADEL[3:0]	0x2	0x2	0x2	0x0
$t_{SDADEL}$	$2 \times 250 \text{ ns} = 500 \text{ ns}$	$2 \times 250 \text{ ns} = 500 \text{ ns}$	$2 \times 125 \text{ ns} = 250 \text{ ns}$	0 ns
SCLDEL[3:0]	0x4	0x4	0x3	0x2
$t_{SCLDEL}$	$5 \times 250 \text{ ns} = 1250 \text{ ns}$	$5 \times 250 \text{ ns} = 1250 \text{ ns}$	$4 \times 125 \text{ ns} = 500 \text{ ns}$	$3 \times 62.5 \text{ ns} = 187.5 \text{ ns}$

1.  $t_{SCL}$  is greater than  $t_{SCLL} + t_{SCLH}$  due to SCL internal detection delay. Values provided for  $t_{SCL}$  are examples only.
2.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 250 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 1000 \text{ ns}$ .
3.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 250 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 750 \text{ ns}$ .
4.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 250 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 500 \text{ ns}$ .

**Table 188. Timing settings for  $f_{I2CCLK}$  of 48 MHz**

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	1000 kHz
PRESC[3:0]	0xB	0xB	0x5	0x5
SCLL[7:0]	0xC7	0x13	0x9	0x3
$t_{SCLL}$	$200 \times 250 \text{ ns} = 50 \mu\text{s}$	$20 \times 250 \text{ ns} = 5.0 \mu\text{s}$	$10 \times 125 \text{ ns} = 1250 \text{ ns}$	$4 \times 125 \text{ ns} = 500 \text{ ns}$
SCLH[7:0]	0xC3	0xF	0x3	0x1
$t_{SCLH}$	$196 \times 250 \text{ ns} = 49 \mu\text{s}$	$16 \times 250 \text{ ns} = 4.0 \mu\text{s}$	$4 \times 125 \text{ ns} = 500 \text{ ns}$	$2 \times 125 \text{ ns} = 250 \text{ ns}$
$t_{SCL}^{(1)}$	$\sim 100 \mu\text{s}^{(2)}$	$\sim 10 \mu\text{s}^{(2)}$	$\sim 2.5 \mu\text{s}^{(3)}$	$\sim 875 \text{ ns}^{(4)}$
SDADEL[3:0]	0x2	0x2	0x3	0x0
$t_{SDADEL}$	$2 \times 250 \text{ ns} = 500 \text{ ns}$	$2 \times 250 \text{ ns} = 500 \text{ ns}$	$3 \times 125 \text{ ns} = 375 \text{ ns}$	0 ns
SCLDEL[3:0]	0x4	0x4	0x3	0x1
$t_{SCLDEL}$	$5 \times 250 \text{ ns} = 1250 \text{ ns}$	$5 \times 250 \text{ ns} = 1250 \text{ ns}$	$4 \times 125 \text{ ns} = 500 \text{ ns}$	$2 \times 125 \text{ ns} = 250 \text{ ns}$

1.  $t_{SCL}$  is greater than  $t_{SCLL} + t_{SCLH}$  due to the SCL internal detection delay. Values provided for  $t_{SCL}$  are only examples.

2.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 83.3 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 1000 \text{ ns}$

3.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 83.3 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 750 \text{ ns}$

4.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 83.3 \text{ ns}$ . Example with  $t_{SYNC1} + t_{SYNC2} = 250 \text{ ns}$

### 30.4.11 SMBus specific features

#### 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 operation principles of the I<sup>2</sup>C-bus. The SMBus provides a control bus for system and power management related tasks.

The I<sup>2</sup>C peripheral is compatible with the SMBus specification (<http://smbus.org>).

The system management bus specification refers to three types of devices:

- **Target** is a device that receives or responds to a command.
- **Controller** is a device that issues commands, generates clocks, and terminates the transfer.
- **Host** is a specialized controller that provides the main interface to the system CPU. A host must be a controller-target and must support the SMBus *host notify* protocol. Only one host is allowed in a system.

The I<sup>2</sup>C peripheral can be configured as a controller or a target device, and also as a host.

#### Bus protocols

There are eleven possible command protocols for any given device. The device can use any or all of them to communicate. These 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*. The protocols must be implemented by the user software.

For more details on these protocols, refer to the SMBus specification (<http://smbus.org>). STM32CubeMX implements an SMBus stack thanks to X-CUBE-SMBUS, a downloadable software pack that allows basic SMBus configuration per I2C instance.

### Address resolution protocol (ARP)

SMBus target address conflicts can be resolved by dynamically assigning a new unique address to each target device. To provide a mechanism to isolate each device for the purpose of address assignment, each device must implement a unique 128-bit device identifier (UDID). In the I2C peripheral, it is implemented by software.

The I2C peripheral supports the Address resolution protocol (ARP). The SMBus device default address (0b1100 001) is enabled by setting the SMBDEN bit of the I2C\_CR1 register. The ARP commands must be implemented by the user software.

Arbitration is also performed in target mode for ARP support.

For more details on the SMBus address resolution protocol, refer to the SMBus specification (<http://smbus.org>).

### Received command and data acknowledge control

An SMBus receiver must be able to NACK each received command or data. In order to allow the ACK control in target mode, the target byte control mode must be enabled, by setting the SBC bit of the I2C\_CR1 register. Refer to [Target byte control mode](#) for more details.

### Host notify protocol

To enable the host notify protocol, set the SMBHEN bit of the I2C\_CR1 register. The I2C peripheral then acknowledges the SMBus host address (0b0001 000).

When this protocol is used, the device acts as a controller and the host as a target.

### SMBus alert

The I2C peripheral supports the SMBALERT# optional signal through the SMBA pin. With the SMBALERT# signal, an SMBus target device can signal to the SMBus host 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/devices which pulled SMBALERT# low acknowledges/acknowledge the alert response address.

When the I2C peripheral is configured as an SMBus target device (SMBHEN = 0), the SMBA pin is pulled low by setting the ALERTEN bit of the I2C\_CR1 register. The alert response address is enabled at the same time.

When the I2C peripheral is configured as an SMBus host (SMBHEN = 1), the ALERT flag of the I2C\_ISR register is set when a falling edge is detected on the SMBA pin and ALERTEN = 1. An interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set. When ALERTEN = 0, the alert line is considered high even if the external SMBA pin is low.

**Note:** If the SMBus alert pin is not required, keep the ALERTEN bit cleared. The SMBA pin can then be used as a standard GPIO.

### Packet error checking

A packet error checking mechanism introduced in the SMBus specification improves reliability and communication robustness. The packet error checking is implemented by

appending a packet error code (PEC) at the end of each message transfer. The PEC is calculated by using the  $C(x) = x^8 + x^2 + x + 1$  CRC-8 polynomial on all the message bytes (including addresses and read/write bits).

The I2C peripheral embeds a hardware PEC calculator and allows a not acknowledge to be sent automatically when the received byte does not match the hardware calculated PEC.

### Timeouts

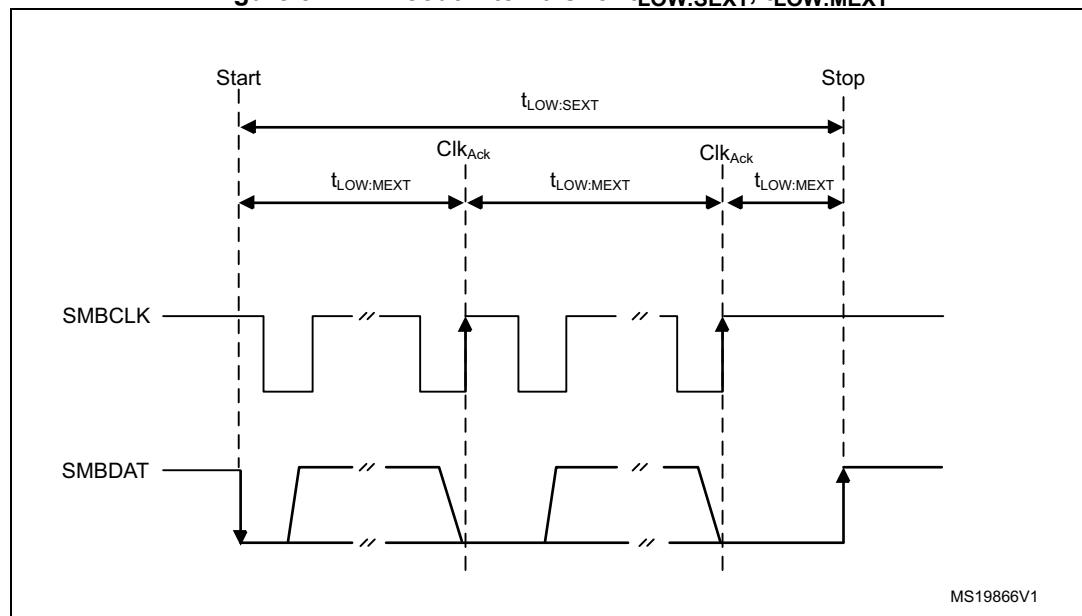
To comply with the SMBus timeout specifications, the I2C peripheral embeds hardware timers.

**Table 189. 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 (target device)	-	25	
$t_{LOW:MEXT}^{(2)}$	Cumulative clock low extend time (controller device)	-	10	

1.  $t_{LOW:SEXT}$  is the cumulative time a given target device is allowed to extend the clock cycles in one message from the initial START to the STOP. It is possible that another target device or the controller also extends the clock causing the combined clock low extend time to be greater than  $t_{LOW:SEXT}$ . The value provided applies to a single target device connected to a full-target controller.
2.  $t_{LOW:MEXT}$  is the cumulative time a controller 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 target device or another controller also extends the clock, causing the combined clock low time to be greater than  $t_{LOW:MEXT}$  on a given byte. The value provided applies to a single target device connected to a full-target controller.

**Figure 322. Timeout intervals for  $t_{LOW:SEXT}$ ,  $t_{LOW:MEXT}$**



### Bus idle detection

A controller can assume that the bus is free if it detects that the clock and data signals have been high for  $t_{IDLE} > t_{HIGH(max)}$  (refer to the table in [Section 30.4.9](#)).

This timing parameter covers the condition where a controller is dynamically added to the bus, and may not have detected a state transition on the SMBCLK or SMBDAT lines. In this case, the controller must wait long enough to ensure that a transfer is not currently in progress. The I2C peripheral supports a hardware bus idle detection.

## 30.4.12 SMBus initialization

In addition to the I2C initialization for the I<sup>2</sup>C-bus, the use of the peripheral for the SMBus communication requires some extra initialization steps.

### Received command and data acknowledge control (target mode)

An SMBus receiver must be able to NACK each received command or data. To allow ACK control in target mode, the target byte control mode must be enabled, by setting the SBC bit of the I2C\_CR1 register. Refer to [Target byte control mode](#) for more details.

### Specific addresses (target mode)

The specific SMBus addresses must be enabled if required. Refer to [Bus idle detection](#) for more details.

The SMBus device default address (0b1100 001) is enabled by setting the SMBDEN bit of the I2C\_CR1 register.

The SMBus host address (0b0001 000) is enabled by setting the SMBHEN bit of the I2C\_CR1 register.

The alert response address (0b0001100) is enabled by setting the ALERTEN bit of the I2C\_CR1 register.

### Packet error checking

PEC calculation is enabled by setting the PECEN bit of the I2C\_CR1 register. Then the PEC transfer is managed with the help of the hardware byte counter associated with the NBYTES[7:0] bitfield of the I2C\_CR2 register. The PECEN bit must be configured before enabling the I2C.

The PEC transfer is managed with the hardware byte counter, so the SBC bit must be set when interfacing the SMBus in target mode. The PEC is transferred after transferring NBYTES[7:0] - 1 data bytes, if 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 I2C peripheral is enabled.

**Table 190. SMBus with PEC configuration**

Mode	SBC bit	RELOAD bit	AUTOEND bit	PECBYTE bit
Controller Tx/Rx NBYTES + PEC+ STOP	X	0	1	1
Controller Tx/Rx NBYTES + PEC + ReSTART	X	0	0	1
Target Tx/Rx with PEC	1	0	X	1

### Timeout detection

The timeout detection is enabled by setting the TIMOUTEN and TEXTEN bits of the I2C\_TIMEOUTTR register. The timers must be programmed in such a way that they detect a timeout before the maximum time given in the SMBus specification.

#### *t<sub>TIMOUT</sub> check*

To check the  $t_{\text{TIMEOUT}}$  parameter, load the 12-bit TIMEOUTA[11:0] bitfield with the timer reload value. Keep the TIDLE bit at 0 to detect the SCL low level timeout.

Then set the TIMOUTEN bit of the I2C\_TIMEOUTTR register, to enable the timer.

If SCL is tied low for longer than the  $(\text{TIMEOUTA} + 1) \times 2048 \times t_{\text{I2CCLK}}$  period, the TIMEOUT flag of the I2C\_ISR register is set.

Refer to [Table 191](#).

**Caution:** Changing the TIMEOUTA[11:0] bitfield and the TIDLE bit values is not allowed when the TIMOUTEN bit is set.

#### *t<sub>LOW:SEXT</sub> and t<sub>LOW:MEXT</sub> check*

A 12-bit timer associated with the TIMEOUTB[11:0] bitfield allows checking  $t_{\text{LOW:SEXT}}$  for the I2C peripheral operating as a target, or  $t_{\text{LOW:MEXT}}$  when it operates as a controller. As the standard only specifies a maximum, the user can choose the same value for both. The timer is then enabled by setting the TEXTEN bit in the I2C\_TIMEOUTTR register.

If the SMBus peripheral performs a cumulative SCL stretch for longer than the  $(\text{TIMEOUTB} + 1) \times 2048 \times t_{\text{I2CCLK}}$  period, and within the timeout interval described in [Bus idle detection](#) section, the TIMEOUT flag of the I2C\_ISR register is set.

Refer to [Table 192](#).

**Caution:** Changing the TIMEOUTB[11:0] bitfield value is not allowed when the TEXTEN bit is set.

### Bus idle detection

To check the  $t_{\text{IDLE}}$  period, the TIMEOUTA[11:0] bitfield associated with 12-bit timer must be loaded with the timer reload value. Keep the TIDLE bit at 1 to detect both SCL and SDA high level timeout. Then set the TIMOUTEN bit of the I2C\_TIMEOUTTR register to enable the timer.

If both the SCL and SDA lines remain high for longer than the  $(\text{TIMEOUTA} + 1) \times 4 \times t_{\text{I2CCLK}}$  period, the TIMEOUT flag of the I2C\_ISR register is set.

Refer to [Table 193](#).

**Caution:** Changing the TIMEOUTA[11:0] bitfield and the TIDLE bit values is not allowed when the TIMOUTEN bit is set.

### 30.4.13 SMBus I2C\_TIMEOUTR register configuration examples

The following tables provide examples of settings to reach desired  $t_{TIMEOUT}$ ,  $t_{LOW:SEXT}$ ,  $t_{LOW:MEXT}$ , and  $t_{IDLE}$  timings at different  $f_{I2CCLK}$  frequencies.

**Table 191. TIMEOUTA[11:0] for maximum  $t_{TIMEOUT}$  of 25 ms**

$f_{I2CCLK}$	TIMEOUTA[11:0]	TIDLE	TIMEOUTEN	$t_{TIMEOUT}$
8 MHz	0x61	0	1	$98 \times 2048 \times 125 \text{ ns} = 25 \text{ ms}$
16 MHz	0xC3	0	1	$196 \times 2048 \times 62.5 \text{ ns} = 25 \text{ ms}$
32 MHz	0x186	0	1	$391 \times 2048 \times 31.25 \text{ ns} = 25 \text{ ms}$
48 MHz	0x249	0	1	$586 \times 2048 \times 20.08 \text{ ns} = 25 \text{ ms}$

**Table 192. TIMEOUTB[11:0] for maximum  $t_{LOW:SEXT}$  and  $t_{LOW:MEXT}$  of 8 ms**

$f_{I2CCLK}$	TIMEOUTB[11:0]	TEXTEN	$t_{LOW:SEXT}$ $t_{LOW:MEXT}$
8 MHz	0x1F	1	$32 \times 2048 \times 125 \text{ ns} = 8 \text{ ms}$
16 MHz	0x3F	1	$64 \times 2048 \times 62.5 \text{ ns} = 8 \text{ ms}$
32 MHz	0x7C	1	$125 \times 2048 \times 31.25 \text{ ns} = 8 \text{ ms}$
48 MHz	0xBB	1	$188 \times 2048 \times 20.08 \text{ ns} = 8 \text{ ms}$

**Table 193. TIMEOUTA[11:0] for maximum  $t_{IDLE}$  of 50  $\mu$ s**

$f_{I2CCLK}$	TIMEOUTA[11:0]	TIDLE	TIMEOUTEN	$t_{IDLE}$
8 MHz	0x63	1	1	$100 \times 4 \times 125 \text{ ns} = 50 \mu\text{s}$
16 MHz	0xC7	1	1	$200 \times 4 \times 62.5 \text{ ns} = 50 \mu\text{s}$
32 MHz	0x18F	1	1	$400 \times 4 \times 31.25 \text{ ns} = 50 \mu\text{s}$
48 MHz	0x257	1	1	$600 \times 4 \times 20.08 \text{ ns} = 50 \mu\text{s}$

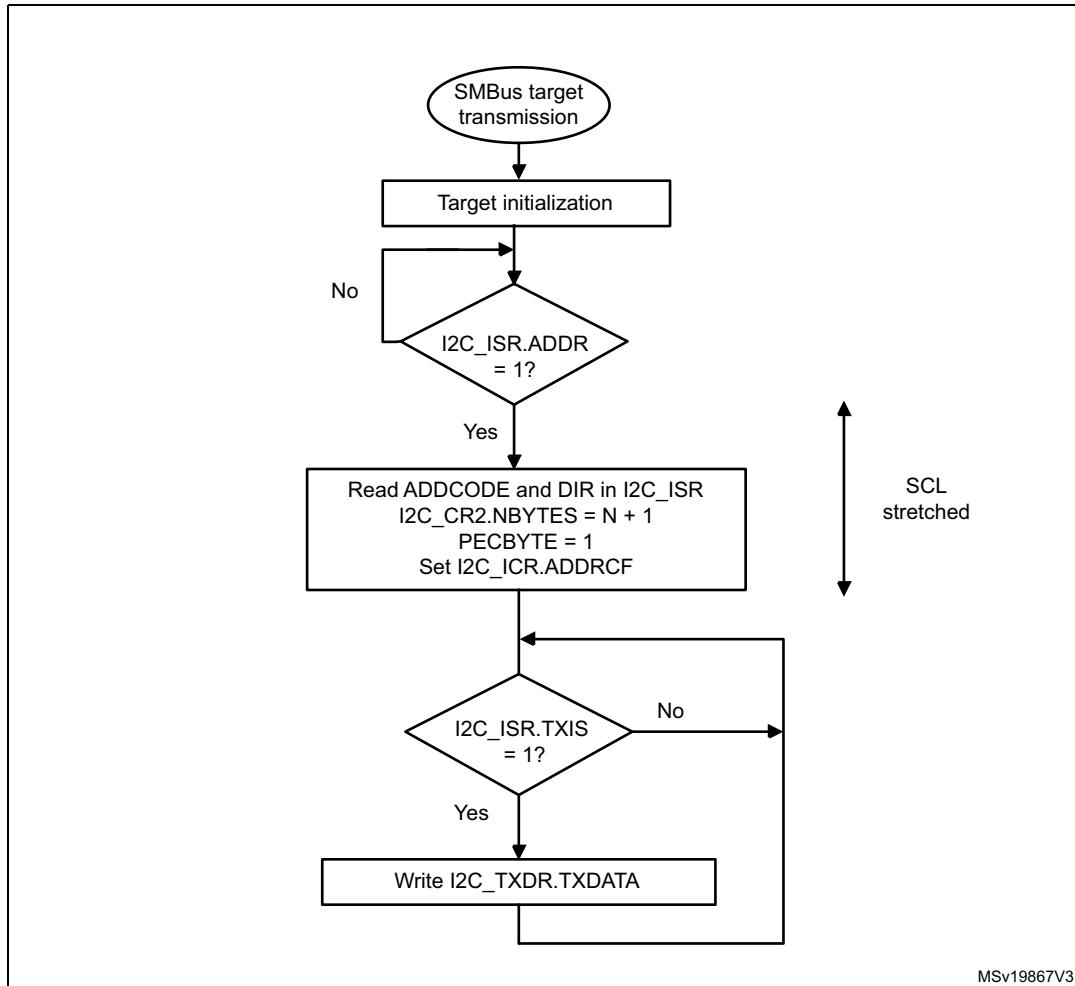
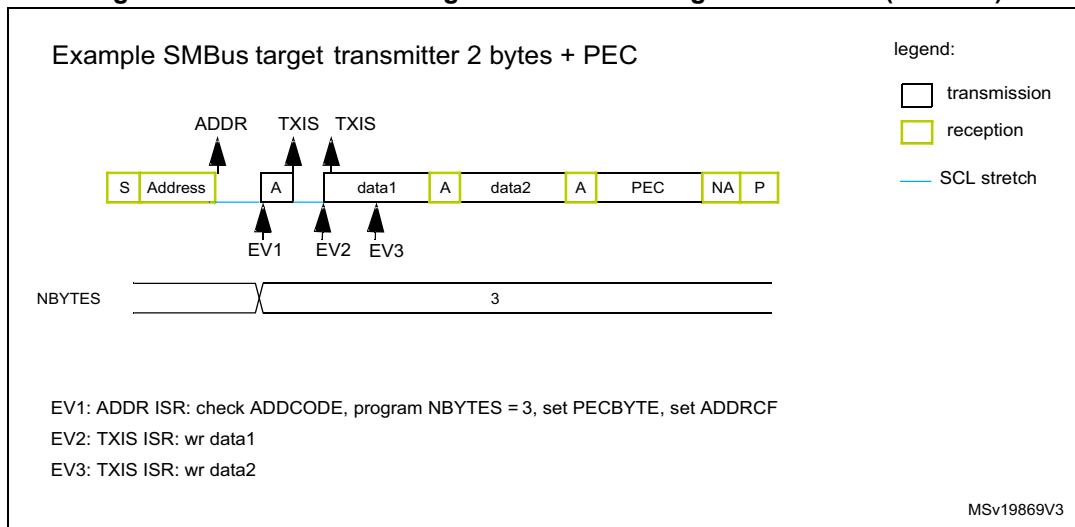
### 30.4.14 SMBus target mode

In addition to I2C target transfer management (refer to [Section 30.4.8: I2C target mode](#)), this section provides extra software flowcharts to support SMBus.

#### SMBus target transmitter

When using the I2C peripheral in SMBus mode, set the SBC bit to enable the PEC transmission at the end of the programmed number of data bytes. When the PECPBYTE bit is set, the number of bytes programmed in NBYTES[7:0] includes the PEC transmission. In that case, the total number of TXIS interrupts is NBYTES[7:0] - 1, and the content of the I2C\_PECR register is automatically transmitted if the controller requests an extra byte after the transfer of the NBYTES[7:0] - 1 data bytes.

**Caution:** The PECPBYTE bit has no effect when the RELOAD bit is set.

**Figure 323. Transfer sequence flow for SMBus target transmitter N bytes + PEC****Figure 324. Transfer bus diagram for SMBus target transmitter (SBC = 1)**

### SMBus target receiver

When using the I2C peripheral in SMBus mode, set the SBC bit to enable the PEC checking at the end of the programmed number of data bytes. To allow the ACK control of each byte, the reload mode must be selected (RELOAD = 1). Refer to [Target byte control mode](#) for more details.

To check the PEC byte, the RELOAD bit must be cleared and the PECBYTE bit must be set. In this case, after the receipt of NBYTES[7:0] - 1 data bytes, the next received byte is compared with the internal I2C\_PECR register content. A NACK is automatically generated if the comparison does not match, and an ACK is automatically generated if the comparison matches, whatever the ACK bit value. Once the PEC byte is received, it is copied into the I2C\_RXDR register like any other data, and the RXNE flag is set.

Upon a PEC mismatch, the PECERR flag is set and an interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set.

If no ACK software control is required, the user can set the PECBYTE bit and, in the same write operation, load NBYTES[7:0] with the number of bytes to receive in a continuous flow. After the receipt of NBYTES[7:0] - 1 bytes, the next received byte is checked as being the PEC.

**Caution:** The PECBYTE bit has no effect when the RELOAD bit is set.

Figure 325. Transfer sequence flow for SMBus target receiver N bytes + PEC

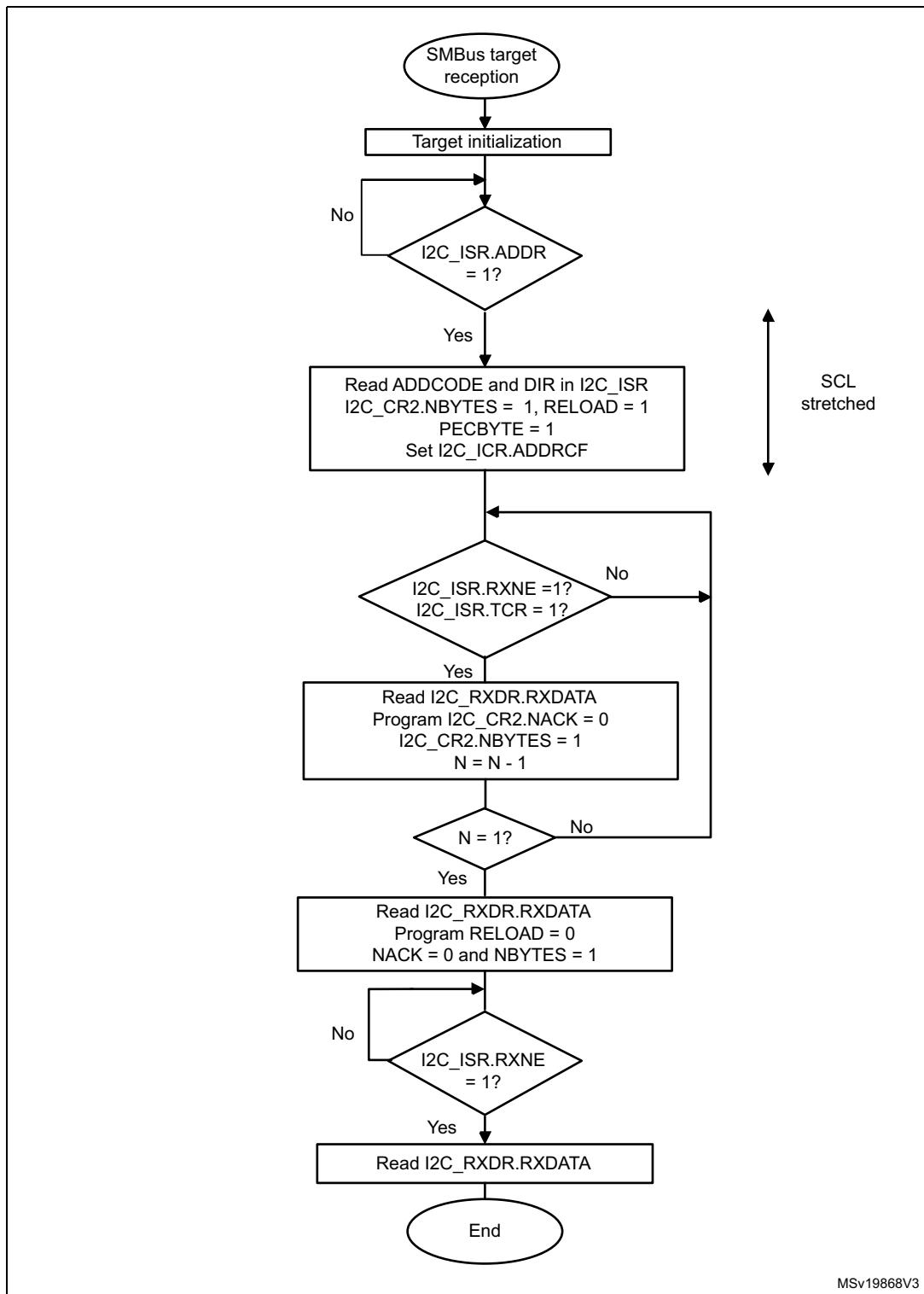
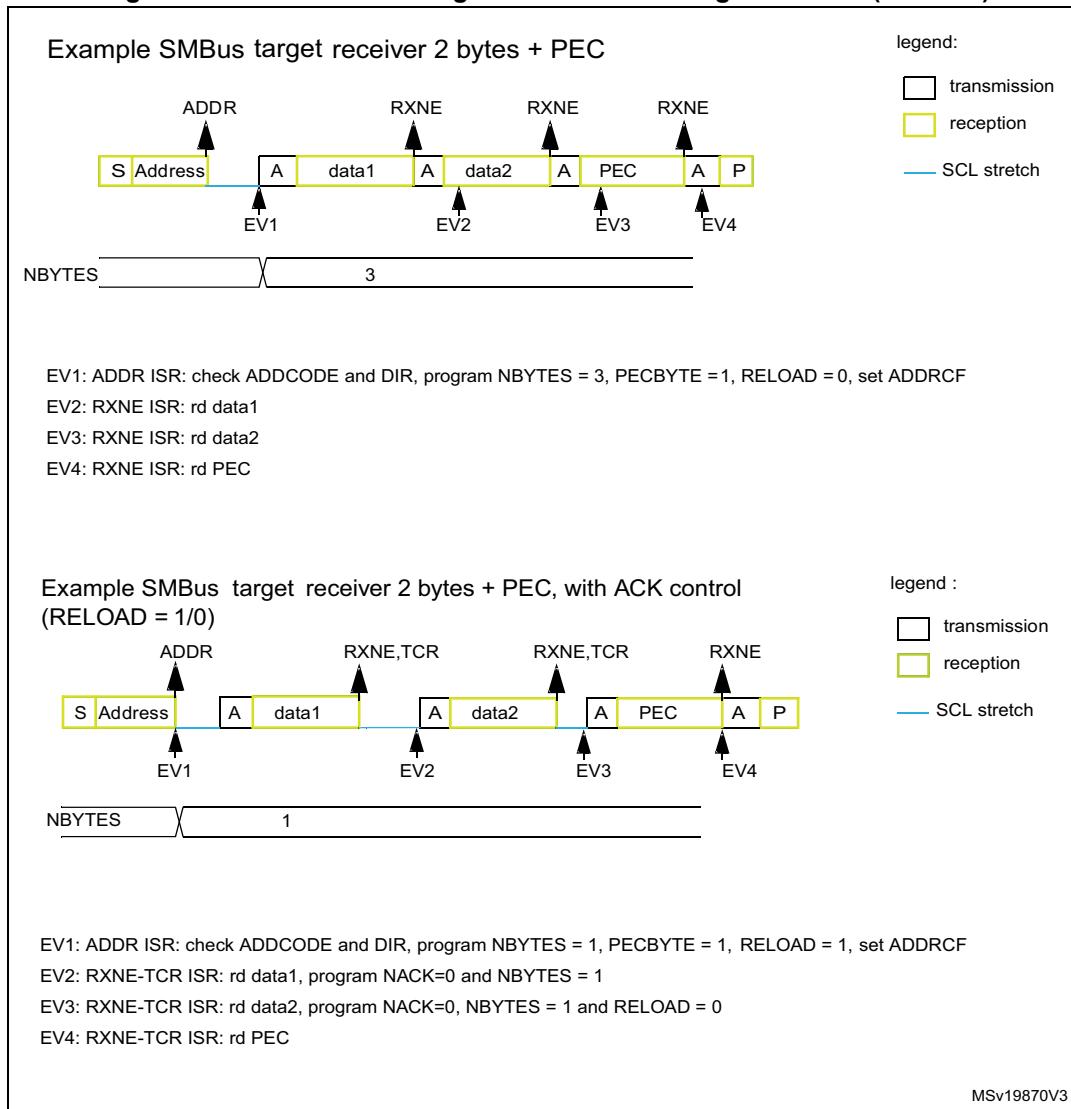


Figure 326. Bus transfer diagrams for SMBus target receiver (SBC = 1)



### 30.4.15 SMBus controller mode

In addition to I2C controller transfer management (refer to [Section 30.4.9: I2C controller mode](#)), this section provides extra software flowcharts to support SMBus.

#### SMBus controller transmitter

When the SMBus controller wants to transmit the PEC, the PECBYTE bit must be set and the number of bytes must be loaded in the NBYTES[7:0] bitfield, before setting the START bit. In this case, the total number of TXIS interrupts is NBYTES[7:0] - 1. So if the PECBYTE bit is set when NBYTES[7:0] = 0x1, the content of the I2C\_PECR register is automatically transmitted.

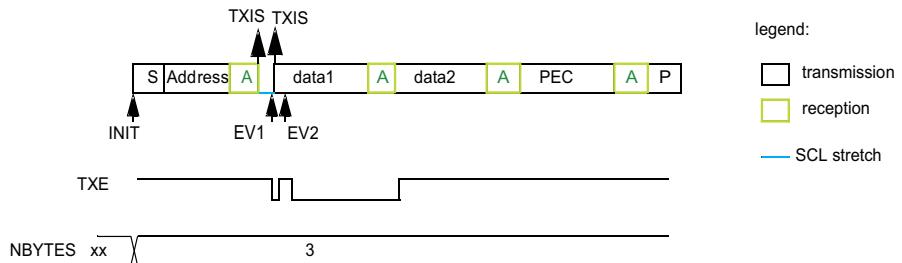
If the SMBus controller wants to send a STOP condition after the PEC, the automatic end mode must be selected (AUTOEND = 1). In this case, the STOP condition automatically follows the PEC transmission.

When the SMBus controller wants to send a RESTART condition after the PEC, the software mode must be selected (AUTOEND = 0). In this case, once NBYTES[7:0] - 1 are transmitted, the I2C\_PECR register content is transmitted. 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.

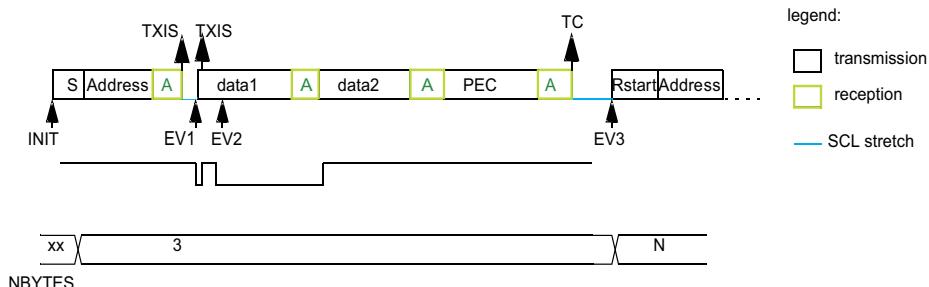
**Figure 327. Bus transfer diagrams for SMBus controller transmitter**

Example SMBus controller transmitter 2 bytes + PEC, automatic end mode (STOP)



INIT: program target address, program NBYTES = 3, AUTOEND=1, set PECBYTE, set START  
EV1: TXIS ISR: wr data1  
EV2: TXIS ISR: wr data2

Example SMBus controller transmitter 2 bytes + PEC, software end mode (RESTART)



INIT: program target address, program NBYTES = 3, AUTOEND=0, set PECBYTE, set START  
EV1: TXIS ISR: wr data1  
EV2: TXIS ISR: wr data2  
EV3: TC ISR: program target address, program NBYTES = N, set START

MSv19871V3

### SMBus controller receiver

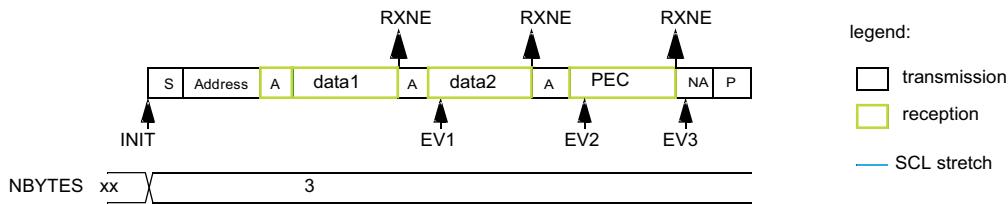
When the SMBus controller wants to receive, at the end of the transfer, the PEC followed by a STOP condition, the automatic end mode can be selected (AUTOEND = 1). The PECBYTE bit must be set and the target address programmed before setting the START bit. In this case, after the receipt of NBYTES[7:0] - 1 data bytes, the next received byte is automatically checked versus the I2C\_PECR register content. A NACK response is given to the PEC byte, followed by a STOP condition.

When the SMBus controller receiver wants to receive, at the end of the transfer, the PEC byte followed by a RESTART condition, the software mode must be selected (AUTOEND = 0). The PECBYTE bit must be set and the target address programmed before setting the START bit. In this case, after the receipt of NBYTES[7:0] - 1 data bytes, the next received byte is automatically checked versus the I2C\_PECR register content. The TC flag is set after the PEC byte reception, stretching the SCL line low. The RESTART condition can be programmed in the TC interrupt subroutine.

**Caution:** The PECBYTE bit has no effect when the RELOAD bit is set.

**Figure 328. Bus transfer diagrams for SMBus controller receiver**

Example SMBus controller receiver 2 bytes + PEC, automatic end mode (STOP)



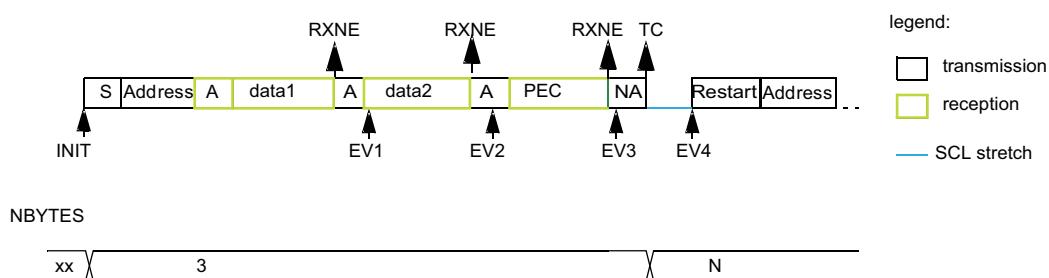
INIT: program target address, program NBYTES = 3, AUTOEND=1, set PECPBYTE, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

EV3: RXNE ISR: rd PEC

Example SMBus controller receiver 2 bytes + PEC, software end mode (RESTART)



INIT: program target 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 target address, program NBYTES = N, set START

MSv19872V3

### 30.4.16 Error conditions

The following errors are the conditions that can cause the communication to fail.

#### Bus error (BERR)

A bus error is detected when a START or a STOP condition is detected and is not located after a multiple of nine SCL clock pulses. START or STOP condition is detected when an SDA edge occurs while SCL is high.

The bus error flag is set only if the I2C peripheral is involved in the transfer as controller or addressed target (that is, not during the address phase in target mode).

In case of a misplaced START or RESTART detection in target mode, the I2C peripheral enters address recognition state like for a correct START condition.

When a bus error is detected, the BERR flag of the I2C\_ISR register is set, and an interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set.

### Arbitration loss (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 controller 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 controller switches automatically to target mode.

In target 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 of the I2C\_ISR register is set and an interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set.

### Overrun/underrun error (OVR)

An overrun or underrun error is detected in target mode when NOSTRETCH = 1 and:

- In reception when a new byte is received and the RXDR register has not been read yet. The new received byte is lost, and a NACK is automatically sent as a response to the new byte.
- In transmission:
  - When STOPF = 1 and the first data byte must be sent. The content of the I2C\_TXDR register is sent if TXE = 0, 0xFF if not.
  - When a new byte must be sent and the I2C\_TXDR register has not been written yet, 0xFF is sent.

When an overrun or underrun error is detected, the OVR flag of the I2C\_ISR register is set and an interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set.

### Packet error checking error (PECERR)

A PEC error is detected when the received PEC byte does not match the I2C\_PECR register content. A NACK is automatically sent after the wrong PEC reception.

When a PEC error is detected, the PECERR flag of the I2C\_ISR register is set and an interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set.

### Timeout error (TIMEOUT)

A timeout error occurs for any of these conditions:

- TIDLE = 0 and SCL remains low for the time defined in the TIMEOUTA[11:0] bitfield: this is used to detect an SMBus timeout.
- TIDLE = 1 and both SDA and SCL remains high for the time defined in the TIMEOUTA [11:0] bitfield: this is used to detect a bus idle condition.
- Controller cumulative clock low extend time reaches the time defined in the TIMEOUTB[11:0] bitfield (SMBus  $t_{LOW:MEXT}$  parameter).
- Target cumulative clock low extend time reaches the time defined in the TIMEOUTB[11:0] bitfield (SMBus  $t_{LOW:SEXT}$  parameter).

When a timeout violation is detected in controller mode, a STOP condition is automatically sent.

When a timeout violation is detected in target mode, the SDA and SCL lines are automatically released.

When a timeout error is detected, the TIMEOUT flag is set in the I2C\_ISR register and an interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set.

### Alert (ALERT)

The ALERT flag is set when the I2C peripheral is configured as a host (SMBHEN = 1), the SMBALERT# signal detection is enabled (ALERTEN = 1), and a falling edge is detected on the SMBA pin. An interrupt is generated if the ERRIE bit of the I2C\_CR1 register is set.

## 30.5 I2C in low-power modes

Table 194. Effect of low-power modes to I2C

Mode	Description
Sleep	No effect. I2C interrupts cause the device to exit the Sleep mode.
Stop	The contents of I2C registers are kept.
Standby	The I2C peripheral is powered down. It must be reinitialized after exiting Standby mode.

## 30.6 I2C interrupts

The following table gives the list of I2C interrupt requests.

Table 195. I2C interrupt requests

Interrupt acronym	Interrupt event	Event flag	Enable control bit	Interrupt clear method	Exit Sleep mode	Exit Stop modes	Exit Standby modes
I2C_EV	Receive buffer not empty	RXNE	RXIE	Read I2C_RXDR register	Yes	No	No
	Transmit buffer interrupt status	TXIS	TXIE	Write I2C_TXDR register			
	STOP detection interrupt flag	STOPF	STOPIE	Write STOPCF = 1			
	Transfer complete reload	TCR	TCIE	Write I2C_CR2 with NBYTES[7:0] ≠ 0			
	Transfer complete	TC		Write START = 1 or STOP = 1			
	Address matched	ADDR	ADDRIE	Write ADDRCF=1			
	NACK reception	NACKF	NACKIE	Write NACKCF=1			

**Table 195. I2C interrupt requests (continued)**

Interrupt acronym	Interrupt event	Event flag	Enable control bit	Interrupt clear method	Exit Sleep mode	Exit Stop modes	Exit Standby modes
I2C_ERR	Bus error	BERR	ERRIE	Write BERRCF = 1	Yes	No	No
	Arbitration loss	ARLO		Write ARLOCF = 1			
	Overrun/underrun	OVR		Write OVRCF = 1			
I2C_ERR	PEC error	PECERR	ERRIE	Write PECERRCF = 1	Yes	No	No
	Timeout/ $t_{\text{LOW}}$ error	TIMEOUT		Write TIMEOUTCF = 1			
	SMBus alert	ALERT		Write ALERTCF = 1			

## 30.7 I2C DMA requests

### 30.7.1 Transmission using DMA

DMA (direct memory access) can be enabled for transmission by setting the TXDMAEN bit of the I2C\_CR1 register. Data is loaded from an SRAM area configured through the DMA peripheral (see [Section 8: Direct memory access controller \(DMA\)](#)) to the I2C\_TXDR register whenever the TXIS bit is set.

Only the data are transferred with DMA.

In controller mode, the initialization, the target address, direction, number of bytes and START bit are programmed by software (the transmitted target address cannot be transferred with DMA). When all data are transferred using DMA, DMA must be initialized before setting the START bit. The end of transfer is managed with the NBYTES counter. Refer to [Controller transmitter](#).

In target mode:

- With NOSTRETCH = 0, when all data are transferred using DMA, 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.

The PEC transfer is managed with the counter associated to the NBYTES[7:0] bitfield. Refer to [SMBus target transmitter](#) and [SMBus controller transmitter](#).

*Note:* If DMA is used for transmission, it is not required to set the TXIE bit.

### 30.7.2 Reception using DMA

DMA (direct memory access) can be enabled for reception by setting the RXDMAEN bit of the I2C\_CR1 register. Data is loaded from the I2C\_RXDR register to an SRAM area configured through the DMA peripheral (refer to [Section 8: Direct memory access controller \(DMA\)](#)) whenever the RXNE bit is set. Only the data (including PEC) are transferred with DMA.

In controller mode, the initialization, the target address, direction, number of bytes and START bit are programmed by software. When all data are transferred using DMA, DMA

must be initialized before setting the START bit. The end of transfer is managed with the NBYTES counter.

In target mode with NOSTRETCH = 0, when all data are transferred using DMA, DMA must be initialized before the address match event, or in the ADDR interrupt subroutine, before clearing the ADDR flag.

The PEC transfer is managed with the counter associated to the NBYTES[7:0] bitfield. Refer to [SMBus target receiver](#) and [SMBus controller receiver](#).

*Note:* If DMA is used for reception, it is not required to set the RXIE bit.

## 30.8 I2C debug modes

When the device enters debug mode (core halted), the SMBus timeout either continues working normally or stops, depending on the DBG\_I2Cx\_SMBUS\_TIMEOUT bits in the DBG block.

## 30.9 I2C registers

Refer to [Section 1.2](#) for the list of abbreviations used in register descriptions.

The registers are accessed by words (32-bit).

### 30.9.1 I2C control register 1 (I2C\_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access, until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERT EN	SMBD EN	SMBH EN	GCEN	Res.	NOSTRETCH	SBC
								rw	rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDM AEN	TXDMA EN	Res.	ANF OFF	DNF[3:0]				ERRIE	TCIE	STOP IE	NACKIE	ADDRIE	RXIE	TXIE	PE
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bit 23 **PECEN**: PEC enable

0: PEC calculation disabled

1: PEC calculation enabled

Bit 22 **ALERTEN**: SMBus alert enable

0: The SMBALERT# signal on SMBA pin is not supported in host mode (SMBHEN = 1). In device mode (SMBHEN = 0), the SMBA pin is released and the alert response address header is disabled (0001100x followed by NACK).

1: The SMBALERT# signal on SMBA pin is supported in host mode (SMBHEN = 1). In device mode (SMBHEN = 0), the SMBA pin is driven low and the alert response address header is enabled (0001100x followed by ACK).

*Note: When ALERTEN = 0, the SMBA pin can be used as a standard GPIO.*

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.

Bit 20 **SMBHEN**: SMBus host address enable

0: Host address disabled. Address 0b0001000x is NACKed.  
1: Host address enabled. Address 0b0001000x is ACKed.

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 target mode. It must be kept cleared in controller mode.

0: Clock stretching enabled  
1: Clock stretching disabled

*Note: This bit can be programmed only when the I2C peripheral is disabled (PE = 0).*

Bit 16 **SBC**: Target byte control

This bit is used to enable hardware byte control in target mode.  
0: Target byte control disabled  
1: Target 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 be programmed only when the I2C peripheral is disabled (PE = 0).*

Bits 11:8 **DNF[3:0]**: Digital noise filter

These bits are used to configure the digital noise filter on SDA and SCL input. The digital filter, filters spikes with a length of up to  $DNF[3:0] * t_{I2CCLK}$

0000: Digital filter disabled

0001: Digital filter enabled and filtering capability up to one  $t_{I2CCLK}$

...

1111: digital filter enabled and filtering capability up to fifteen  $t_{I2CCLK}$

*Note: If the analog filter is enabled, the digital filter is added to it. This filter can be programmed only when the I2C peripheral 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 generates 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 generates 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 (target 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 disabled

1: Peripheral enabled

*Note: When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least three APB clock cycles.*

### 30.9.2 I2C control register 2 (I2C\_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to  $2 \times \text{PCLK1} + 6 \times \text{I2CCLK}$ .

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	PECBYTE	AUTOEND	RELOAD	NBYTES[7:0]							
					rs	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NACK	STOP	START	HEAD10R	ADD10	RD_WRN	SADD[9:0]							rs	rs	rs
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:27 Reserved, must be kept at reset value.

Bit 26 **PECBYTE**: Packet error checking byte

This bit is set by software, and cleared by hardware when the PEC is transferred, or when a STOP condition or an Address matched is received, also when PE = 0.

0: No PEC transfer

1: PEC transmission/reception is requested

*Note: Writing 0 to this bit has no effect.*

*This bit has no effect when RELOAD is set, and in target mode when SBC = 0.*

Bit 25 **AUTOEND**: Automatic end mode (controller 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 target mode or when the RELOAD bit is set.*

Bit 24 **RELOAD**: NBYTES reload mode

This bit is set and cleared by software.

0: The transfer is completed after the NBYTES data transfer (STOP or RESTART follows).  
1: The transfer is not completed after the NBYTES data transfer (NBYTES is reloaded). TCR flag is set when NBYTES data are transferred, stretching SCL low.

Bits 23:16 **NBYTES[7:0]**: Number of bytes

The number of bytes to be transmitted/received is programmed there. This field is don't care in target mode with SBC = 0.

*Note: Changing these bits when the START bit is set is not allowed.*

**Bit 15 NACK:** NACK generation (target 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 only in target mode: in controller receiver mode, NACK is automatically generated after last byte preceding STOP or RESTART condition, whatever the NACK bit value.*

*When an overrun occurs in target 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 condition generation

This bit only pertains to controller mode. It is set by software and cleared by hardware when a STOP condition is detected or when PE = 0.

- 0: No STOP generation
- 1: STOP generation after current byte transfer

*Note: Writing 0 to this bit has no effect.*

**Bit 13 START:** START condition generation

This bit is set by software. It is cleared by hardware after the START condition 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 setting the ADDRCF bit of the I2C\_ICR register.

- 0: No START generation
- 1: RESTART/START generation:

If the I2C is already in controller mode with AUTOEND = 0, setting this bit generates a repeated START condition when RELOAD = 0, after the end of the NBYTES transfer.

Otherwise, setting this bit generates a START condition once the bus is free.

*Note: Writing 0 to this bit has no effect.*

*The START bit can be set even if the bus is BUSY or I2C is in target mode.*

*This bit has no effect when RELOAD is set.*

**Bit 12 HEAD10R:** 10-bit address header only read direction (controller receiver mode)

0: The controller sends the complete 10-bit target address read sequence: START + 2 bytes 10-bit address in write direction + RESTART + first seven bits of the 10-bit address in read direction.

- 1: The controller sends only the first seven 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 (controller mode)

- 0: The controller operates in 7-bit addressing mode
- 1: The controller 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 (controller mode)

- 0: Controller requests a write transfer
- 1: Controller requests a read transfer

*Note: Changing this bit when the START bit is set is not allowed.*

Bits 9:0 **SADD[9:0]**: Target address (controller mode)

**Condition: In 7-bit addressing mode (ADD10 = 0):**

SADD[7:1] must be written with the 7-bit target address to be sent. Bits SADD[9], SADD[8] and SADD[0] are don't care.

**Condition: In 10-bit addressing mode (ADD10 = 1):**

SADD[9:0] must be written with the 10-bit target address to be sent.

*Note: Changing these bits when the START bit is set is not allowed.*

### 30.9.3 I2C own address 1 register (I2C\_OAR1)

Address offset: 0x08

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OA1EN	Res.	Res.	Res.	Res.	OA1M ODE	OA1[9:0]									
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bit 15 **OA1EN**: Own address 1 enable

0: Own address 1 disabled. The received target address OA1 is NACKed.

1: Own address 1 enabled. The received target address OA1 is ACKed.

Bits 14:11 Reserved, must be kept at reset value.

Bit 10 **OA1MODE**: Own address 1 10-bit mode

0: Own address 1 is a 7-bit address.

1: Own address 1 is a 10-bit address.

*Note: This bit can be written only when OA1EN = 0.*

Bits 9:0 **OA1[9:0]**: Interface own target address

7-bit addressing mode: OA1[7:1] contains the 7-bit own target address. Bits OA1[9], OA1[8] and OA1[0] are don't care.

10-bit addressing mode: OA1[9:0] contains the 10-bit own target address.

*Note: These bits can be written only when OA1EN = 0.*

### 30.9.4 I2C own address 2 register (I2C\_OAR2)

Address offset: 0x0C

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access, until the previous one is completed. The latency of the second write access can be up to 2x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OA2EN	Res.	Res.	Res.	Res.	Res.	OA2MSK[2:0]		OA2[7:1]							Res.
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:16 Reserved, must be kept at reset value.

Bit 15 **OA2EN**: Own address 2 enable

0: Own address 2 disabled. The received target address OA2 is NACKed.

1: Own address 2 enabled. The received target 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 ≠ 0, the reserved I2C addresses (0b0000xxx and 0b1111xxx) are not acknowledged, even if the comparison matches.*

Bits 7:1 **OA2[7:1]**: Interface address

7-bit addressing mode: 7-bit address

*Note: These bits can be written only when OA2EN = 0.*

Bit 0 Reserved, must be kept at reset value.

### 30.9.5 I2C timing register (I2C\_TIMINGR)

Address offset: 0x10

Reset value: 0x0000 0000

Access: no wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRESC[3:0]				Res.	Res.	Res.	Res.	SCLDEL[3:0]				SDADEL[3:0]			
rw	rw	rw	rw					rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCLH[7:0]								SCLL[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:28 **PRESC[3:0]**: Timing prescaler

This field is used to prescale I2CCLK to generate the clock period  $t_{PRESC}$  used for data setup and hold counters (refer to section [I2C timings](#)), and for SCL high and low level counters (refer to section [I2C controller initialization](#)).

$$t_{PRESC} = (\text{PRESC} + 1) \times t_{I2CCLK}$$

Bits 27:24 Reserved, must be kept at reset value.

Bits 23:20 **SCLDEL[3:0]**: Data setup time

This field is used to generate a delay  $t_{SCLDEL} = (\text{SCLDEL} + 1) \times t_{PRESC}$  between SDA edge and SCL rising edge. In controller and in target modes with NOSTRETCH = 0, the SCL line is stretched low during  $t_{SCLDEL}$ .

*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 controller and in target modes with NOSTRETCH = 0, the SCL line is stretched low during  $t_{SDADEL}$ .

$$t_{SDADEL} = \text{SDADEL} \times t_{PRESC}$$

*Note:*  $t_{SDADEL}$  is used to generate  $t_{HD:DAT}$  timing.

Bits 15:8 **SCLH[7:0]**: SCL high period (controller mode)

This field is used to generate the SCL high period in controller mode.

$$t_{SCLH} = (\text{SCLH} + 1) \times t_{PRESC}$$

*Note:*  $t_{SCLH}$  is also used to generate  $t_{SU:STO}$  and  $t_{HD:STA}$  timing.

Bits 7:0 **SCLL[7:0]**: SCL low period (controller mode)

This field is used to generate the SCL low period in controller mode.

$$t_{SCLL} = (\text{SCLL} + 1) \times t_{PRESC}$$

*Note:*  $t_{SCLL}$  is also used to generate  $t_{BUF}$  and  $t_{SU:STA}$  timings.

*Note:* This register must be configured when the I2C peripheral is disabled ( $PE = 0$ ).

*Note:* The STM32CubeMX tool calculates and provides the I2C\_TIMINGR content in the I2C Configuration window.

### 30.9.6 I2C timeout register (I2C\_TIMEOUTTR)

Address offset: 0x14

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to  $2 \times \text{PCLK1} + 6 \times \text{I2CCLK}$ .

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
TEXTE N	Res.	Res.	Res.	TIMEOUTB[11:0]													
rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
TIMOU TEN	Res.	Res.	TIDLE	TIMEOUTA[11:0]													
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 31 **TEXTE N**: Extended clock timeout enable

0: Extended clock timeout detection is disabled

1: Extended clock timeout detection is enabled. When a cumulative SCL stretch for more than  $t_{\text{LOW:EXT}}$  is done by the I2C interface, a timeout error is detected (TIMEOUT = 1).

Bits 30:28 Reserved, must be kept at reset value.

Bits 27:16 **TIMEOUTB[11:0]**: Bus timeout B

This field is used to configure the cumulative clock extension timeout:

- Controller mode: the controller cumulative clock low extend time ( $t_{\text{LOW:MEXT}}$ ) is detected
- Target mode: the target cumulative clock low extend time ( $t_{\text{LOW:SEXT}}$ ) is detected

$$t_{\text{LOW:EXT}} = (\text{TIMEOUTB} + \text{TIDLE} = 01) \times 2048 \times t_{\text{I2CCLK}}$$

*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.*

### 30.9.7 I2C interrupt and status register (I2C\_ISR)

Address offset: 0x18

Reset value: 0x0000 0001

Access: no wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADD CODE[6:0]								DIR
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
BUSY	Res.	ALERT	TIMEOUT UT	PECERR R	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 (target 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 two MSBs of the address.

Bit 16 DIR: Transfer direction (target mode)

This flag is updated when an address match event occurs (ADDR = 1).

0: Write transfer, target enters receiver mode.

1: Read transfer, target 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, and 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 an 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.*

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.*

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.*

Bit 10 OVR: Overrun/underrun (target mode)

This flag is set by hardware in target 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 target mode. It is cleared by software by setting the BERRCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 7 **TCR**: Transfer complete reload

This flag is set by hardware when RELOAD = 1 and NBYTES data have been transferred. It is cleared by software when NBYTES is written to a non-zero value.

*Note: This bit is cleared by hardware when PE = 0.*

*This flag is only for controller mode, or for target mode when the SBC bit is set.*

Bit 6 **TC**: Transfer complete (controller 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:

- as a controller, provided that the STOP condition is generated by the peripheral.
- as a target, 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 (target mode)

This bit is set by hardware as soon as the received target address matched with one of the enabled target 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 I2C\_RXDR register, and is ready to be read. It is cleared when I2C\_RXDR is read.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 1 **TXIS**: Transmit interrupt status (transmitters)

This bit is set by hardware when the I2C\_TXDR register is empty and the data to be transmitted must be written in the I2C\_TXDR register. It is cleared when the next data to be sent is written in the I2C\_TXDR register.

This bit can be written to 1 by software only when NOSTRETCH = 1, to generate a TXIS event (interrupt if TXIE = 1 or DMA request if TXDMAEN = 1).

*Note: This bit is cleared by hardware when PE = 0.*

Bit 0 **TXE**: Transmit data register empty (transmitters)

This bit is set by hardware when the I2C\_TXDR register is empty. It is cleared when the next data to be sent is written in the I2C\_TXDR register.

This bit can be written to 1 by software in order to flush the transmit data register I2C\_TXDR.

*Note: This bit is set by hardware when PE = 0.*

### 30.9.8 I2C interrupt clear register (I2C\_ICR)

Address offset: 0x1C

Reset value: 0x0000 0000

Access: no wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	ALERTCF	TIMOUTCF	PECCF	OVRCF	ARLOCF	BERRCF	Res.	Res.	STOPCF	NACKCF	ADDRCF	Res.	Res.	Res.
		w	w	w	w	w	w			w	w	w			

Bits 31:14 Reserved, must be kept at reset value.

Bit 13 **ALERTCF**: Alert flag clear

Writing 1 to this bit clears the ALERT flag in the I2C\_ISR register.

Bit 12 **TIMOUTCF**: Timeout detection flag clear

Writing 1 to this bit clears the TIMEOUT flag in the I2C\_ISR register.

Bit 11 **PECCF**: PEC error flag clear

Writing 1 to this bit clears the PECERR flag in the I2C\_ISR register.

Bit 10 **OVRCF**: Overrun/underrun flag clear

Writing 1 to this bit clears the OVR flag in the I2C\_ISR register.

Bit 9 **ARLOCF**: Arbitration lost flag clear

Writing 1 to this bit clears the ARLO flag in the I2C\_ISR register.

Bit 8 **BERRCF**: Bus error flag clear

Writing 1 to this bit clears the BERRF flag in the I2C\_ISR register.

Bits 7:6 Reserved, must be kept at reset value.

Bit 5 **STOPCF**: STOP detection flag clear

Writing 1 to this bit clears the STOPF flag in the I2C\_ISR register.

Bit 4 **NACKCF**: Not acknowledge flag clear

Writing 1 to this bit clears the NACKF flag in I2C\_ISR register.

Bit 3 **ADDRCF**: Address matched flag clear

Writing 1 to this bit clears the ADDR flag in the I2C\_ISR register. Writing 1 to this bit also clears the START bit in the I2C\_CR2 register.

Bits 2:0 Reserved, must be kept at reset value.

### 30.9.9 I2C PEC register (I2C\_PECR)

Address offset: 0x20

Reset value: 0x0000 0000

Access: no wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.								PEC[7:0]							
								r	r	r	r	r	r	r	r

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **PEC[7:0]:** Packet error checking register

This field contains the internal PEC when PECEN=1.

The PEC is cleared by hardware when PE = 0.

### 30.9.10 I2C receive data register (I2C\_RXDR)

Address offset: 0x24

Reset value: 0x0000 0000

Access: no wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.								RXDATA[7:0]							
								r	r	r	r	r	r	r	r

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **RXDATA[7:0]:** 8-bit receive data

Data byte received from the I<sup>2</sup>C-bus.

### 30.9.11 I2C transmit data register (I2C\_TXDR)

Address offset: 0x28

Reset value: 0x0000 0000

Access: no wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TXDATA[7:0]														
								rw	rw	rw	rw	rw	rw	rw	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.

### 30.9.12 I2C register map

The table below provides the I2C register map and the reset values.

**Table 196. I2C register map and reset values**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	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.		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x04	I2C_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	0	0	0	0	0	0	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	I2C_OAR1	Res.	Res.	Res.	Res.	Res.	Res.	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		
0x0C	I2C_OAR2	Res.	Res.	Res.	Res.	Res.	Res.	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		
0x10	I2C_TIMINGR	PRESC[3:0]	SCLDEL [3:0]		SDADEL [3:0]		SCLH[7:0]				SCLL[7:0]				OA1[9:0]														OA2[7:1]		OA2MSK [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	0	0		
0x14	I2C_TIMEOUTR	TEXLEN	TIMEOUTB[11:0]										TIMEOUTA[11:0]																DNF[3:0]		SADD[9:0]		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x18	I2C_ISR	Res.	Res.	Res.	Res.	Res.	Res.	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		
0x1C	I2C_ICR	Res.	Res.	Res.	Res.	Res.	Res.	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		
0x20	I2C_PECR	Res.	Res.	Res.	Res.	Res.	Res.	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		
0x24	I2C_RXDR	Res.	Res.	Res.	Res.	Res.	Res.	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		
0x28	I2C_TXDR	Res.	Res.	Res.	Res.	Res.	Res.	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		

Refer to [Section 2.2: Memory organization](#) for the register boundary addresses.

## 31 Universal synchronous/asynchronous receiver transmitter (USART/UART)

### 31.1 Introduction

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of Full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a programmable baud rate generator.

It supports synchronous one-way communication and half-duplex single-wire communication, as well as multiprocessor communications. It also supports the LIN (Local Interconnect Network), smartcard protocol and IrDA (Infrared Data Association) SIR ENDEC specifications and modem operations (CTS/RTS).

High speed data communication is possible by using the DMA (direct memory access) for multibuffer configuration.

### 31.2 USART main features

- Full-duplex asynchronous communications
- NRZ standard format (mark/space)
- Configurable oversampling method by 16 or 8 to give flexibility between speed and clock tolerance
- A common programmable transmit and receive baud rate of up to 27 Mbit/s when USART clock source is the system clock frequency (Max is 216 MHz) and the oversampling by 8 is used
- Dual clock domain allowing:
  - Convenient baud rate programming independent from the PCLK reprogramming
- Auto baud rate detection
- Programmable data word length (7, 8 or 9 bits)
- Programmable data order with MSB-first or LSB-first shifting
- Configurable stop bits (1 or 2 stop bits)
- Synchronous mode and clock output for synchronous communications
- Single-wire half-duplex communications
- Continuous communications using DMA
- Received/transmitted bytes are buffered in reserved SRAM using centralized DMA
- Separate enable bits for transmitter and receiver
- Separate signal polarity control for transmission and reception
- Swappable Tx/Rx pin configuration
- Hardware flow control for modem and RS-485 transceiver

- Communication control/error detection flags
- Parity control:
  - Transmits parity bit
  - Checks parity of received data byte
- Fourteen interrupt sources with flags
- Multiprocessor communications
  - The USART enters mute mode if the address does not match.
- Wake-up from mute mode (by idle line detection or address mark detection)

### 31.3 USART extended features

- LIN master synchronous break send capability and LIN slave break detection capability
  - 13-bit break generation and 10/11-bit break detection when USART is hardware configured for LIN
- IrDA SIR encoder decoder supporting 3/16 bit duration for normal mode
- Smartcard mode
  - Supports the T=0 and T=1 asynchronous protocols for smartcards as defined in the ISO/IEC 7816-3 standard
  - 0.5 and 1.5 stop bits for smartcard operation
- Support for ModBus communication
  - Timeout feature
  - CR/LF character recognition

## 31.4 USART implementation

Table 197. STM32F75xxx and STM32F74xxx USART features

USART modes/features <sup>(1)</sup>	USART1/USART2/ USART3/USART6	UART4/UART5/ UART7/UART8
Hardware flow control for modem	X	X
Continuous communication using DMA	X	X
Multiprocessor communication	X	X
Synchronous mode	X	-
Smartcard mode	X	-
Single-wire half-duplex communication	X	X
IrDA SIR ENDEC block	X	X
LIN mode	X	X
Dual clock domain	X	X
Receiver timeout interrupt	X	X
Modbus communication	X	X
Auto baud rate detection	X	X
Driver Enable	X	X
USART data length	7 <sup>(2)</sup> , 8 and 9 bits	

1. X = supported.

2. In 7-bit data length mode, smartcard mode, LIN master mode and Auto baud rate (0x7F and 0x55 frames) detection are not supported.

## 31.5 USART functional description

Any USART bidirectional communication requires a minimum of two pins: Receive data In (RX) and Transmit data Out (TX):

- **RX:** Receive data Input.  
This is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.
- **TX:** Transmit data Output.  
When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this I/O is used to transmit and receive the data.

Serial data are transmitted and received through these pins in normal USART mode. The frames are comprised of:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (7, 8 or 9 bits) least significant bit first
- 0.5, 1, 1.5, 2 stop bits indicating that the frame is complete
- The USART interface uses a baud rate generator
- A status register (USART\_ISR)
- Receive and transmit data registers (USART\_RDR, USART\_TDR)
- A baud rate register (USART\_BRR)
- A guard-time register (USART\_GTPR) in case of smartcard mode.

Refer to [Section 31.8: USART registers on page 1053](#) for the definitions of each bit.

The following pin is required to interface in synchronous mode and smartcard mode:

- **CK:** Clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel, data can be received synchronously on RX. This can be used to control peripherals that have shift registers. The clock phase and polarity are software programmable. In smartcard mode, CK output can provide the clock to the smartcard.

The following pins are required in RS232 hardware flow control mode:

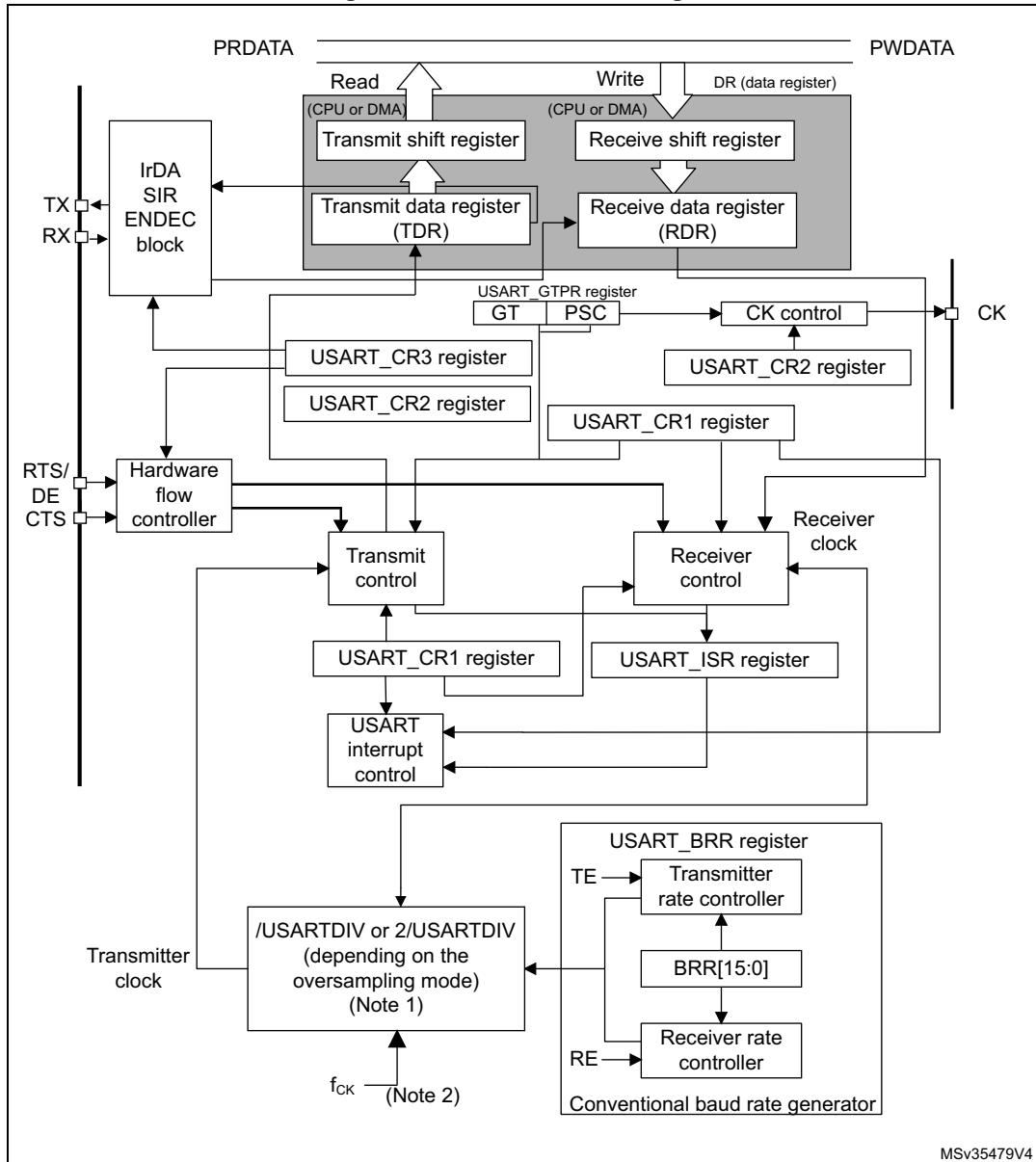
- **CTS:** Clear To Send blocks the data transmission at the end of the current transfer (when high)
- **RTS:** Request to send indicates that the USART is ready to receive data (when low).

The following pin is required in RS485 hardware control mode:

- **DE:** Driver Enable activates the transmission mode of the external transceiver.

Note: *DE and RTS share the same pin.*

Figure 329. USART block diagram



1. For details on coding USARTDIV in the USART\_BRR register, refer to [Section 31.5.4: USART baud rate generation](#).

2. f<sub>CK</sub> can be f<sub>LSE</sub>, f<sub>HSI</sub>, f<sub>PCLK</sub>, f<sub>SYS</sub>.

### 31.5.1 USART character description

The word length can be selected as being either 7 or 8 or 9 bits by programming the M[1:0] bits in the USART\_CR1 register (see [Figure 330](#)).

- 7-bit character length: M[1:0] = 10
- 8-bit character length: M[1:0] = 00
- 9-bit character length: M[1:0] = 01

**Note:** *The 7-bit mode is supported only on some USARTs. In addition, not all modes are supported in 7-bit data length mode. Refer to [Section 31.4: USART implementation](#) for additional information.*

By default, the signal (TX or RX) is in low state during the start bit. It is in high state during the stop bit.

These values can be inverted, separately for each signal, through polarity configuration control.

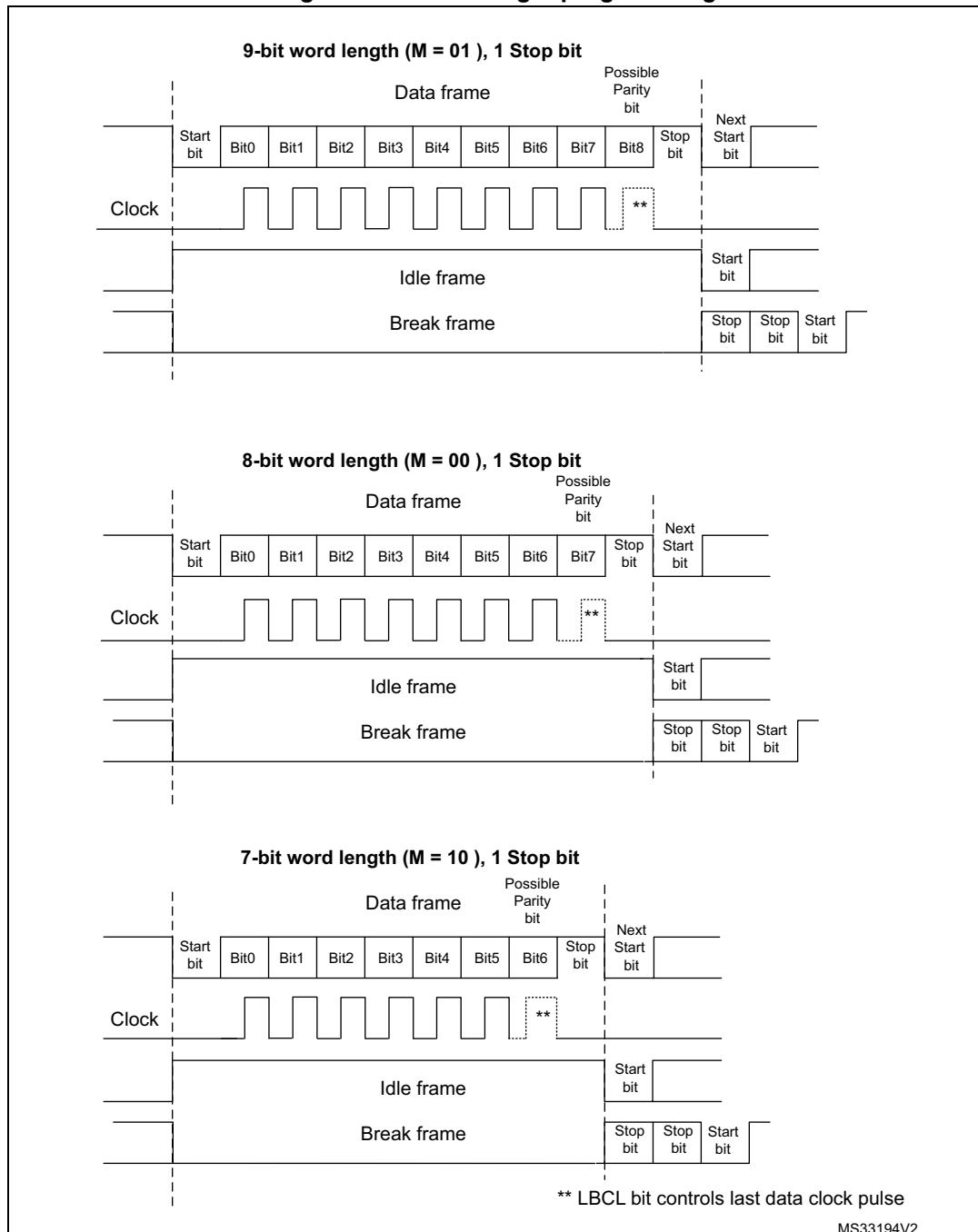
An **Idle character** is interpreted as an entire frame of “1”s (the number of “1”s includes the number of stop bits).

A **Break character** is interpreted on receiving “0”s for a frame period. At the end of the break frame, the transmitter inserts 2 stop bits.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver.

The details of each block is given below.

Figure 330. Word length programming



### 31.5.2 USART transmitter

The transmitter can send data words of either 7, 8 or 9 bits depending on the M bits status. The Transmit Enable bit (TE) must be set in order to activate the transmitter function. The data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

#### Character transmission

During an USART transmission, data shifts out least significant bit first (default configuration) on the TX pin. In this mode, the USART\_TDR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see [Figure 329](#)).

Every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.

The following stop bits are supported by USART: 0.5, 1, 1.5 and 2 stop bits.

*Note:* *The TE bit must be set before writing the data to be transmitted to the USART\_TDR.*

*The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted is lost.*

*An idle frame is sent after the TE bit is enabled.*

#### Configurable stop bits

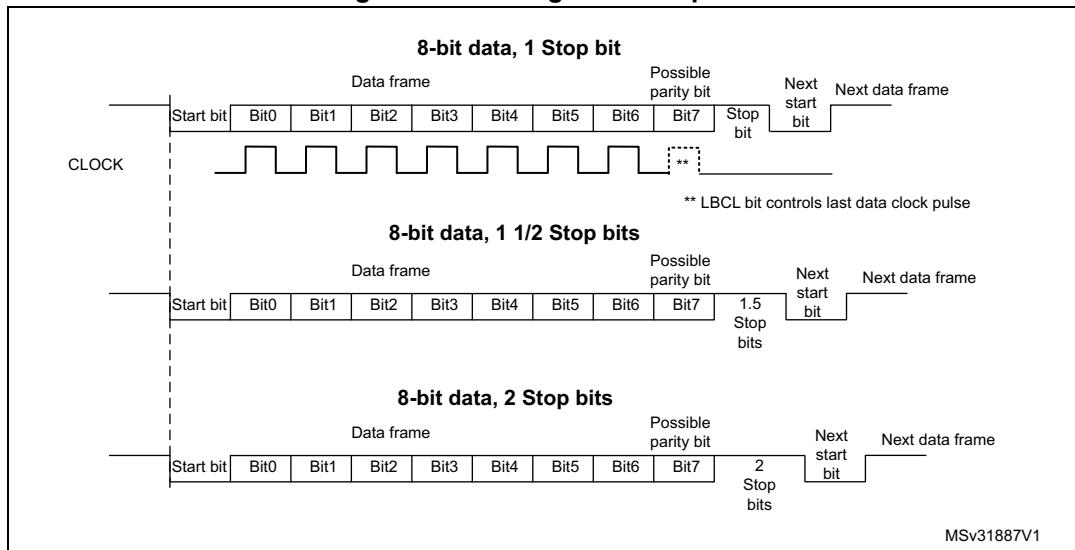
The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

- **1 stop bit:** This is the default value of number of stop bits.
- **2 stop bits:** This is supported by normal USART, single-wire and modem modes.
- **1.5 stop bits:** To be used in smartcard mode.
- **0.5 stop bit:** To be used when receiving data in smartcard mode.

An idle frame transmission will include the stop bits.

A break transmission is 10 low bits (when M[1:0] = 00) or 11 low bits (when M[1:0] = 01) or 9 low bits (when M[1:0] = 10) followed by 2 stop bits (see [Figure 331](#)). It is not possible to transmit long breaks (break of length greater than 9/10/11 low bits).

Figure 331. Configurable stop bits



### Character transmission procedure

1. Program the M bits in USART\_CR1 to define the word length.
2. Select the desired baud rate using the USART\_BRR register.
3. Program the number of stop bits in USART\_CR2.
4. Enable the USART by writing the UE bit in USART\_CR1 register to 1.
5. Select DMA enable (DMAT) in USART\_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the TE bit in USART\_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART\_TDR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART\_TDR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.

### Single byte communication

Clearing the TXE bit is always performed by a write to the transmit data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from the USART\_TDR register to the shift register and the data transmission has started.
- The USART\_TDR register is empty.
- The next data can be written in the USART\_TDR register without overwriting the previous data.

This flag generates an interrupt if the TXEIE bit is set.

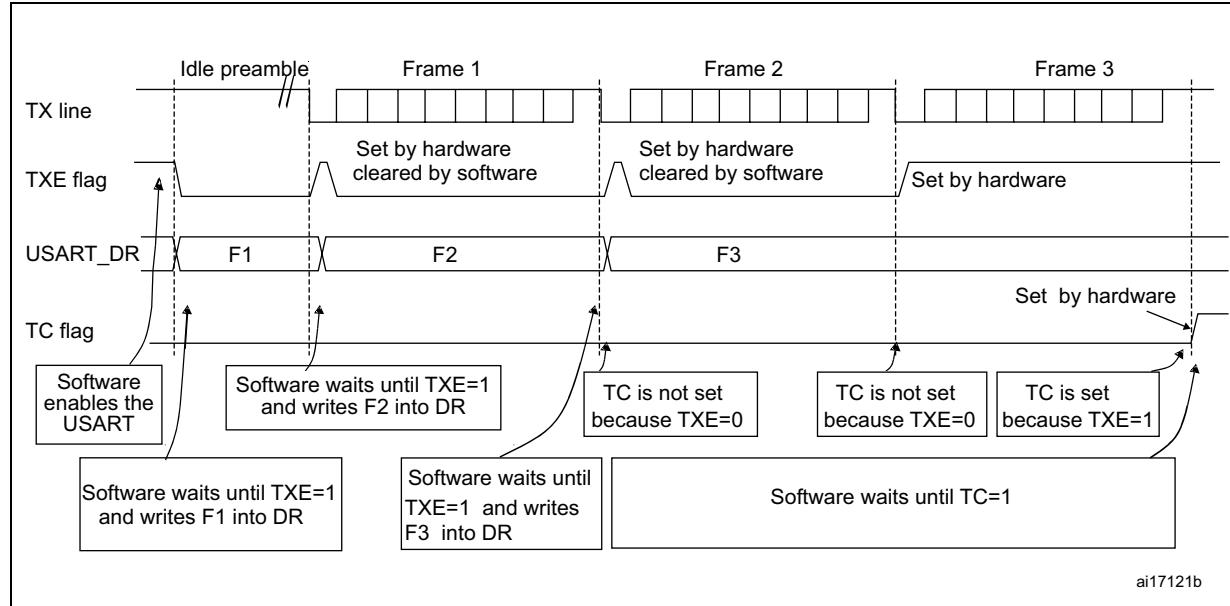
When a transmission is taking place, a write instruction to the USART\_TDR register stores the data in the TDR register; next, the data is copied in the shift register at the end of the currently ongoing transmission.

When no transmission is taking place, a write instruction to the USART\_TDR register places the data in the shift register, the data transmission starts, and the TXE bit is set.

If a frame is transmitted (after the stop bit) and the TXE bit is set, the TC bit goes high. An interrupt is generated if the TCIE bit is set in the USART\_CR1 register.

After writing the last data in the USART\_TDR register, it is mandatory to wait for TC=1 before disabling the USART or causing the microcontroller to enter the low-power mode (see [Figure 332: TC/TXE behavior when transmitting](#)).

**Figure 332. TC/TXE behavior when transmitting**



### Break characters

Setting the SBKRQ bit transmits a break character. The break frame length depends on the M bits (see [Figure 330](#)).

If a '1' is written to the SBKRQ bit, a break character is sent on the TX line after completing the current character transmission. The SBKF bit is set by the write operation and it is reset by hardware when the break character is completed (during the stop bits after the break character). The USART inserts a logic 1 signal (STOP) for the duration of 2 bits at the end of the break frame to guarantee the recognition of the start bit of the next frame.

In the case the application needs to send the break character following all previously inserted data, including the ones not yet transmitted, the software should wait for the TXE flag assertion before setting the SBKRQ bit.

### Idle characters

Setting the TE bit drives the USART to send an idle frame before the first data frame.

### 31.5.3 USART receiver

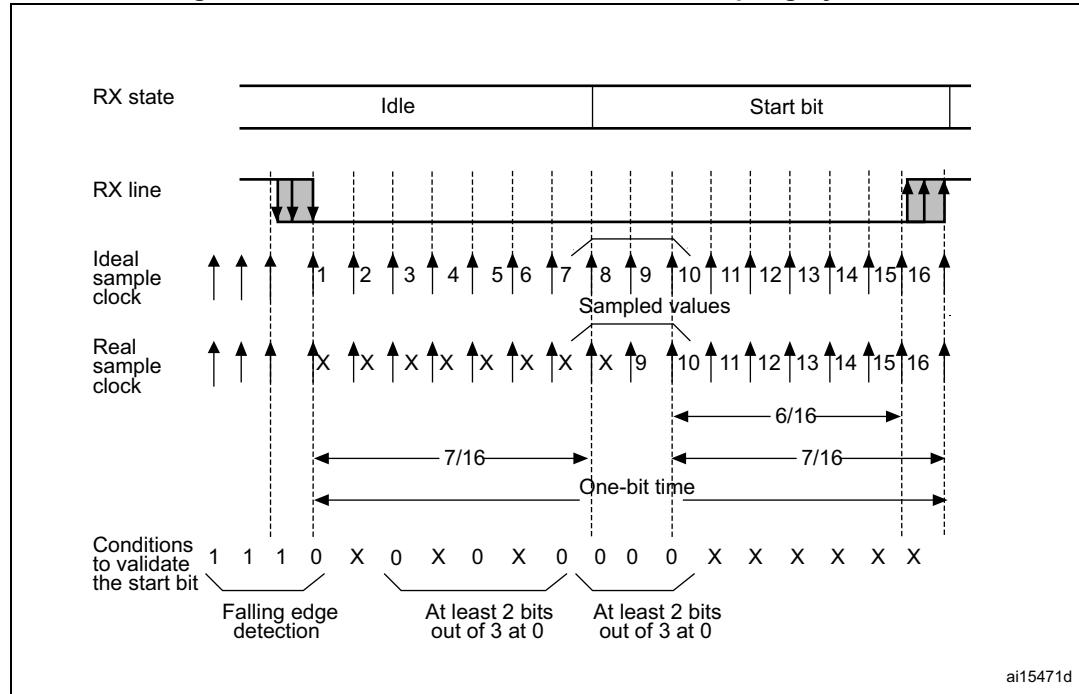
The USART can receive data words of either 7, 8 or 9 bits depending on the M bits in the USART\_CR1 register.

### Start bit detection

The start bit detection sequence is the same when oversampling by 16 or by 8.

In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0 X 0 X 0 X 0 X 0.

**Figure 333. Start bit detection when oversampling by 16 or 8**



ai15471d

**Note:** If the sequence is not complete, the start bit detection aborts and the receiver returns to the idle state (no flag is set), where it waits for a falling edge.

The start bit is confirmed (RXNE flag set, interrupt generated if RXNEIE=1) if the 3 sampled bits are at 0 (first sampling on the 3rd, 5th and 7th bits finds the 3 bits at 0 and second sampling on the 8th, 9th and 10th bits also finds the 3 bits at 0).

The start bit is validated (RXNE flag set, interrupt generated if RXNEIE=1) but the NF noise flag is set if,

- a) for both samplings, 2 out of the 3 sampled bits are at 0 (sampling on the 3rd, 5th and 7th bits and sampling on the 8th, 9th and 10th bits)
- or
- b) for one of the samplings (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), 2 out of the 3 bits are found at 0.

If neither conditions a. or b. are met, the start detection aborts and the receiver returns to the idle state (no flag is set).

## Character reception

During an USART reception, data shifts in least significant bit first (default configuration) through the RX pin. In this mode, the USART\_RDR register consists of a buffer (RDR) between the internal bus and the receive shift register.

### Character reception procedure

1. Program the M bits in USART\_CR1 to define the word length.
2. Select the desired baud rate using the baud rate register USART\_BRR
3. Program the number of stop bits in USART\_CR2.
4. Enable the USART by writing the UE bit in USART\_CR1 register to 1.
5. Select DMA enable (DMAR) in USART\_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the RE bit USART\_CR1. This enables the receiver which begins searching for a start bit.

When a character is received:

- The RXNE bit is set to indicate that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception. PE flag can also be set with RXNE.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read of the Receive data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART\_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART\_RQR register. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

### Break character

When a break character is received, the USART handles it as a framing error.

### Idle character

When an idle frame is detected, there is the same procedure as for a received data character plus an interrupt if the IDLEIE bit is set.

### Overrun error

An overrun error occurs when a character is received when RXNE has not been reset. Data can not be transferred from the shift register to the RDR register until the RXNE bit is cleared.

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced. When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART\_RDR is performed.
- The shift register are overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or EIE bit is set.
- The ORE bit is reset by setting the ORECF bit in the ICR register.

*Note:*

*The ORE bit, when set, indicates that at least 1 datum has been lost. There are two possibilities:*

- if RXNE=1, then the last valid data is stored in the receive register RDR and can be read,
- if RXNE=0, then it means that the last valid data has already been read and thus there is nothing to be read in the RDR. This case can occur when the last valid data is read in the RDR at the same time as the new (and lost) data is received.

### Selecting the clock source and the proper oversampling method

The choice of the clock source is done through the Clock Control system (see Section Reset and clock control (RCC))). The clock source must be chosen before enabling the USART (by setting the UE bit).

The clock source frequency is  $f_{CK}$ .

When the dual clock domain is supported, the clock source can be one of the following sources: PCLK (default), LSE, HSI or SYSCLK.

The communication speed range (specially the maximum communication speed) is also determined by the clock source.

The receiver implements different user-configurable oversampling techniques for data recovery by discriminating between valid incoming data and noise. This allows a trade-off between the maximum communication speed and noise/clock inaccuracy immunity.

The oversampling method can be selected by programming the OVER8 bit in the USART\_CR1 register and can be either 16 or 8 times the baud rate clock ([Figure 334](#) and [Figure 335](#)).

Depending on the application:

- Select oversampling by 8 (OVER8=1) to achieve higher speed (up to  $f_{CK}/8$ ). In this case the maximum receiver tolerance to clock deviation is reduced (refer to [Section 31.5.5: Tolerance of the USART receiver to clock deviation on page 1028](#))
- Select oversampling by 16 (OVER8=0) to increase the tolerance of the receiver to clock deviations. In this case, the maximum speed is limited to maximum  $f_{CK}/16$  where  $f_{CK}$  is the clock source frequency.

Programming the ONEBIT bit in the USART\_CR3 register selects the method used to evaluate the logic level. There are two options:

- The majority vote of the three samples in the center of the received bit. In this case, when the 3 samples used for the majority vote are not equal, the NF bit is set
- A single sample in the center of the received bit

Depending on the application:

- select the three samples' majority vote method (ONEBIT=0) when operating in a noisy environment and reject the data when a noise is detected (refer to [Figure 198](#)) because this indicates that a glitch occurred during the sampling.
- select the single sample method (ONEBIT=1) when the line is noise-free to increase the receiver's tolerance to clock deviations (see [Section 31.5.5: Tolerance of the USART receiver to clock deviation on page 1028](#)). In this case the NF bit will never be set.

When noise is detected in a frame:

- The NF bit is set at the rising edge of the RXNE bit.
- The invalid data is transferred from the Shift register to the USART\_RDR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt is issued if the EIE bit is set in the USART\_CR3 register.

The NF bit is reset by setting NFCF bit in ICR register.

**Note:** *Oversampling by 8 is not available in LIN, smartcard and IrDA modes. In those modes, the OVER8 bit is forced to '0' by hardware.*

**Figure 334. Data sampling when oversampling by 16**

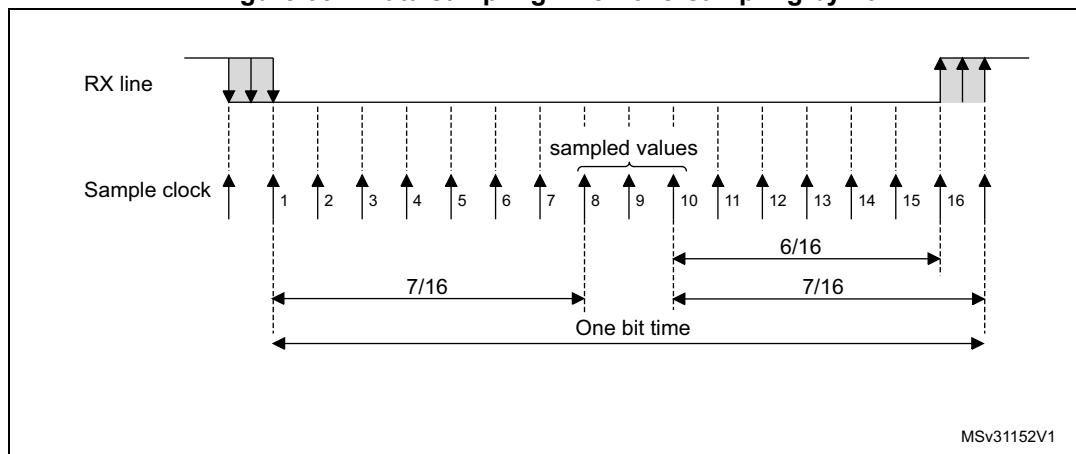


Figure 335. Data sampling when oversampling by 8

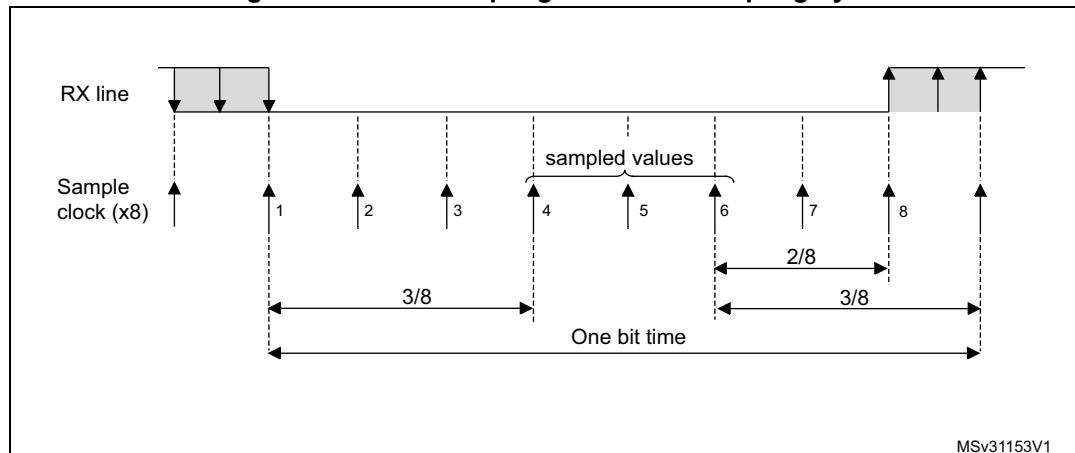


Table 198. Noise detection from sampled data

Sampled value	NE status	Received bit value
000	0	0
001	1	0
010	1	0
011	1	1
100	1	0
101	1	1
110	1	1
111	0	1

### Framing error

A framing error is detected when the stop bit is not recognized on reception at the expected time, following either a de-synchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data is transferred from the Shift register to the USART\_RDR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt is issued if the EIE bit is set in the USART\_CR3 register.

The FE bit is reset by writing 1 to the FECF in the USART\_ICR register.

### Configurable stop bits during reception

The number of stop bits to be received can be configured through the control bits of Control Register 2 - it can be either 1 or 2 in normal mode and 0.5 or 1.5 in smartcard mode.

- **0.5 stop bit (reception in smartcard mode):** No sampling is done for 0.5 stop bit. As a consequence, no framing error and no break frame can be detected when 0.5 stop bit is selected.
- **1 stop bit:** Sampling for 1 stop Bit is done on the 8th, 9th and 10th samples.
- **1.5 stop bits (smartcard mode):** When transmitting in smartcard mode, the device must check that the data is correctly sent. Thus the receiver block must be enabled (RE =1 in the USART\_CR1 register) and the stop bit is checked to test if the smartcard has detected a parity error. In the event of a parity error, the smartcard forces the data signal low during the sampling - NACK signal-, which is flagged as a framing error. Then, the FE flag is set with the RXNE at the end of the 1.5 stop bits. Sampling for 1.5 stop bits is done on the 16th, 17th and 18th samples (1 baud clock period after the beginning of the stop bit). The 1.5 stop bits can be decomposed into 2 parts: one 0.5 baud clock period during which nothing happens, followed by 1 normal stop bit period during which sampling occurs halfway through. Refer to [Section 31.5.13: USART smartcard mode on page 1039](#) for more details.
- **2 stop bits:** Sampling for 2 stop bits is done on the 8th, 9th and 10th samples of the first stop bit. If a framing error is detected during the first stop bit the framing error flag is set. The second stop bit is not checked for framing error. The RXNE flag is set at the end of the first stop bit.

### 31.5.4 USART baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the USART\_BRR register.

#### Equation 1: Baud rate for standard USART (SPI mode included) (OVER8 = 0 or 1)

In case of oversampling by 16, the equation is:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\text{USARTDIV}}$$

In case of oversampling by 8, the equation is:

$$\text{Tx/Rx baud} = \frac{2 \times f_{CK}}{\text{USARTDIV}}$$

#### Equation 2: Baud rate in smartcard, LIN and IrDA modes (OVER8 = 0)

In smartcard, LIN and IrDA modes, only oversampling by 16 is supported:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\text{USARTDIV}}$$

USARTDIV is an unsigned fixed point number that is coded on the USART\_BRR register.

- When OVER8 = 0, BRR = USARTDIV.
- When OVER8 = 1
  - BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.
  - BRR[3] must be kept cleared.
  - BRR[15:4] = USARTDIV[15:4]

**Note:** *The baud counters are updated to the new value in the baud registers after a write operation to USART\_BRR. Hence the baud rate register value should not be changed during communication.*

*In case of oversampling by 16 or 8, USARTDIV must be greater than or equal to 16d.*

### How to derive USARTDIV from USART\_BRR register values

#### Example 1

To obtain 9600 baud with  $f_{CK} = 8$  MHz.

- In case of oversampling by 16:  
USARTDIV =  $8\ 000\ 000/9600$   
BRR = USARTDIV =  $0d833 = 0x341$
- In case of oversampling by 8:  
USARTDIV =  $2 * 8\ 000\ 000/9600$   
USARTDIV =  $1666,66$  ( $0d1667 = 0x683$ )  
BRR[3:0] =  $0x3 >> 1 = 0x1$   
BRR =  $0x681$

#### Example 2

To obtain 921.6 kbaud with  $f_{CK} = 48$  MHz.

- In case of oversampling by 16:  
USARTDIV =  $48\ 000\ 000/921\ 600$   
BRR = USARTDIV =  $52d = 34h$
- In case of oversampling by 8:  
USARTDIV =  $2 * 48\ 000\ 000/921\ 600$   
USARTDIV =  $104$  ( $104d = 68h$ )  
BRR[3:0] = USARTDIV[3:0]  $>> 1 = 8h >> 1 = 4h$   
BRR =  $0x64$

**Table 199. Error calculation for programmed baud rates at  $f_{CK} = 216$  MHz  
in both cases of oversampling by 8 (OVER8 = 1)<sup>(1)</sup>**

Desired baud rate (Bps)	Actual baud rate (Bps)	BRR	Error
9600	9600	AFC4	0.00000000
19200	19200	57E2	0.00000000
38400	38400	2BF1	0.00000000
57600	57600	1D46	0.00000000
115200	115200	EA3	0.00000000
230400	230400	751	0.00000000
460800	461538.461	3A4	0.160256293
921600	923076.923	1D2	0.001602564
13500000	13500000.000	20	0.00000000
27000000	27000000.000	10	0.00000000

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

**Table 200. Error calculation for programmed baud rates at  $f_{CK} = 216$  MHz  
in both cases of oversampling by 16 (OVER8 = 0)<sup>(1)</sup>**

Desired baud rate (Bps)	Actual baud rate (Bps)	BRR	Error
9600	9600.000	57E4	0.00000000
19200	19200.000	2BF2	0.00000000
38400	38400.000	15F9	0.00000000
57600	57600.000	EA6	0.00000000
115200	115200.000	753	0.00000000
230400	230522.946	3A9	0.05336179
460800	461538.462	1D4	0.16025641
921600	923076.923	EA	0.16025641
4000000	4000000.000	36	0.00000000
6000000	6000000.000	24	0.00000000
10000000	10285714.286	15	2.85714286
13500000	13500000.000	10	0.00000000

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

### 31.5.5 Tolerance of the USART receiver to clock deviation

The asynchronous receiver of the USART works correctly only if the total clock system deviation is less than the tolerance of the USART receiver. The causes which contribute to the total deviation are:

- DTRA: Deviation due to the transmitter error (which also includes the deviation of the transmitter's local oscillator)
- DQUANT: Error due to the baud rate quantization of the receiver
- DREC: Deviation of the receiver's local oscillator
- DTCL: Deviation due to the transmission line (generally due to the transceivers which can introduce an asymmetry between the low-to-high transition timing and the high-to-low transition timing)

$$\text{DTRA} + \text{DQUANT} + \text{DREC} + \text{DTCL} < \text{USART receiver's tolerance}$$

The USART receiver can receive data correctly at up to the maximum tolerated deviation specified in [Table 201](#) and [Table 202](#) depending on the following choices:

- 9-, 10- or 11-bit character length defined by the M bits in the USART\_CR1 register
- Oversampling by 8 or 16 defined by the OVER8 bit in the USART\_CR1 register
- Bits BRR[3:0] of USART\_BRR register are equal to or different from 0000.
- Use of 1 bit or 3 bits to sample the data, depending on the value of the ONEBIT bit in the USART\_CR3 register.

**Table 201. Tolerance of the USART receiver when BRR [3:0] = 0000**

M bits	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
00	3.75%	4.375%	2.50%	3.75%
01	3.41%	3.97%	2.27%	3.41%
10	4.16%	4.86%	2.77%	4.16%

**Table 202. Tolerance of the USART receiver when BRR [3:0] is different from 0000**

M bits	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
00	3.33%	3.88%	2%	3%
01	3.03%	3.53%	1.82%	2.73%
10	3.7%	4.31%	2.22%	3.33%

Note:

The data specified in [Table 201](#) and [Table 202](#) may slightly differ in the special case when the received frames contain some Idle frames of exactly 10-bit durations when M bits = 00 (11-bit durations when M bits =01 or 9- bit durations when M bits = 10).

### 31.5.6 USART auto baud rate detection

The USART is able to detect and automatically set the USART\_BRR register value based on the reception of one character. Automatic baud rate detection is useful under two circumstances:

- The communication speed of the system is not known in advance
- The system is using a relatively low accuracy clock source and this mechanism allows the correct baud rate to be obtained without measuring the clock deviation.

The clock source frequency must be compatible with the expected communication speed (when oversampling by 16, the baud rate is between  $f_{CK}/65535$  and  $f_{CK}/16$ . when oversampling by 8, the baud rate is between  $f_{CK}/65535$  and  $f_{CK}/8$ ).

Before activating the auto baud rate detection, the auto baud rate detection mode must be chosen. There are various modes based on different character patterns.

They can be chosen through the ABRMOD[1:0] field in the USART\_CR2 register. In these auto baud rate modes, the baud rate is measured several times during the synchronization data reception and each measurement is compared to the previous one.

These modes are:

- **Mode 0:** Any character starting with a bit at 1. In this case the USART measures the duration of the Start bit (falling edge to rising edge).
- **Mode 1:** Any character starting with a 10xx bit pattern. In this case, the USART measures the duration of the Start and of the 1st data bit. The measurement is done falling edge to falling edge, ensuring better accuracy in the case of slow signal slopes.
- **Mode 2:** A 0x7F character frame (it may be a 0x7F character in LSB first mode or a 0xFE in MSB first mode). In this case, the baud rate is updated first at the end of the start bit (BRs), then at the end of bit 6 (based on the measurement done from falling edge to falling edge: BR6). Bit 0 to bit 6 are sampled at BRs while further bits of the character are sampled at BR6.
- **Mode 3:** A 0x55 character frame. In this case, the baud rate is updated first at the end of the start bit (BRs), then at the end of bit 0 (based on the measurement done from falling edge to falling edge: BR0), and finally at the end of bit 6 (BR6). Bit 0 is sampled at BRs, bit 1 to bit 6 are sampled at BR0, and further bits of the character are sampled at BR6.

In parallel, another check is performed for each intermediate transition of RX line. An error is generated if the transitions on RX are not sufficiently synchronized with the receiver (the receiver being based on the baud rate calculated on bit 0).

Prior to activating auto baud rate detection, the USART\_BRR register must be initialized by writing a non-zero baud rate value.

The automatic baud rate detection is activated by setting the ABREN bit in the USART\_CR2 register. The USART will then wait for the first character on the RX line. The auto baud rate operation completion is indicated by the setting of the ABRF flag in the USART\_ISR register. If the line is noisy, the correct baud rate detection cannot be guaranteed. In this case the BRR value may be corrupted and the ABRE error flag is set. This also happens if the communication speed is not compatible with the automatic baud rate detection range (bit duration not between 16 and 65536 clock periods (oversampling by 16) and not between 8 and 65536 clock periods (oversampling by 8)).

The RXNE interrupt will signal the end of the operation.

At any later time, the auto baud rate detection may be relaunched by resetting the ABRF flag (by writing a 0).

**Note:** *If the USART is disabled (UE=0) during an auto baud rate operation, the BRR value may be corrupted.*

### 31.5.7 Multiprocessor communication using USART

In multiprocessor communication, the following bits are to be kept cleared:

- LINEN bit in the USART\_CR2 register,
- HDSEL, IREN and SCEN bits in the USART\_CR3 register.

It is possible to perform multiprocessor communication with the USART (with several USARTs connected in a network). For instance one of the USARTs can be the master, its TX output connected to the RX inputs of the other USARTs. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master.

In multiprocessor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant USART service overhead for all non addressed receivers.

The non addressed devices may be placed in mute mode by means of the muting function. In order to use the mute mode feature, the MME bit must be set in the USART\_CR1 register.

In mute mode:

- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in USART\_ISR register is set to 1. RWU can be controlled automatically by hardware or by software, through the MMRQ bit in the USART\_RQR register, under certain conditions.

The USART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the USART\_CR1 register:

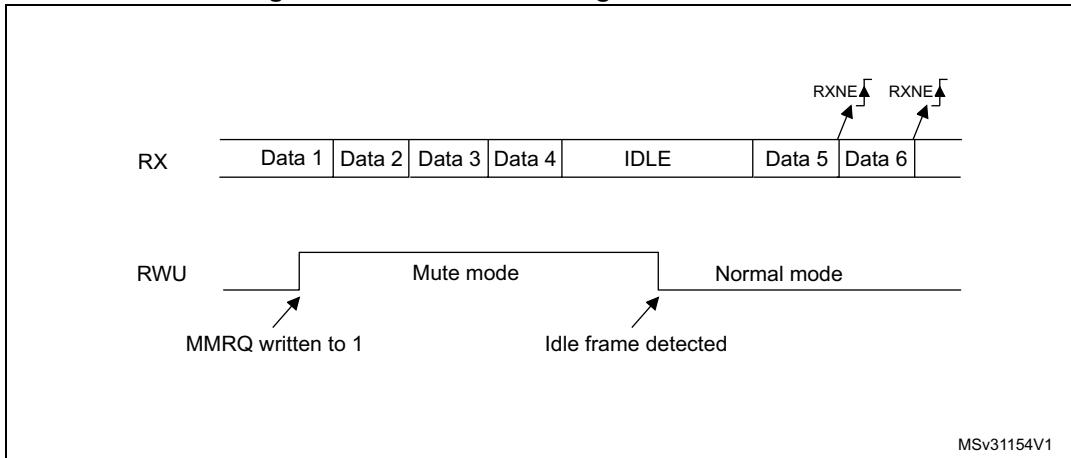
- Idle Line detection if the WAKE bit is reset,
- Address Mark detection if the WAKE bit is set.

### Idle line detection (WAKE=0)

The USART enters mute mode when the MMRQ bit is written to 1 and the RWU is automatically set.

It wakes up when an Idle frame is detected. Then the RWU bit is cleared by hardware but the IDLE bit is not set in the USART\_ISR register. An example of mute mode behavior using Idle line detection is given in [Figure 336](#).

**Figure 336. Mute mode using Idle line detection**



**Note:** If the MMRQ is set while the IDLE character has already elapsed, mute mode will not be entered (RWU is not set).

If the USART is activated while the line is idle, the idle state is detected after the duration of one IDLE frame (not only after the reception of one character frame).

### 4-bit/7-bit address mark detection (WAKE=1)

In this mode, bytes are recognized as addresses if their MSB is a '1' otherwise they are considered as data. In an address byte, the address of the targeted receiver is put in the 4 or 7 LSBs. The choice of 7 or 4-bit address detection is done using the ADDM7 bit. This 4-bit/7-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the USART\_CR2 register.

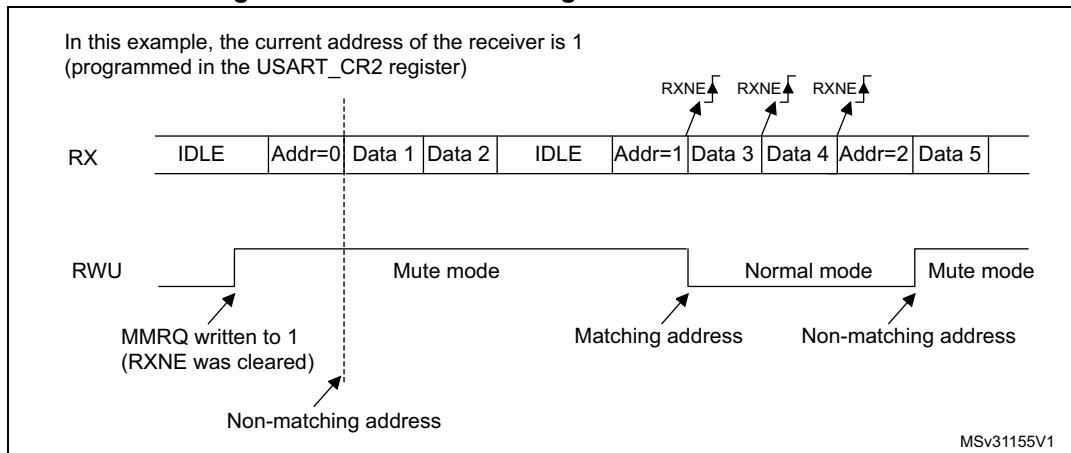
**Note:** In 7-bit and 9-bit data modes, address detection is done on 6-bit and 8-bit addresses (ADD[5:0] and ADD[7:0]) respectively.

The USART enters mute mode when an address character is received which does not match its programmed address. In this case, the RWU bit is set by hardware. The RXNE flag is not set for this address byte and no interrupt or DMA request is issued when the USART enters mute mode.

The USART also enters mute mode when the MMRQ bit is written to 1. The RWU bit is also automatically set in this case.

The USART exits from mute mode when an address character is received which matches the programmed address. Then the RWU bit is cleared and subsequent bytes are received normally. The RXNE bit is set for the address character since the RWU bit has been cleared.

An example of mute mode behavior using address mark detection is given in [Figure 337](#).

**Figure 337. Mute mode using address mark detection**

### 31.5.8 Modbus communication using USART

The USART offers basic support for the implementation of Modbus/RTU and Modbus/ASCII protocols. Modbus/RTU is a half duplex, block transfer protocol. The control part of the protocol (address recognition, block integrity control and command interpretation) must be implemented in software.

The USART offers basic support for the end of the block detection, without software overhead or other resources.

#### Modbus/RTU

In this mode, the end of one block is recognized by a “silence” (idle line) for more than 2 character times. This function is implemented through the programmable timeout function.

The timeout function and interrupt must be activated, through the RTOEN bit in the USART\_CR2 register and the RTOIE in the USART\_CR1 register. The value corresponding to a timeout of 2 character times (for example 22 x bit duration) must be programmed in the RTO register. When the receive line is idle for this duration, after the last stop bit is received, an interrupt is generated, informing the software that the current block reception is completed.

#### Modbus/ASCII

In this mode, the end of a block is recognized by a specific (CR/LF) character sequence. The USART manages this mechanism using the character match function.

By programming the LF ASCII code in the ADD[7:0] field and by activating the character match interrupt (CMIE=1), the software is informed when a LF has been received and can check the CR/LF in the DMA buffer.

### 31.5.9 USART parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCE bit in the USART\_CR1 register. Depending on the frame length defined by the M bits, the possible USART frame formats are as listed in [Table 203](#).

**Table 203. Frame formats**

M bits	PCE bit	USART frame <sup>(1)</sup>
00	0	SB   8-bit data   STB
00	1	SB   7-bit data   PB   STB
01	0	SB   9-bit data   STB
01	1	SB   8-bit data   PB   STB
10	0	SB   7-bit data   STB
10	1	SB   6-bit data   PB   STB

- Legends: SB: start bit, STB: stop bit, PB: parity bit. In the data register, the PB is always taking the MSB position (9th, 8th or 7th, depending on the M bits value).

#### Even parity

The parity bit is calculated to obtain an even number of “1s” inside the frame of the 6, 7 or 8 LSB bits (depending on M bits values) and the parity bit.

As an example, if data=00110101, and 4 bits are set, then the parity bit is 0 if even parity is selected (PS bit in USART\_CR1 = 0).

#### Odd parity

The parity bit is calculated to obtain an odd number of “1s” inside the frame made of the 6, 7 or 8 LSB bits (depending on M bits values) and the parity bit.

As an example, if data=00110101 and 4 bits set, then the parity bit is 1 if odd parity is selected (PS bit in USART\_CR1 = 1).

#### Parity checking in reception

If the parity check fails, the PE flag is set in the USART\_ISR register and an interrupt is generated if PEIE is set in the USART\_CR1 register. The PE flag is cleared by software writing 1 to the PECF in the USART\_ICR register.

#### Parity generation in transmission

If the PCE bit is set in USART\_CR1, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit (even number of “1s” if even parity is selected (PS=0) or an odd number of “1s” if odd parity is selected (PS=1)).

### 31.5.10 USART LIN (local interconnection network) mode

This section is relevant only when LIN mode is supported. Please refer to [Section 31.4: USART implementation on page 1012](#).

The LIN mode is selected by setting the LINEN bit in the USART\_CR2 register. In LIN mode, the following bits must be kept cleared:

- STOP[1:0] and CLKEN in the USART\_CR2 register,
- SCEN, HDSEL and IREN in the USART\_CR3 register.

#### LIN transmission

The procedure explained in [Section 31.5.2: USART transmitter](#) has to be applied for LIN Master transmission. It must be the same as for normal USART transmission with the following differences:

- Clear the M bits to configure 8-bit word length.
- Set the LINEN bit to enter LIN mode. In this case, setting the SBKRQ bit sends 13 '0' bits as a break character. Then 2 bits of value '1' are sent to allow the next start detection.

#### LIN reception

When LIN mode is enabled, the break detection circuit is activated. The detection is totally independent from the normal USART receiver. A break can be detected whenever it occurs, during Idle state or during a frame.

When the receiver is enabled (RE=1 in USART\_CR1), the circuit looks at the RX input for a start signal. The method for detecting start bits is the same when searching break characters or data. After a start bit has been detected, the circuit samples the next bits exactly like for the data (on the 8th, 9th and 10th samples). If 10 (when the LBDL = 0 in USART\_CR2) or 11 (when LBDL=1 in USART\_CR2) consecutive bits are detected as '0', and are followed by a delimiter character, the LBDF flag is set in USART\_ISR. If the LBDIE bit=1, an interrupt is generated. Before validating the break, the delimiter is checked for as it signifies that the RX line has returned to a high level.

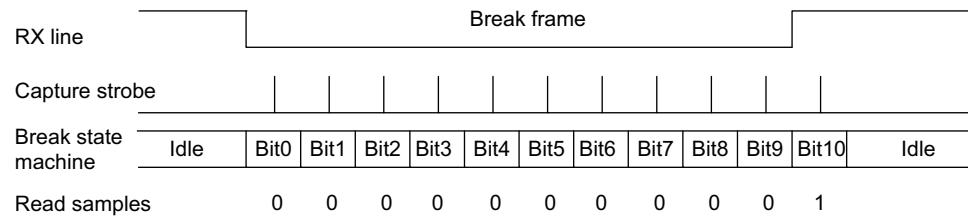
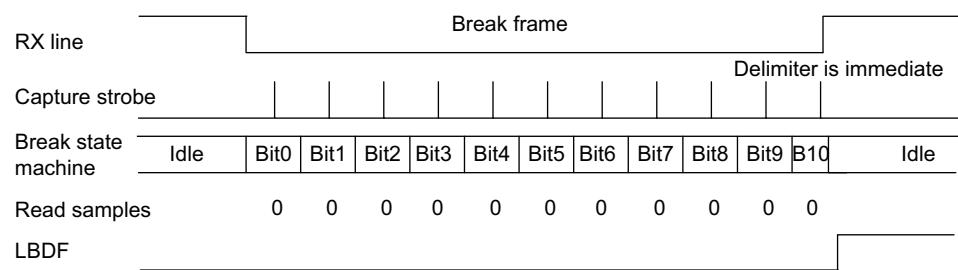
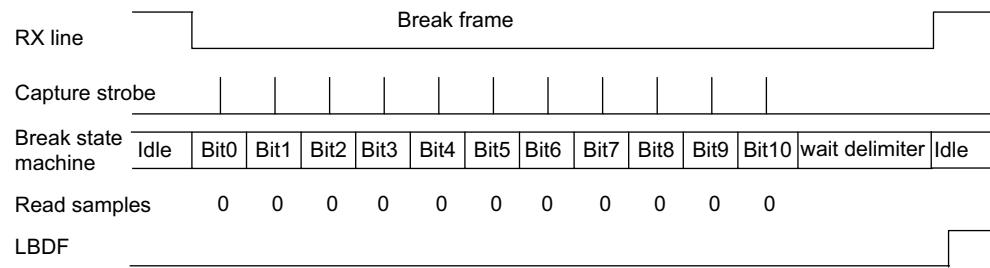
If a '1' is sampled before the 10 or 11 have occurred, the break detection circuit cancels the current detection and searches for a start bit again.

If the LIN mode is disabled (LINEN=0), the receiver continues working as normal USART, without taking into account the break detection.

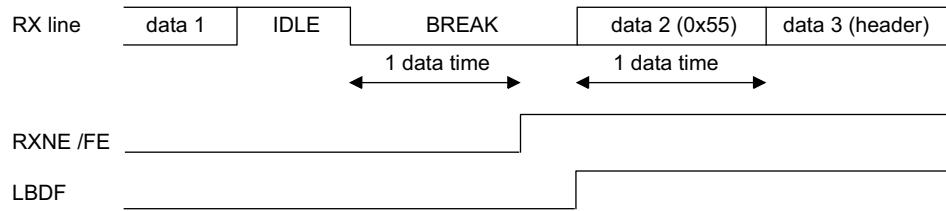
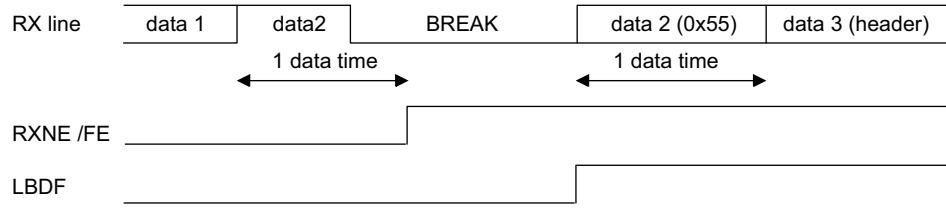
If the LIN mode is enabled (LINEN=1), as soon as a framing error occurs (i.e. stop bit detected at '0', which is the case for any break frame), the receiver stops until the break detection circuit receives either a '1', if the break word was not complete, or a delimiter character if a break has been detected.

The behavior of the break detector state machine and the break flag is shown on the [Figure 338: Break detection in LIN mode \(11-bit break length - LBDL bit is set\) on page 1035](#).

Examples of break frames are given on [Figure 339: Break detection in LIN mode vs. Framing error detection on page 1036](#).

**Figure 338. Break detection in LIN mode (11-bit break length - LBDL bit is set)****Case 1: break signal not long enough => break discarded, LBDF is not set****Case 2: break signal just long enough => break detected, LBDF is set****Case 3: break signal long enough => break detected, LBDF is set**

MSv31156V1

**Figure 339. Break detection in LIN mode vs. Framing error detection****Case 1: break occurring after an Idle****Case 2: break occurring while data is being received**

MSv31157V1

### 31.5.11 USART synchronous mode

The synchronous mode is selected by writing the CLKEN bit in the USART\_CR2 register to 1. In synchronous mode, the following bits must be kept cleared:

- LINEN bit in the USART\_CR2 register,
- SCEN, HDSEL and IREN bits in the USART\_CR3 register.

In this mode, the USART can be used to control bidirectional synchronous serial communications in master mode. The CK pin is the output of the USART transmitter clock. No clock pulses are sent to the CK pin during start bit and stop bit. Depending on the state of the LBCL bit in the USART\_CR2 register, clock pulses are, or are not, generated during the last valid data bit (address mark). The CPOL bit in the USART\_CR2 register is used to select the clock polarity, and the CPHA bit in the USART\_CR2 register is used to select the phase of the external clock (see [Figure 340](#), [Figure 341](#) and [Figure 342](#)).

During the Idle state, preamble and send break, the external CK clock is not activated.

In synchronous mode the USART transmitter works exactly like in asynchronous mode. But as CK is synchronized with TX (according to CPOL and CPHA), the data on TX is synchronous.

In this mode the USART receiver works in a different manner compared to the asynchronous mode. If RE=1, the data is sampled on CK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time must be respected (which depends on the baud rate: 1/16 bit duration).

**Note:** The CK pin works in conjunction with the TX pin. Thus, the clock is provided only if the transmitter is enabled ( $TE=1$ ) and data is being transmitted (the data register USART\_TDR written). This means that it is not possible to receive synchronous data without transmitting data.

The LBCL, CPOL and CPHA bits have to be selected when the USART is disabled ( $UE=0$ ) to ensure that the clock pulses function correctly.

Figure 340. USART example of synchronous transmission

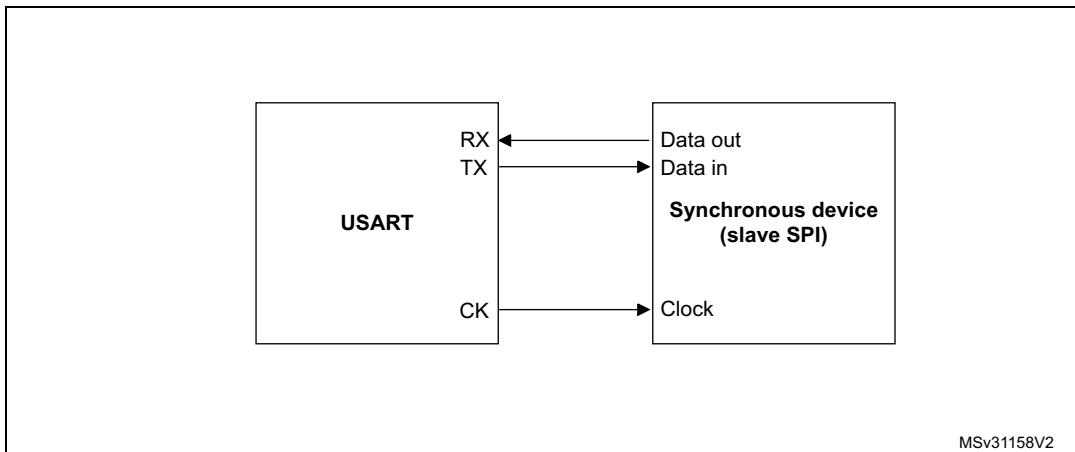


Figure 341. USART data clock timing diagram (M bits = 00)

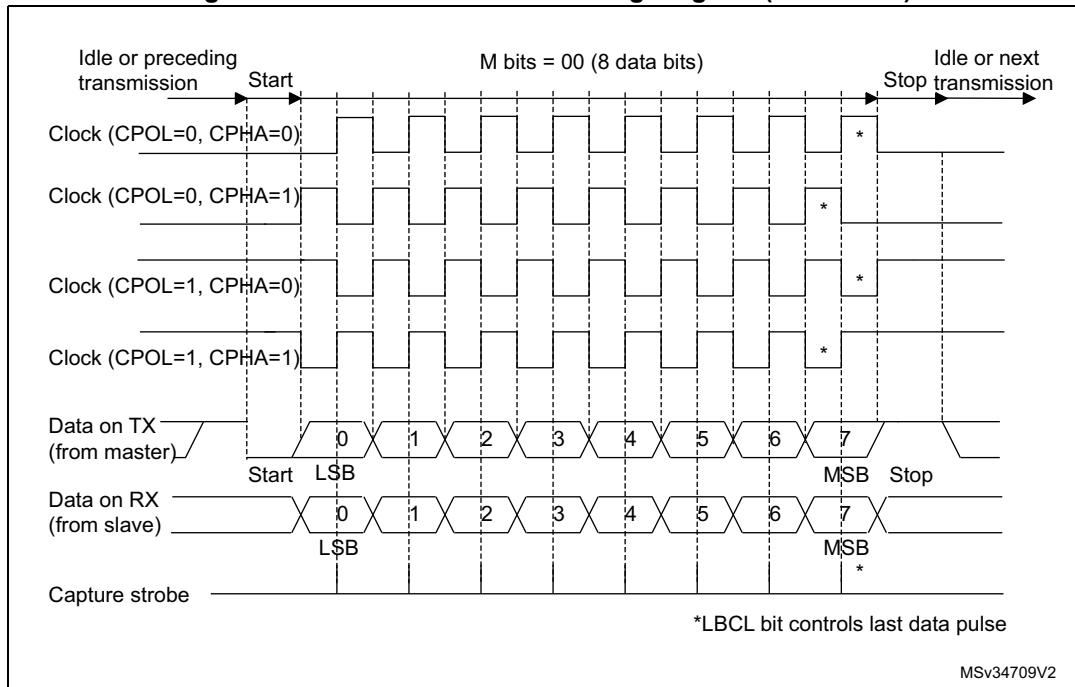


Figure 342. USART data clock timing diagram (M bits = 01)

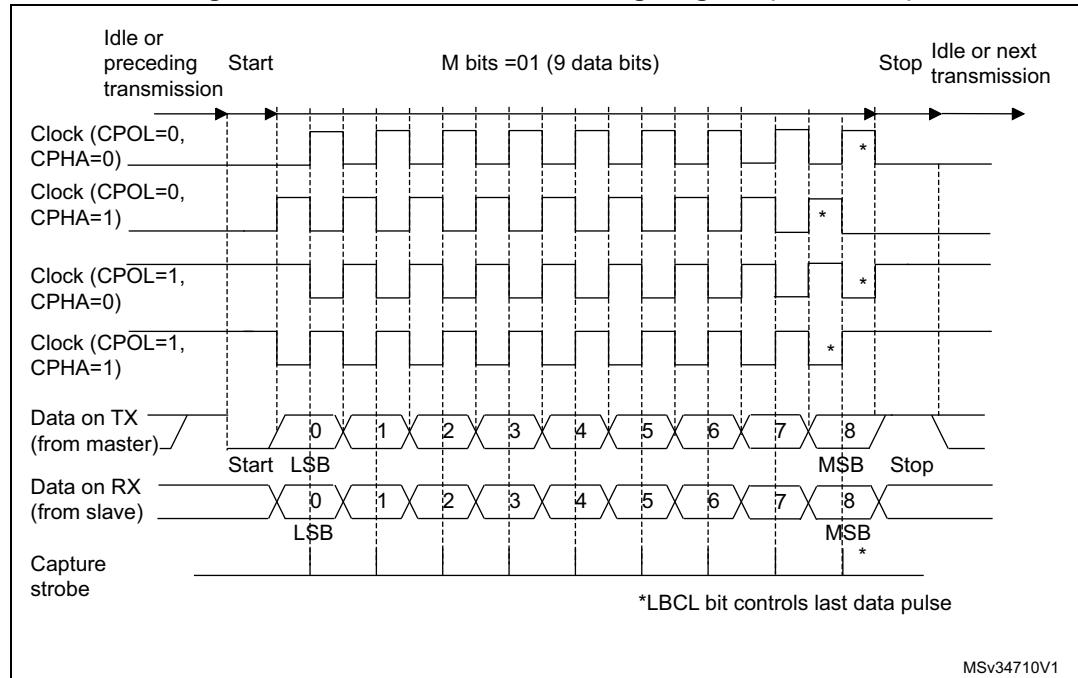
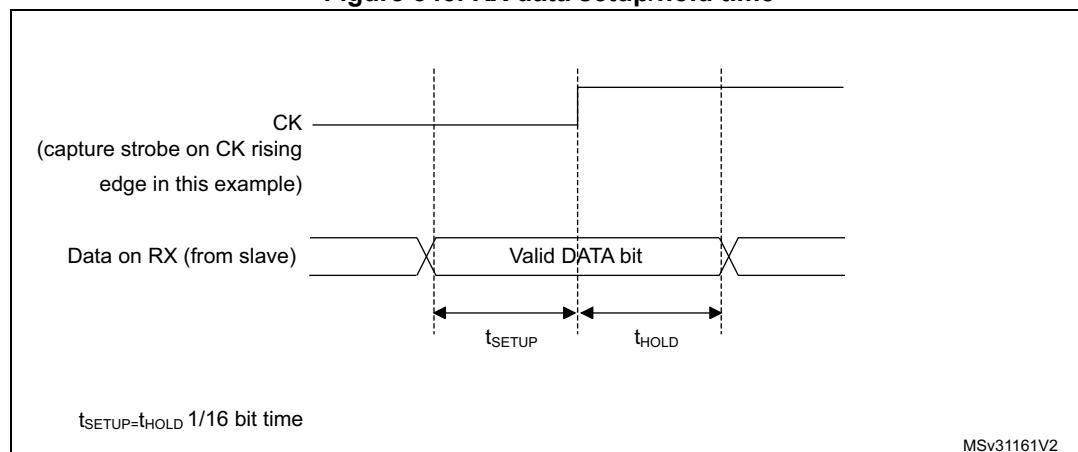


Figure 343. RX data setup/hold time



Note: The function of CK is different in smartcard mode. Refer to [Section 31.5.13: USART smartcard mode](#) for more details.

### 31.5.12 USART single-wire half-duplex communication

Single-wire half-duplex mode is selected by setting the HDSEL bit in the USART\_CR3 register. In this mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART\_CR2 register,
- SCEN and IREN bits in the USART\_CR3 register.

The USART can be configured to follow a single-wire half-duplex protocol where the TX and RX lines are internally connected. The selection between half- and Full-duplex communication is made with a control bit HDSEL in USART\_CR3.

As soon as HDSEL is written to 1:

- The TX and RX lines are internally connected
- The RX pin is no longer used
- The TX pin is always released when no data is transmitted. Thus, it acts as a standard I/O in idle or in reception. It means that the I/O must be configured so that TX is configured as alternate function open-drain with an external pull-up.

Apart from this, the communication protocol is similar to normal USART mode. Any conflicts on the line must be managed by software (by the use of a centralized arbiter, for instance). In particular, the transmission is never blocked by hardware and continues as soon as data is written in the data register while the TE bit is set.

### 31.5.13 USART smartcard mode

This section is relevant only when smartcard mode is supported. Please refer to [Section 31.4: USART implementation on page 1012](#).

Smartcard mode is selected by setting the SCEN bit in the USART\_CR3 register. In smartcard mode, the following bits must be kept cleared:

- LINEN bit in the USART\_CR2 register,
- HDSEL and IREN bits in the USART\_CR3 register.

Moreover, the CLKEN bit may be set in order to provide a clock to the smartcard.

The smartcard interface is designed to support asynchronous protocol for smartcards as defined in the ISO 7816-3 standard. Both T=0 (character mode) and T=1 (block mode) are supported.

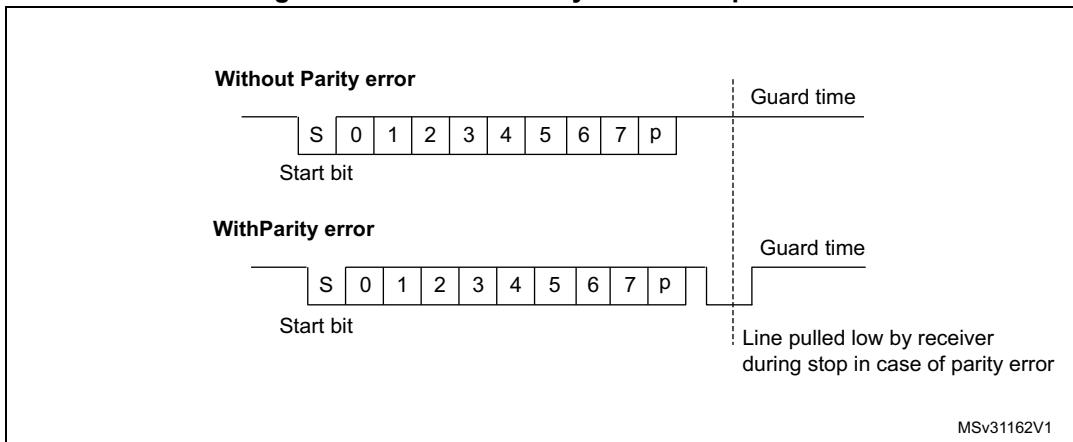
The USART should be configured as:

- 8 bits plus parity: where word length is set to 8 bits and PCE=1 in the USART\_CR1 register
- 1.5 stop bits: where STOP=11 in the USART\_CR2 register. It is also possible to choose 0.5 stop bit for receiving.

In T=0 (character) mode, the parity error is indicated at the end of each character during the guard time period.

[Figure 344](#) shows examples of what can be seen on the data line with and without parity error.

Figure 344. ISO 7816-3 asynchronous protocol



When connected to a smartcard, the TX output of the USART drives a bidirectional line that is also driven by the smartcard. The TX pin must be configured as open drain.

Smartcard mode implements a single wire half duplex communication protocol.

- Transmission of data from the transmit shift register is guaranteed to be delayed by a minimum of 1/2 baud clock. In normal operation a full transmit shift register starts shifting on the next baud clock edge. In smartcard mode this transmission is further delayed by a guaranteed 1/2 baud clock.
- In transmission, if the smartcard detects a parity error, it signals this condition to the USART by driving the line low (NACK). This NACK signal (pulling transmit line low for 1 baud clock) causes a framing error on the transmitter side (configured with 1.5 stop bits). The USART can handle automatic re-sending of data according to the protocol. The number of retries is programmed in the SCARCNT bit field. If the USART continues receiving the NACK after the programmed number of retries, it stops transmitting and signals the error as a framing error. The TXE bit can be set using the TXFRQ bit in the USART\_RQR register.
- Smartcard auto-retry in transmission: a delay of 2.5 baud periods is inserted between the NACK detection by the USART and the start bit of the repeated character. The TC bit is set immediately at the end of reception of the last repeated character (no guard-time). If the software wants to repeat it again, it must insure the minimum 2 baud periods required by the standard.
- If a parity error is detected during reception of a frame programmed with a 1.5 stop bits period, the transmit line is pulled low for a baud clock period after the completion of the receive frame. This is to indicate to the smartcard that the data transmitted to the USART has not been correctly received. A parity error is NACKed by the receiver if the NACK control bit is set, otherwise a NACK is not transmitted (to be used in T=1 mode). If the received character is erroneous, the RXNE/receive DMA request is not activated. According to the protocol specification, the smartcard must resend the same character. If the received character is still erroneous after the maximum number of retries specified in the SCARCNT bit field, the USART stops transmitting the NACK and signals the error as a parity error.
- Smartcard auto-retry in reception: the BUSY flag remains set if the USART NACKs the card but the card doesn't repeat the character.

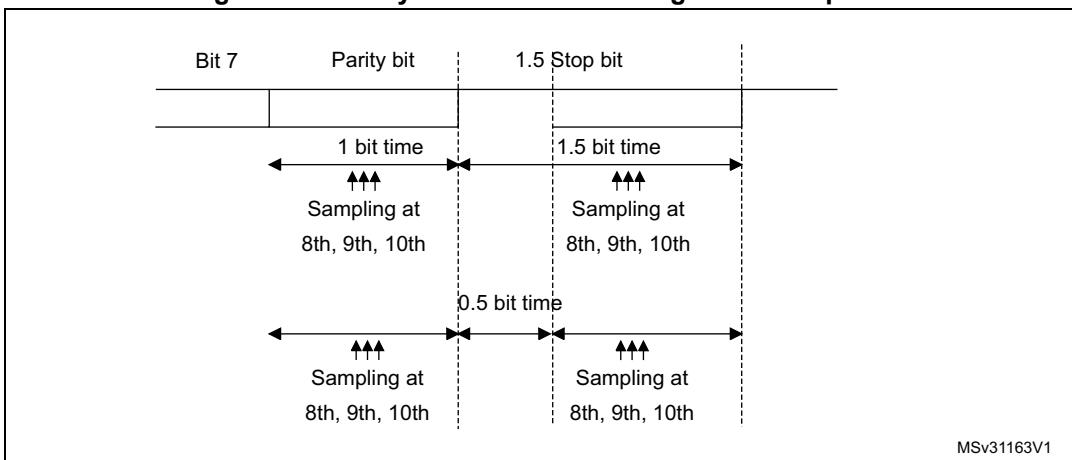
- In transmission, the USART inserts the Guard Time (as programmed in the Guard Time register) between two successive characters. As the Guard Time is measured after the stop bit of the previous character, the GT[7:0] register must be programmed to the desired CGT (Character Guard Time, as defined by the 7816-3 specification) minus 12 (the duration of one character).
- The assertion of the TC flag can be delayed by programming the Guard Time register. In normal operation, TC is asserted when the transmit shift register is empty and no further transmit requests are outstanding. In smartcard mode an empty transmit shift register triggers the Guard Time counter to count up to the programmed value in the Guard Time register. TC is forced low during this time. When the Guard Time counter reaches the programmed value TC is asserted high.
- The de-assertion of TC flag is unaffected by smartcard mode.
- If a framing error is detected on the transmitter end (due to a NACK from the receiver), the NACK is not detected as a start bit by the receive block of the transmitter. According to the ISO protocol, the duration of the received NACK can be 1 or 2 baud clock periods.
- On the receiver side, if a parity error is detected and a NACK is transmitted the receiver does not detect the NACK as a start bit.

*Note:* A break character is not significant in smartcard mode. A 0x00 data with a framing error is treated as data and not as a break.

No Idle frame is transmitted when toggling the TE bit. The Idle frame (as defined for the other configurations) is not defined by the ISO protocol.

*Figure 345* details how the NACK signal is sampled by the USART. In this example the USART is transmitting data and is configured with 1.5 stop bits. The receiver part of the USART is enabled in order to check the integrity of the data and the NACK signal.

**Figure 345. Parity error detection using the 1.5 stop bits**



The USART can provide a clock to the smartcard through the CK output. In smartcard mode, CK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler. The division ratio is configured in the prescaler register USART\_GTPR. CK frequency can be programmed from  $f_{CK}/2$  to  $f_{CK}/62$ , where  $f_{CK}$  is the peripheral input clock.

### Block mode (T=1)

In T=1 (block) mode, the parity error transmission is deactivated, by clearing the NACK bit in the USART\_CR3 register.

When requesting a read from the smartcard, in block mode, the software must enable the receiver Timeout feature by setting the RTOEN bit in the USART\_CR2 register and program the RTO bits field in the RTOR register to the BWT (block wait time) - 11 value. If no answer is received from the card before the expiration of this period, the RTOF flag is set and a timeout interrupt is generated (if RTOIE bit in the USART\_CR1 register is set). If the first character is received before the expiration of the period, it is signaled by the RXNE interrupt.

**Note:** *The RXNE interrupt must be enabled even when using the USART in DMA mode to read from the smartcard in block mode. In parallel, the DMA must be enabled only after the first received byte.*

After the reception of the first character (RXNE interrupt), the RTO bit fields in the RTOR register must be programmed to the CWT (character wait time) - 11 value, in order to allow the automatic check of the maximum wait time between two consecutive characters. This time is expressed in baudtime units. If the smartcard does not send a new character in less than the CWT period after the end of the previous character, the USART signals this to the software through the RTOF flag and interrupt (when RTOIE bit is set).

**Note:** *The RTO counter starts counting:*

- From the end of the stop bit in case STOP = 00.
- From the end of the second stop bit in case of STOP = 10.
- 1 bit duration after the beginning of the STOP bit in case STOP = 11.
- From the beginning of the STOP bit in case STOP = 01.

*As in the smartcard protocol definition, the BWT/CWT values are defined from the beginning (start bit) of the last character. The RTO register must be programmed to BWT -11 or CWT - 11, respectively, taking into account the length of the last character itself.*

A block length counter is used to count all the characters received by the USART. This counter is reset when the USART is transmitting (TXE=0). The length of the block is communicated by the smartcard in the third byte of the block (prologue field). This value must be programmed to the BLEN field in the USART\_RTOR register. When using DMA mode, before the start of the block, this register field must be programmed to the minimum value (0x0). With this value, an interrupt is generated after the 4th received character. The software must read the LEN field (third byte), its value must be read from the receive buffer.

In interrupt driven receive mode, the length of the block may be checked by software or by programming the BLEN value. However, before the start of the block, the maximum value of BLEN (0xFF) may be programmed. The real value is programmed after the reception of the third character.

If the block is using the LRC longitudinal redundancy check (1 epilogue byte), the BLEN=LEN. If the block is using the CRC mechanism (2 epilogue bytes), BLEN=LEN+1 must be programmed. The total block length (including prologue, epilogue and information fields) equals BLEN+4. The end of the block is signaled to the software through the EOBF flag and interrupt (when EOBIIE bit is set).

In case of an error in the block length, the end of the block is signaled by the RTO interrupt (Character wait Time overflow).

**Note:** *The error checking code (LRC/CRC) must be computed/verified by software.*

## Direct and inverse convention

The smartcard protocol defines two conventions: direct and inverse.

The direct convention is defined as: LSB first, logical bit value of 1 corresponds to a H state of the line and parity is even. In order to use this convention, the following control bits must be programmed: MSBFIRST=0, DATAINV=0 (default values).

The inverse convention is defined as: MSB first, logical bit value 1 corresponds to an L state on the signal line and parity is even. In order to use this convention, the following control bits must be programmed: MSBFIRST=1, DATAINV=1.

**Note:** *When logical data values are inverted (0=H, 1=L), the parity bit is also inverted in the same way.*

In order to recognize the card convention, the card sends the initial character, TS, as the first character of the ATR (Answer To Reset) frame. The two possible patterns for the TS are: LHHL LLL LLH and LHHL HHH LLH.

- (H) LHHL LLL LLH sets up the inverse convention: state L encodes value 1 and moment 2 conveys the most significant bit (MSB first). when decoded by inverse convention, the conveyed byte is equal to '3F'.
- (H) LHHL HHH LLH sets up the direct convention: state H encodes value 1 and moment 2 conveys the least significant bit (LSB first). when decoded by direct convention, the conveyed byte is equal to '3B'.

Character parity is correct when there is an even number of bits set to 1 in the nine moments 2 to 10.

As the USART does not know which convention is used by the card, it needs to be able to recognize either pattern and act accordingly. The pattern recognition is not done in hardware, but through a software sequence. Moreover, supposing that the USART is configured in direct convention (default) and the card answers with the inverse convention, TS = LHHL LLL LLH => the USART received character is '03' and the parity is odd.

Therefore, two methods are available for TS pattern recognition:

### Method 1

The USART is programmed in standard smartcard mode/direct convention. In this case, the TS pattern reception generates a parity error interrupt and error signal to the card.

- The parity error interrupt informs the software that the card didn't answer correctly in direct convention. Software then reprograms the USART for inverse convention
- In response to the error signal, the card retries the same TS character, and it is correctly received this time, by the reprogrammed USART

Alternatively, in answer to the parity error interrupt, the software may decide to reprogram the USART and to also generate a new reset command to the card, then wait again for the TS.

### Method 2

The USART is programmed in 9-bit/no-parity mode, no bit inversion. In this mode it receives any of the two TS patterns as:

- (H) LHHL LLL LLH = 0x103 -> inverse convention to be chosen
- (H) LHHL HHH LLH = 0x13B -> direct convention to be chosen

The software checks the received character against these two patterns and, if any of them match, then programs the USART accordingly for the next character reception.

If none of the two is recognized, a card reset may be generated in order to restart the negotiation.

### 31.5.14 USART IrDA SIR ENDEC block

This section is relevant only when IrDA mode is supported. Please refer to [Section 31.4: USART implementation on page 1012](#).

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 346](#)).

The SIR Transmit encoder modulates the Non Return to Zero (NRZ) transmit bit stream output from USART. The output pulse stream is transmitted to an external output driver and infrared LED. USART supports only bit rates up to 115.2 Kbps for the SIR ENDEC. In normal mode the transmitted pulse width is specified as 3/16 of a bit period.

The SIR receive decoder demodulates the return-to-zero bit stream from the infrared detector and outputs the received NRZ serial bit stream to the USART. The decoder input is normally high (marking state) in the Idle state. The transmit encoder output has the opposite polarity to the decoder input. A start bit is detected when the decoder input is low.

- IrDA is a half duplex communication protocol. If the Transmitter is busy (when the USART is sending data to the IrDA encoder), any data on the IrDA receive line is ignored by the IrDA decoder and if the Receiver is busy (when the USART is receiving decoded data from the IrDA decoder), data on the TX from the USART to IrDA is not encoded. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.
- A 0 is transmitted as a high pulse and a 1 is transmitted as a 0. The width of the pulse is specified as 3/16th of the selected bit period in normal mode (see [Figure 347](#)).
- The SIR decoder converts the IrDA compliant receive signal into a bit stream for USART.
- The SIR receive logic interprets a high state as a logic one and low pulses as logic zeros.
- The transmit encoder output has the opposite polarity to the decoder input. The SIR output is in low state when Idle.
- The IrDA specification requires the acceptance of pulses greater than 1.41  $\mu$ s. The acceptable pulse width is programmable. Glitch detection logic on the receiver end filters out pulses of width less than 2 PSC periods (PSC is the prescaler value programmed in the USART\_GTPR). Pulses of width less than 1 PSC period are always rejected, but those of width greater than one and less than two periods may be accepted or rejected, those greater than 2 periods are accepted as a pulse. The IrDA encoder/decoder doesn't work when PSC=0.
- The receiver can communicate with a low-power transmitter.
- In IrDA mode, the STOP bits in the USART\_CR2 register must be configured to "1 stop bit".

## IrDA low-power mode

### Transmitter

In low-power mode the pulse width is not maintained at 3/16 of the bit period. Instead, the width of the pulse is 3 times the low-power baud rate which can be a minimum of 1.42 MHz.

Generally, this value is 1.8432 MHz (1.42 MHz < PSC < 2.12 MHz). A low-power mode programmable divisor divides the system clock to achieve this value.

### Receiver

Receiving in low-power mode is similar to receiving in normal mode. For glitch detection the USART should discard pulses of duration shorter than 1 PSC period. A valid low is accepted only if its duration is greater than 2 periods of the IrDA low-power Baud clock (PSC value in the USART\_GTPR).

**Note:** *A pulse of width less than two and greater than one PSC period(s) may or may not be rejected.*

*The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).*

Figure 346. IrDA SIR ENDEC- block diagram

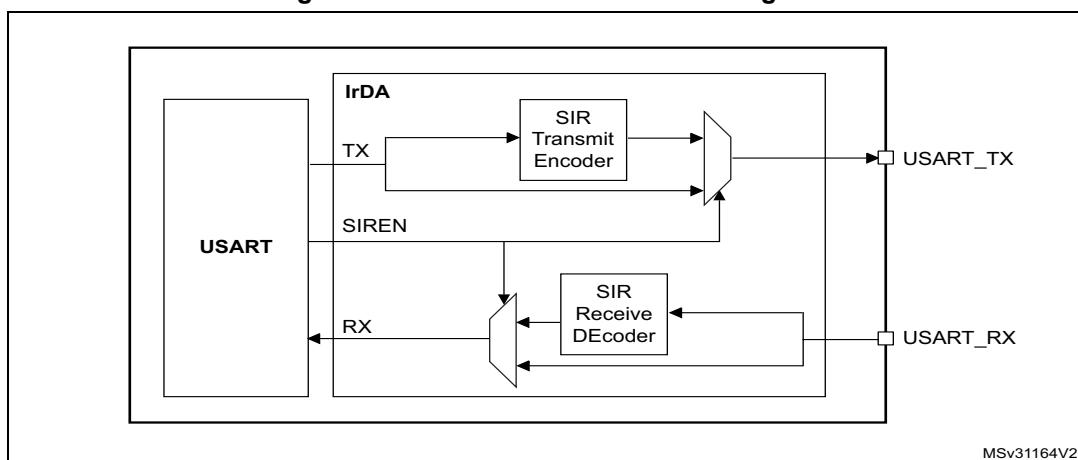
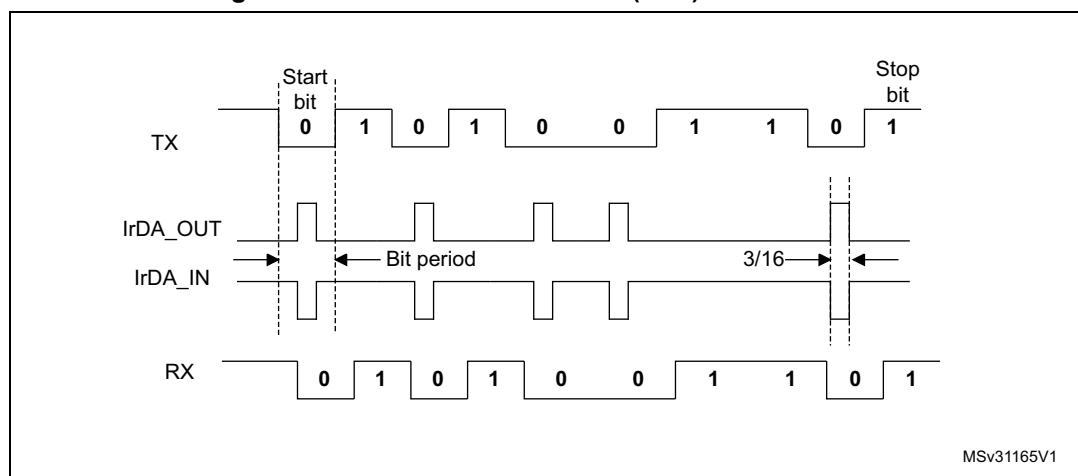


Figure 347. IrDA data modulation (3/16) - normal mode



### 31.5.15 USART continuous communication in DMA mode

The USART is capable of performing continuous communication using the DMA. The DMA requests for Rx buffer and Tx buffer are generated independently.

**Note:** Please refer to [Section 31.4: USART implementation on page 1012](#) to determine if the DMA mode is supported. If DMA is not supported, use the USART as explained in [Section 31.5.2: USART transmitter](#) or [Section 31.5.3: USART receiver](#). To perform continuous communication, the user can clear the TXE/ RXNE flags in the USART\_ISR register.

#### Transmission using DMA

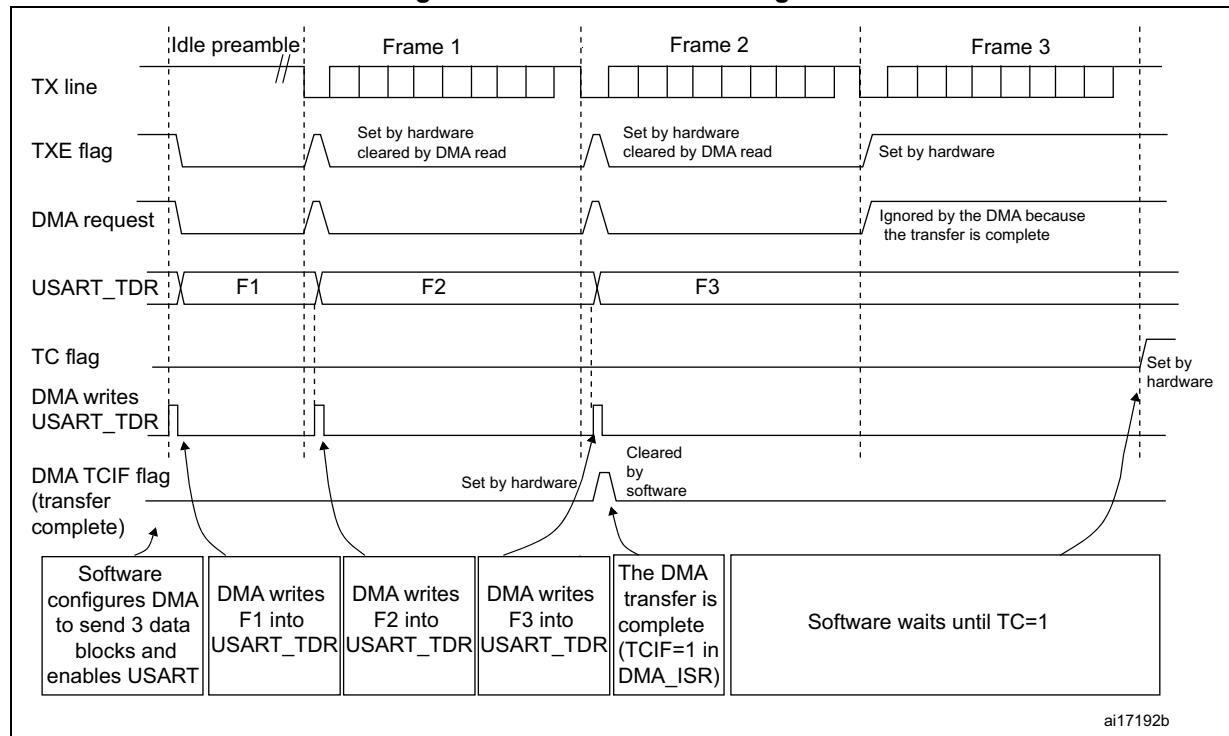
DMA mode can be enabled for transmission by setting DMAT bit in the USART\_CR3 register. Data is loaded from a SRAM area configured using the DMA peripheral (refer to [Section 8: Direct memory access controller \(DMA\)](#)) to the USART\_TDR register whenever the TXE bit is set. To map a DMA channel for USART transmission, use the following procedure (x denotes the channel number):

1. Write the USART\_TDR register address in the DMA control register to configure it as the destination of the transfer. The data is moved to this address from memory after each TXE event.
2. Write the memory address in the DMA control register to configure it as the source of the transfer. The data is loaded into the USART\_TDR register from this memory area after each TXE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA register
5. Configure DMA interrupt generation after half/ full transfer as required by the application.
6. Clear the TC flag in the USART\_ISR register by setting the TCCF bit in the USART\_ICR register.
7. Activate the channel in the DMA register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector.

In transmission mode, once the DMA has written all the data to be transmitted (the TCIF flag is set in the DMA\_ISR register), the TC flag can be monitored to make sure that the USART communication is complete. This is required to avoid corrupting the last transmission before disabling the USART or entering Stop mode. Software must wait until TC=1. The TC flag remains cleared during all data transfers and it is set by hardware at the end of transmission of the last frame.

Figure 348. Transmission using DMA



ai17192b

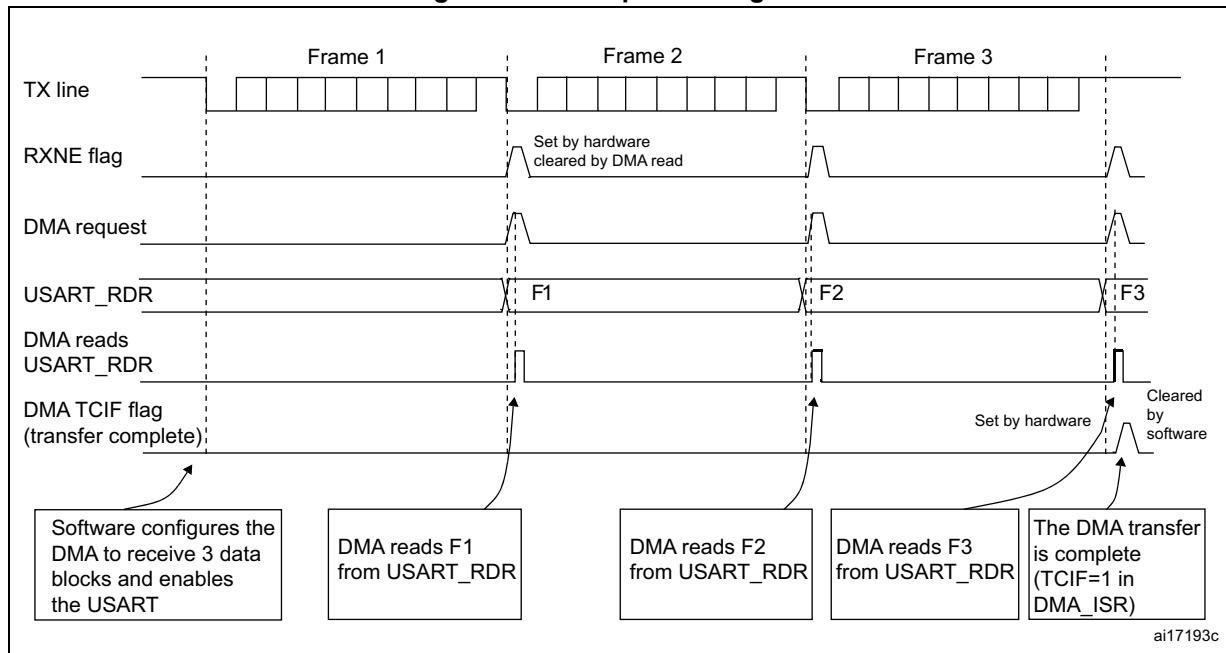
### Reception using DMA

DMA mode can be enabled for reception by setting the DMAR bit in USART\_CR3 register. Data is loaded from the USART\_RDR register to a SRAM area configured using the DMA peripheral (refer to [Section 8: Direct memory access controller \(DMA\)](#)) whenever a data byte is received. To map a DMA channel for USART reception, use the following procedure:

1. Write the USART\_RDR register address in the DMA control register to configure it as the source of the transfer. The data is moved from this address to the memory after each RXNE event.
2. Write the memory address in the DMA control register to configure it as the destination of the transfer. The data is loaded from USART\_RDR to this memory area after each RXNE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA control register
5. Configure interrupt generation after half/ full transfer as required by the application.
6. Activate the channel in the DMA control register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector.

Figure 349. Reception using DMA



ai17193c

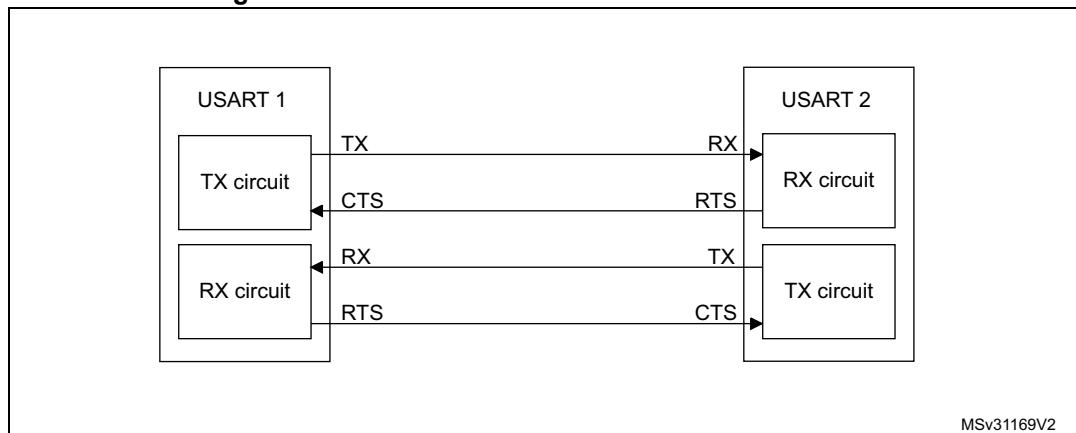
### Error flagging and interrupt generation in multibuffer communication

In multibuffer communication if any error occurs during the transaction the error flag is asserted after the current byte. An interrupt is generated if the interrupt enable flag is set. For framing error, overrun error and noise flag which are asserted with RXNE in single byte reception, there is a separate error flag interrupt enable bit (EIE bit in the USART\_CR3 register), which, if set, enables an interrupt after the current byte if any of these errors occur.

### 31.5.16 RS232 hardware flow control and RS485 driver enable using USART

It is possible to control the serial data flow between 2 devices by using the CTS input and the RTS output. The [Figure 350](#) shows how to connect 2 devices in this mode:

Figure 350. Hardware flow control between 2 USARTs



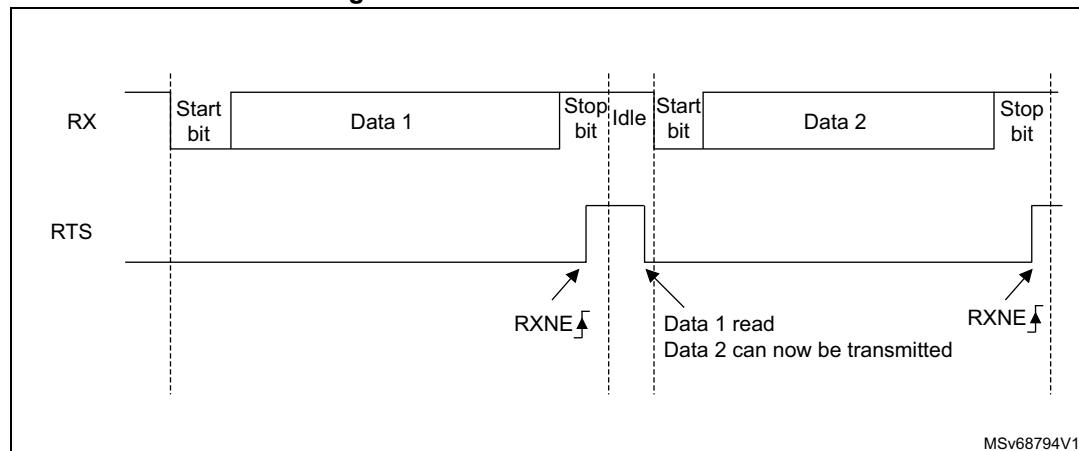
MSv31169V2

RS232 RTS and CTS flow control can be enabled independently by writing the RTSE and CTSE bits respectively to 1 (in the USART\_CR3 register).

### RS232 RTS flow control

If the RTS flow control is enabled (RTSE=1), then RTS is deasserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register is full, RTS is asserted, indicating that the transmission is expected to stop at the end of the current frame. [Figure 351](#) shows an example of communication with RTS flow control enabled.

**Figure 351. RS232 RTS flow control**

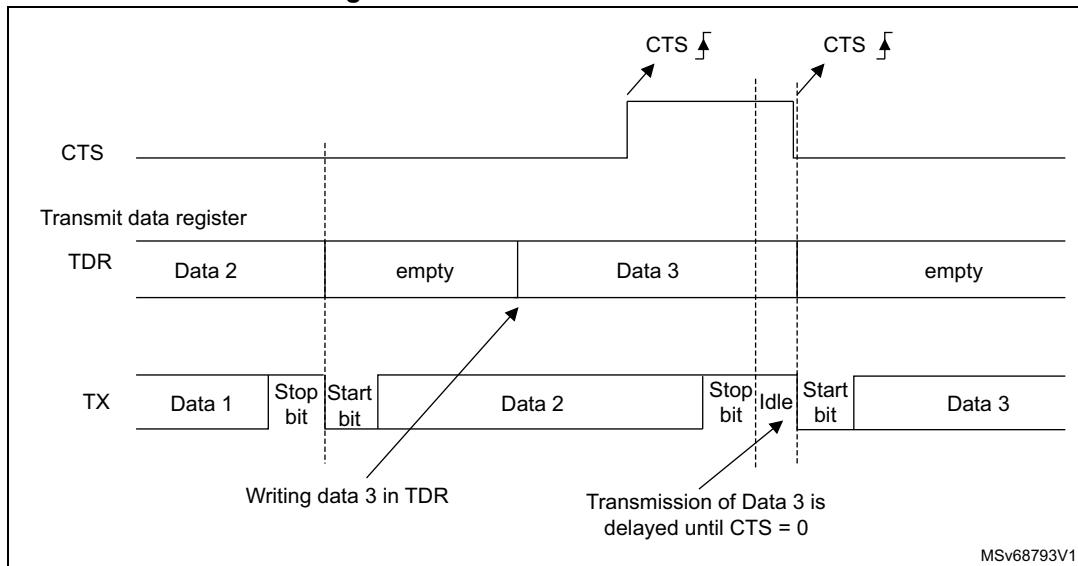


### RS232 CTS flow control

If the CTS flow control is enabled (CTSE=1), then the transmitter checks the CTS input before transmitting the next frame. If CTS is deasserted (tied low), then the next data is transmitted (assuming that data is to be transmitted, in other words, if TXE=0), else the transmission does not occur. When CTS is asserted during a transmission, the current transmission is completed before the transmitter stops.

When CTSE=1, the CTSIF status bit is automatically set by hardware as soon as the CTS input toggles. It indicates when the receiver becomes ready or not ready for communication. An interrupt is generated if the CTSIE bit in the USART\_CR3 register is set. [Figure 352](#) shows an example of communication with CTS flow control enabled.

Figure 352. RS232 CTS flow control



**Note:** For correct behavior, CTS must be deasserted at least 3 USART clock source periods before the end of the current character. In addition it should be noted that the CTSCF flag may not be set for pulses shorter than 2 x PCLK periods.

### RS485 Driver Enable

The driver enable feature is enabled by setting bit DEM in the USART\_CR3 control register. This allows the user to activate the external transceiver control, through the DE (Driver Enable) signal. The assertion time is the time between the activation of the DE signal and the beginning of the START bit. It is programmed using the DEAT [4:0] bit fields in the USART\_CR1 control register. The de-assertion time is the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE signal. It is programmed using the DEDT [4:0] bit fields in the USART\_CR1 control register. The polarity of the DE signal can be configured using the DEP bit in the USART\_CR3 control register.

In USART, the DEAT and DEDT are expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

## 31.6 USART in low-power modes

Table 204. Effect of low-power modes on the USART

Mode	Description
Sleep	No effect. USART interrupt causes the device to exit Sleep mode.
Stop	No effect.
Standby	The USART is powered down and must be reinitialized when the device has exited from Standby mode.

## 31.7 USART interrupts

Table 205. USART interrupt requests

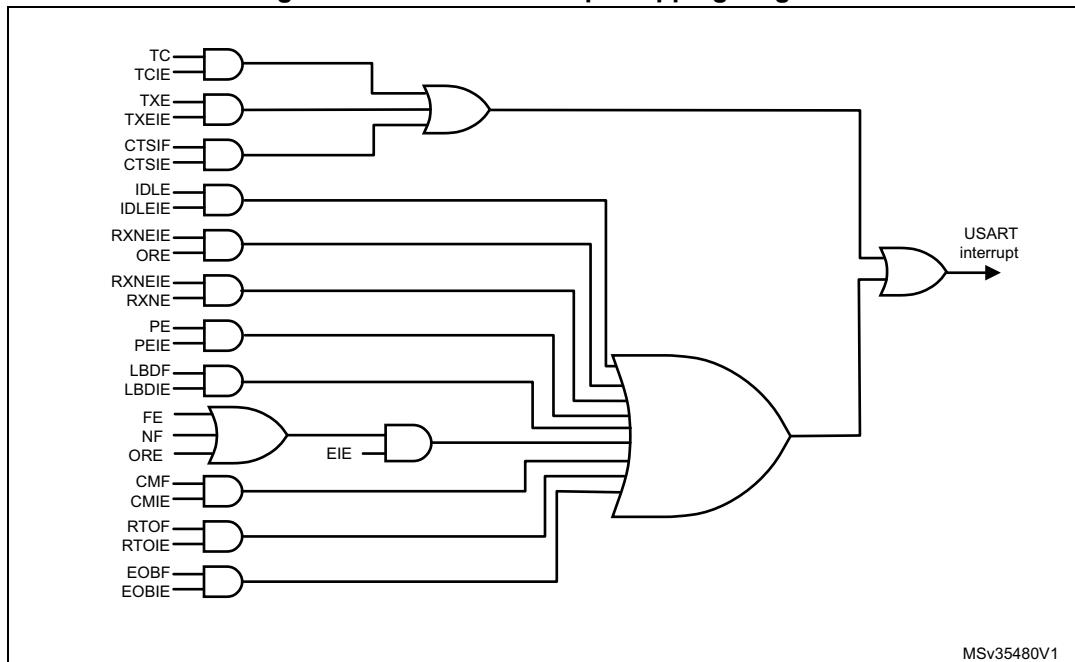
Interrupt event	Event flag	Enable Control bit
Transmit data register empty	TXE	TXEIE
CTS interrupt	CTSIF	CTSIE
Transmission Complete	TC	TCIE
Receive data register not empty (data ready to be read)	RXNE	RXNEIE
Overrun error detected	ORE	
Idle line detected	IDLE	IDLEIE
Parity error	PE	PEIE
LIN break	LBDF	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffer communication.	NF or ORE or FE	EIE
Character match	CMF	CMIE
Receiver timeout	RTOF	RTOIE
End of Block	EOBF	EOBIE

The USART interrupt events are connected to the same interrupt vector (see [Figure 353](#)).

- During transmission: Transmission Complete, Clear to Send, Transmit data Register empty or Framing error (in smartcard mode) interrupt.
- During reception: Idle Line detection, Overrun error, Receive data register not empty, Parity error, LIN break detection, Noise Flag, Framing Error, Character match, etc.

These events generate an interrupt if the corresponding Enable Control Bit is set.

Figure 353. USART interrupt mapping diagram



## 31.8 USART registers

Refer to [Section 1.2 on page 63](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32 bits).

### 31.8.1 USART control register 1 (USART\_CR1)

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.	M1	EOBIE	RTOIE	DEAT[4:0]					DEDT[4:0]				
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	Res.	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bits 31:29 Reserved, must be kept at reset value.

#### Bit 28 M1: Word length

This bit, with bit 12 (M0), determines the word length. It is set or cleared by software.

M[1:0] = 00: 1 Start bit, 8 data bits, n stop bits

M[1:0] = 01: 1 Start bit, 9 data bits, n stop bits

M[1:0] = 10: 1 Start bit, 7 data bits, n stop bits

This bit can only be written when the USART is disabled (UE=0).

*Note:* Not all modes are supported In 7-bit data length mode. Refer to [Section 31.4: USART implementation](#) for details.

#### Bit 27 EOBIE: End of Block interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated when the EOBF flag is set in the USART\_ISR register.

*Note:* If the USART does not support smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).

#### Bit 26 RTOIE: Receiver timeout interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated when the RTOF bit is set in the USART\_ISR register.

*Note:* If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. [Section 31.4: USART implementation on page 1012](#).

#### Bits 25:21 DEAT[4:0]: Driver Enable assertion time

This 5-bit value defines the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

This bit field can only be written when the USART is disabled (UE=0).

*Note:* If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).

**Bits 20:16 DEDT[4:0]: Driver Enable de-assertion time**

This 5-bit value defines the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

If the USART\_TDR register is written during the DEDT time, the new data is transmitted only when the DEDT and DEAT times have both elapsed.

This bit field can only be written when the USART is disabled (UE=0).

*Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

**Bit 15 OVER8: Oversampling mode**

- 0: Oversampling by 16
- 1: Oversampling by 8

This bit can only be written when the USART is disabled (UE=0).

*Note: In LIN, IrDA and modes, this bit must be kept at reset value.*

**Bit 14 CMIE: Character match interrupt enable**

This bit is set and cleared by software.

- 0: Interrupt is inhibited
- 1: A USART interrupt is generated when the CMF bit is set in the USART\_ISR register.

**Bit 13 MME: Mute mode enable**

This bit activates the mute mode function of the USART. When set, the USART can switch between the active and mute modes, as defined by the WAKE bit. It is set and cleared by software.

- 0: Receiver in active mode permanently
- 1: Receiver can switch between mute mode and active mode.

**Bit 12 M0: Word length**

This bit, with bit 28 (M1), determines the word length. It is set or cleared by software. See Bit 28 (M1) description.  
This bit can only be written when the USART is disabled (UE=0).

**Bit 11 WAKE: Receiver wake-up method**

This bit determines the USART wake-up method from mute mode. It is set or cleared by software.

- 0: Idle line
- 1: Address mark

This bit field can only be written when the USART is disabled (UE=0).

**Bit 10 PCE: Parity control enable**

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

- 0: Parity control disabled
- 1: Parity control enabled

This bit field can only be written when the USART is disabled (UE=0).

**Bit 9 PS: Parity selection**

This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity is selected after the current byte.

- 0: Even parity
- 1: Odd parity

This bit field can only be written when the USART is disabled (UE=0).

Bit 8 **PEIE**: PE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever PE=1 in the USART\_ISR register

Bit 7 **TXEIE**: interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever TXE=1 in the USART\_ISR register

Bit 6 **TCIE**: Transmission complete interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever TC=1 in the USART\_ISR register

Bit 5 **RXNEIE**: RXNE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART\_ISR register

Bit 4 **IDLEIE**: IDLE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever IDLE=1 in the USART\_ISR register

Bit 3 **TE**: Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

*Note: During transmission, a “0” pulse on the TE bit (“0” followed by “1”) sends a preamble (idle line) after the current word, except in smartcard mode. In order to generate an idle character, the TE must not be immediately written to 1. In order to ensure the required duration, the software can poll the TEACK bit in the USART\_ISR register.*

*In smartcard mode, when TE is set there is a 1 bit-time delay before the transmission starts.*

Bit 2 **RE**: Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

## Bit 1 Reserved, must be kept at reset value.

Bit 0 **UE**: USART enable

When this bit is cleared, the USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USART\_ISR are set to their default values. This bit is set and cleared by software.

0: USART prescaler and outputs disabled, low-power mode

1: USART enabled

*Note: In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the USART\_ISR to be set before resetting the UE bit.*

*The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.*

### 31.8.2 USART control register 2 (USART\_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADD[7:4]				ADD[3:0]				RTOEN	ABRMOD[1:0]		ABREN	MSBFIRST	DATAINV	TXINV	RXINV
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWAP	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	ADDM7	Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw				

Bits 31:28 **ADD[7:4]: Address of the USART node**

This bit-field gives the address of the USART node or a character code to be recognized.

This is used in multiprocessor communication during mute mode, for wake-up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match.

This bit field can only be written when reception is disabled (RE = 0) or the USART is disabled (UE=0)

Bits 27:24 **ADD[3:0]: Address of the USART node**

This bit-field gives the address of the USART node or a character code to be recognized.

This is used in multiprocessor communication during mute mode, for wake-up with address mark detection.

This bit field can only be written when reception is disabled (RE = 0) or the USART is disabled (UE=0)

Bit 23 **RTOEN: Receiver timeout enable**

This bit is set and cleared by software.

0: Receiver timeout feature disabled.

1: Receiver timeout feature enabled.

When this feature is enabled, the RTOF flag in the USART\_ISR register is set if the RX line is idle (no reception) for the duration programmed in the RTOR (receiver timeout register).

*Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. Please refer to Section 31.4: USART implementation on page 1012.*

Bits 22:21 **ABRMODE[1:0]: Auto baud rate mode**

These bits are set and cleared by software.

00: Measurement of the start bit is used to detect the baud rate.

01: Falling edge to falling edge measurement. (the received frame must start with a single bit = 1 -> Frame = Start10xxxxxx)

10: 0x7F frame detection.

11: 0x55 frame detection

This bit field can only be written when ABREN = 0 or the USART is disabled (UE=0).

*Note: If DATAINV=1 and/or MSBFIRST=1 the patterns must be the same on the line, for example 0xAA for MSBFIRST)*

*If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to Section 31.4: USART implementation on page 1012.*

**Bit 20 ABREN:** Auto baud rate enable

This bit is set and cleared by software.

0: Auto baud rate detection is disabled.

1: Auto baud rate detection is enabled.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

**Bit 19 MSBFIRST:** Most significant bit first

This bit is set and cleared by software.

0: data is transmitted/received with data bit 0 first, following the start bit.

1: data is transmitted/received with the MSB (bit 7/8/9) first, following the start bit.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 18 DATAINV:** Binary data inversion

This bit is set and cleared by software.

0: Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

1: Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 17 TXINV:** TX pin active level inversion

This bit is set and cleared by software.

0: TX pin signal works using the standard logic levels ( $V_{DD} = 1/\text{idle}$ , Gnd=0/mark)

1: TX pin signal values are inverted. ( $V_{DD} = 0/\text{mark}$ , Gnd=1/idle).

This allows the use of an external inverter on the TX line.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 16 RXINV:** RX pin active level inversion

This bit is set and cleared by software.

0: RX pin signal works using the standard logic levels ( $V_{DD} = 1/\text{idle}$ , Gnd=0/mark)

1: RX pin signal values are inverted. ( $V_{DD} = 0/\text{mark}$ , Gnd=1/idle).

This allows the use of an external inverter on the RX line.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 15 SWAP:** Swap TX/RX pins

This bit is set and cleared by software.

0: TX/RX pins are used as defined in standard pinout

1: The TX and RX pins functions are swapped. This allows to work in the case of a cross-wired connection to another USART.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 14 LINEN:** LIN mode enable

This bit is set and cleared by software.

0: LIN mode disabled

1: LIN mode enabled

The LIN mode enables the capability to send LIN synchronous breaks (13 low bits) using the SBKRQ bit in the USART\_RQR register, and to detect LIN Sync breaks.

This bit field can only be written when the USART is disabled (UE=0).

*Note: If the USART does not support LIN mode, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

**Bits 13:12 STOP[1:0]: STOP bits**

These bits are used for programming the stop bits.

- 00: 1 stop bit
- 01: 0.5 stop bit
- 10: 2 stop bits
- 11: 1.5 stop bits

This bit field can only be written when the USART is disabled (UE=0).

**Bit 11 CLKEN: Clock enable**

This bit allows the user to enable the CK pin.

- 0: CK pin disabled
- 1: CK pin enabled

This bit can only be written when the USART is disabled (UE=0).

*Note: If neither synchronous mode nor smartcard mode is supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

*In order to provide correctly the CK clock to the smartcard when CK is always available When CLKEN = 1, regardless of the UE bit value, the steps below must be respected:*

- UE = 0
- SCEN = 1
- GTPR configuration (If PSC needs to be configured, it is recommended to configure PSC and GT in a single access to USART\_GTPR register).
- CLKEN= 1
- UE = 1

**Bit 10 CPOL: Clock polarity**

This bit allows the user to select the polarity of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPHA bit to produce the desired clock/data relationship

- 0: Steady low value on CK pin outside transmission window
- 1: Steady high value on CK pin outside transmission window

This bit can only be written when the USART is disabled (UE=0).

*Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

**Bit 9 CPHA: Clock phase**

This bit is used to select the phase of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPOL bit to produce the desired clock/data relationship (see [Figure 341](#) and [Figure 342](#))

- 0: The first clock transition is the first data capture edge
- 1: The second clock transition is the first data capture edge

This bit can only be written when the USART is disabled (UE=0).

*Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

**Bit 8 LBCL: Last bit clock pulse**

This bit is used to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the CK pin in synchronous mode.

- 0: The clock pulse of the last data bit is not output to the CK pin
- 1: The clock pulse of the last data bit is output to the CK pin

**Caution:** The last bit is the 7th or 8th or 9th data bit transmitted depending on the 7 or 8 or 9 bit format selected by the M bits in the USART\_CR1 register.

This bit can only be written when the USART is disabled (UE=0).

*Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

**Bit 7 Reserved, must be kept at reset value.**

Bit 6 **LBDIE**: LIN break detection interrupt enable

Break interrupt mask (break detection using break delimiter).

0: Interrupt is inhibited

1: An interrupt is generated whenever LBDF=1 in the USART\_ISR register

*Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 31.4: USART implementation on page 1012.*

Bit 5 **LBDL**: LIN break detection length

This bit is for selection between 11 bit or 10 bit break detection.

0: 10-bit break detection

1: 11-bit break detection

This bit can only be written when the USART is disabled (UE=0).

*Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 31.4: USART implementation on page 1012.*

Bit 4 **ADDM7**: 7-bit Address Detection/4-bit Address Detection

This bit is for selection between 4-bit address detection or 7-bit address detection.

0: 4-bit address detection

1: 7-bit address detection (in 8-bit data mode)

This bit can only be written when the USART is disabled (UE=0)

*Note: In 7-bit and 9-bit data modes, the address detection is done on 6-bit and 8-bit address (ADD[5:0] and ADD[7:0]) respectively.*

Bits 3:0 Reserved, must be kept at reset value.

*Note:* The 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.

### 31.8.3 USART control register 3 (USART\_CR3)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SCARC NT2	SCARC NT1	SCARC NT0	Res.
												rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DEP	DEM	DDRE	OVRDIS	ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:25 Reserved, must be kept at reset value.

Bit 24 Reserved, must be kept at reset value.

Bit 23 Reserved, must be kept at reset value.

Bits 22:20 Reserved, must be kept at reset value.

Bits 19:17 **SCARCNT[2:0]**: Smartcard auto-retry count

This bit-field specifies the number of retries in transmit and receive, in smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number of erroneous reception trials, before generating a reception error (RXNE and PE bits set). This bit field must be programmed only when the USART is disabled (UE=0). When the USART is enabled (UE=1), this bit field may only be written to 0x0, in order to stop retransmission.

0x0: retransmission disabled - No automatic retransmission in transmit mode.  
0x1 to 0x7: number of automatic retransmission attempts (before signaling error)

*Note: If smartcard mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 16 Reserved, must be kept at reset value.

Bit 15 **DEP**: Driver enable polarity selection

0: DE signal is active high.  
1: DE signal is active low.

This bit can only be written when the USART is disabled (UE=0).

*Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 14 **DEM**: Driver enable mode

This bit allows the user to activate the external transceiver control, through the DE signal.  
0: DE function is disabled.  
1: DE function is enabled. The DE signal is output on the RTS pin.  
This bit can only be written when the USART is disabled (UE=0).

*Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. [Section 31.4: USART implementation on page 1012](#).*

Bit 13 **DDRE**: DMA Disable on Reception Error

0: DMA is not disabled in case of reception error. The corresponding error flag is set but RXNE is kept 0 preventing from overrun. As a consequence, the DMA request is not asserted, so the erroneous data is not transferred (no DMA request), but next correct received data are transferred (used for smartcard mode).  
1: DMA is disabled following a reception error. The corresponding error flag is set, as well as RXNE. The DMA request is masked until the error flag is cleared. This means that the software must first disable the DMA request (DMAR = 0) or clear RXNE before clearing the error flag.  
This bit can only be written when the USART is disabled (UE=0).

*Note: The reception errors are: parity error, framing error or noise error.*

Bit 12 **OVRDIS**: Overrun Disable

This bit is used to disable the receive overrun detection.  
0: Overrun Error Flag, ORE, is set when received data is not read before receiving new data.  
1: Overrun functionality is disabled. If new data is received while the RXNE flag is still set the ORE flag is not set and the new received data overwrites the previous content of the USART\_RDR register.  
This bit can only be written when the USART is disabled (UE=0).

*Note: This control bit allows checking the communication flow without reading the data.*

Bit 11 **ONEBIT**: One sample bit method enable

This bit allows the user to select the sample method. When the one sample bit method is selected the noise detection flag (NF) is disabled.

- 0: Three sample bit method
- 1: One sample bit method

This bit can only be written when the USART is disabled (UE=0).

*Note: ONEBIT feature applies only to data bits, It does not apply to Start bit.*

Bit 10 **CTSIE**: CTS interrupt enable

- 0: Interrupt is inhibited
- 1: An interrupt is generated whenever CTSIF=1 in the USART\_ISR register

*Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 9 **CTSE**: CTS enable

- 0: CTS hardware flow control disabled
- 1: CTS mode enabled, data is only transmitted when the CTS input is deasserted (tied to 0). If the CTS input is asserted while data is being transmitted, then the transmission is completed before stopping. If data is written into the data register while CTS is asserted, the transmission is postponed until CTS is deasserted.

This bit can only be written when the USART is disabled (UE=0)

*Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 8 **RTSE**: RTS enable

- 0: RTS hardware flow control disabled
- 1: RTS output enabled, data is only requested when there is space in the receive buffer. The transmission of data is expected to cease after the current character has been transmitted. The RTS output is deasserted (pulled to 0) when data can be received.

This bit can only be written when the USART is disabled (UE=0).

*Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 7 **DMAT**: DMA enable transmitter

- This bit is set/reset by software
- 1: DMA mode is enabled for transmission
- 0: DMA mode is disabled for transmission

Bit 6 **DMAR**: DMA enable receiver

- This bit is set/reset by software
- 1: DMA mode is enabled for reception
- 0: DMA mode is disabled for reception

Bit 5 **SCEN**: Smartcard mode enable

This bit is used for enabling smartcard mode.

- 0: Smartcard mode disabled
- 1: Smartcard mode enabled

This bit field can only be written when the USART is disabled (UE=0).

*Note: If the USART does not support smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 4 **NACK**: Smartcard NACK enable

0: NACK transmission in case of parity error is disabled

1: NACK transmission during parity error is enabled

This bit field can only be written when the USART is disabled (UE=0).

*Note: If the USART does not support smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 3 **HDSEL**: Half-duplex selection

Selection of single-wire half-duplex mode

0: Half-duplex mode is not selected

1: Half-duplex mode is selected

This bit can only be written when the USART is disabled (UE=0).

Bit 2 **IRLP**: IrDA low-power

This bit is used for selecting between normal and low-power IrDA modes

0: Normal mode

1: Low-power mode

This bit can only be written when the USART is disabled (UE=0).

*Note: If IrDA mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 1 **IREN**: IrDA mode enable

This bit is set and cleared by software.

0: IrDA disabled

1: IrDA enabled

This bit can only be written when the USART is disabled (UE=0).

*Note: If IrDA mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 0 **EIE**: Error interrupt enable

Error Interrupt Enable Bit is required to enable interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USART\_ISR register).

0: Interrupt is inhibited

1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USART\_ISR register.

### 31.8.4 USART baud rate register (USART\_BRR)

This register can only be written when the USART is disabled (UE=0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
BRR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:4 **BRR[15:4]**: USART baud rate

$\text{BRR}[15:4] = \text{USARTDIV}[15:4]$

Bits 3:0 **BRR[3:0]**: USART baud rate

When  $\text{OVER8} = 0$ ,  $\text{BRR}[3:0] = \text{USARTDIV}[3:0]$ .

When  $\text{OVER8} = 1$ :

$\text{BRR}[2:0] = \text{USARTDIV}[3:0]$  shifted 1 bit to the right.

$\text{BRR}[3]$  must be kept cleared.

### 31.8.5 USART guard time and prescaler register (USART\_GTPR)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GT[7:0]								PSC[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:8 **GT[7:0]**: Guard time value

This bit-field is used to program the Guard time value in terms of number of baud clock periods.

This is used in smartcard mode. The Transmission Complete flag is set after this guard time value.

This bit field can only be written when the USART is disabled (UE=0).

*Note: If smartcard mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bits 7:0 **PSC[7:0]**: Prescaler value

**In IrDA Low-power and normal IrDA mode:**

PSC[7:0] = IrDA normal and low-power baud rate

Used for programming the prescaler for dividing the USART source clock to achieve the low-power frequency:

The source clock is divided by the value given in the register (8 significant bits):

00000000: Reserved - do not program this value

00000001: divides the source clock by 1

00000010: divides the source clock by 2

...

**In smartcard mode:**

PSC[4:0]: Prescaler value

Used for programming the prescaler for dividing the USART source clock to provide the smartcard clock.

The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency:

00000: Reserved - do not program this value

00001: divides the source clock by 2

00010: divides the source clock by 4

00011: divides the source clock by 6

...

This bit field can only be written when the USART is disabled (UE=0).

*Note: Bits [7:5] must be kept at reset value if smartcard mode is used.*

*This bit field is reserved and must be kept at reset value when the smartcard and IrDA modes are not supported. Please refer to [Section 31.4: USART implementation on page 1012](#).*

### 31.8.6 USART receiver timeout register (USART\_RTOR)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BLEN[7:0]								RTO[23:16]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTO[15:0]								RTO[15:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

**Bits 31:24 BLEN[7:0]: Block Length**

This bit-field gives the Block length in smartcard T=1 Reception. Its value equals the number of information characters + the length of the Epilogue Field (1-LEC/2-CRC) - 1.

Examples:

BLEN = 0 -> 0 information characters + LEC

BLEN = 1 -> 0 information characters + CRC

BLEN = 255 -> 254 information characters + CRC (total 256 characters))

In smartcard mode, the Block length counter is reset when TXE=0.

This bit-field can be used also in other modes. In this case, the Block length counter is reset when RE=0 (receiver disabled) and/or when the EOBCF bit is written to 1.

*Note: This value can be programmed after the start of the block reception (using the data from the LEN character in the Prologue Field). It must be programmed only once per received block.*

**Bits 23:0 RTO[23:0]: Receiver timeout value**

This bit-field gives the Receiver timeout value in terms of number of bit duration.

In standard mode, the RTOF flag is set if, after the last received character, no new start bit is detected for more than the RTO value.

In smartcard mode, this value is used to implement the CWT and BWT. See smartcard section for more details.

In this case, the timeout measurement is done starting from the Start Bit of the last received character.

*Note: This value must only be programmed once per received character.*

**Note:** *RTOR can be written on the fly. If the new value is lower than or equal to the counter, the RTOF flag is set.*

*This register is reserved and forced by hardware to “0x00000000” when the Receiver timeout feature is not supported. Please refer to [Section 31.4: USART implementation on page 1012](#).*

### 31.8.7 USART request register (USART\_RQR)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TXFRQ	RXFRQ	MMRQ	SBKRQ	ABRRQ										
											w	w	w	w	w

Bits 31:5 Reserved, must be kept at reset value.

Bit 4 **TXFRQ**: Transmit data flush request

Writing 1 to this bit sets the TXE flag.

This allows to discard the transmit data. This bit must be used only in smartcard mode, when data has not been sent due to errors (NACK) and the FE flag is active in the USART\_ISR register.

If the USART does not support smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).

Bit 3 **RXFRQ**: Receive data flush request

Writing 1 to this bit clears the RXNE flag.

This allows to discard the received data without reading it, and avoid an overrun condition.

Bit 2 **MMRQ**: Mute mode request

Writing 1 to this bit puts the USART in mute mode and sets the RWU flag.

Bit 1 **SBKRQ**: Send break request

Writing 1 to this bit sets the SBKF flag and request to send a BREAK on the line, as soon as the transmit machine is available.

*Note: In the case the application needs to send the break character following all previously inserted data, including the ones not yet transmitted, the software should wait for the TXE flag assertion before setting the SBKRQ bit.*

Bit 0 **ABRRQ**: Auto baud rate request

Writing 1 to this bit resets the ABRF and ABRE flags in the USART\_ISR and request an automatic baud rate measurement on the next received data frame.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

### 31.8.8 USART interrupt and status register (USART\_ISR)

Address offset: 0x1C

Reset value: 0x0200 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	TEACK	Res.	RWU	SBKF	CMF	BUSY						
										r		r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res.	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:25 Reserved, must be kept at reset value.

Bits 24:22 Reserved, must be kept at reset value.

Bit 21 **TEACK**: Transmit enable acknowledge flag

This bit is set/reset by hardware, when the Transmit Enable value is taken into account by the USART.

It can be used when an idle frame request is generated by writing TE=0, followed by TE=1 in the USART\_CR1 register, in order to respect the TE=0 minimum period.

Bit 20 Reserved, must be kept at reset value.

Bit 19 **RWU**: Receiver wake-up from mute mode

This bit indicates if the USART is in mute mode. It is cleared/set by hardware when a wake-up/mute sequence is recognized. The mute mode control sequence (address or IDLE) is selected by the WAKE bit in the USART\_CR1 register.

When wake-up on Idle mode is selected, this bit can only be set by software, writing 1 to the MMRQ bit in the USART\_RQR register.

- 0: Receiver in active mode
- 1: Receiver in mute mode

Bit 18 **SBKF**: Send break flag

This bit indicates that a send break character was requested. It is set by software, by writing 1 to the SBKRQ bit in the USART\_RQR register. It is automatically reset by hardware during the stop bit of break transmission.

- 0: No break character is transmitted
- 1: Break character is transmitted

Bit 17 **CMF**: Character match flag

This bit is set by hardware, when the character defined by ADD[7:0] is received. It is cleared by software, writing 1 to the CMCF in the USART\_ICR register.

An interrupt is generated if CMIE=1 in the USART\_CR1 register.

- 0: No Character match detected
- 1: Character Match detected

Bit 16 **BUSY**: Busy flag

This bit is set and reset by hardware. It is active when a communication is ongoing on the RX line (successful start bit detected). It is reset at the end of the reception (successful or not).

- 0: USART is idle (no reception)
- 1: Reception on going

Bit 15 **ABRF**: Auto baud rate flag

This bit is set by hardware when the automatic baud rate has been set (RXNE will also be set, generating an interrupt if RXNEIE = 1) or when the auto baud rate operation was completed without success (ABRE=1) (ABRE, RXNE and FE are also set in this case)

It is cleared by software, in order to request a new auto baud rate detection, by writing 1 to the ABRRQ in the USART\_RQR register.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and kept at reset value.*

Bit 14 **ABRE**: Auto baud rate error

This bit is set by hardware if the baud rate measurement failed (baud rate out of range or character comparison failed)

It is cleared by software, by writing 1 to the ABRRQ bit in the USART\_RQR register.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and kept at reset value.*

## Bit 13 Reserved, must be kept at reset value.

Bit 12 **EOBF**: End of block flag

This bit is set by hardware when a complete block has been received (for example T=1 smartcard mode). The detection is done when the number of received bytes (from the start of the block, including the prologue) is equal or greater than BLEN + 4.

An interrupt is generated if EOBIIE = 1 in the USART\_CR1 register.

It is cleared by software, writing 1 to EOBCF in the USART\_ICR register.

0: End of Block not reached

1: End of Block (number of characters) reached

*Note: If smartcard mode is not supported, this bit is reserved and kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 11 **RTOF**: Receiver timeout

This bit is set by hardware when the timeout value, programmed in the RTOR register has lapsed, without any communication. It is cleared by software, writing 1 to the RTOCF bit in the USART\_ICR register.

An interrupt is generated if RTOIE=1 in the USART\_CR1 register.

In smartcard mode, the timeout corresponds to the CWT or BWT timings.

0: Timeout value not reached

1: Timeout value reached without any data reception

*Note: If a time equal to the value programmed in RTOR register separates 2 characters, RTOF is not set. If this time exceeds this value + 2 sample times (2/16 or 2/8, depending on the oversampling method), RTOF flag is set.*

*The counter counts even if RE = 0 but RTOF is set only when RE = 1. If the timeout has already elapsed when RE is set, then RTOF is set.*

*If the USART does not support the Receiver timeout feature, this bit is reserved and kept at reset value.*

Bit 10 **CTS**: CTS flag

This bit is set/reset by hardware. It is an inverted copy of the status of the CTS input pin.

0: CTS line set

1: CTS line reset

*Note: If the hardware flow control feature is not supported, this bit is reserved and kept at reset value.*

Bit 9 **CTSIF**: CTS interrupt flag

This bit is set by hardware when the CTS input toggles, if the CTSE bit is set. It is cleared by software, by writing 1 to the CTSCF bit in the USART\_ICR register.

An interrupt is generated if CTSIE=1 in the USART\_CR3 register.

0: No change occurred on the CTS status line

1: A change occurred on the CTS status line

*Note: If the hardware flow control feature is not supported, this bit is reserved and kept at reset value.*

Bit 8 **LBDIF**: LIN break detection flag

This bit is set by hardware when the LIN break is detected. It is cleared by software, by writing 1 to the LBDCF in the USART\_ICR.

An interrupt is generated if LBDIE = 1 in the USART\_CR2 register.

0: LIN Break not detected

1: LIN break detected

*Note: If the USART does not support LIN mode, this bit is reserved and kept at reset value.*

*Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the USART\_TDR register has been transferred into the shift register. It is cleared by a write to the USART\_TDR register. The TXE flag can also be cleared by writing 1 to the TXFRQ in the USART\_RQR register, in order to discard the data (only in smartcard T=0 mode, in case of transmission failure). An interrupt is generated if the TXEIE bit =1 in the USART\_CR1 register.

0: data is not transferred to the shift register  
1: data is transferred to the shift register)

*Note: This bit is used during single buffer transmission.*

Bit 6 **TC**: Transmission complete

This bit is set by hardware if the transmission of a frame containing data is complete and if TXE is set. An interrupt is generated if TCIE=1 in the USART\_CR1 register. It is cleared by software, writing 1 to the TCCF in the USART\_ICR register or by a write to the USART\_TDR register.

An interrupt is generated if TCIE=1 in the USART\_CR1 register.  
0: Transmission is not complete  
1: Transmission is complete

*Note: If TE bit is reset and no transmission is on going, the TC bit is set immediately.*

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART\_RDR register. It is cleared by a read to the USART\_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART\_RQR register.

An interrupt is generated if RXNEIE=1 in the USART\_CR1 register.  
0: data is not received  
1: Received data is ready to be read.

Bit 4 **IDLE**: Idle line detected

This bit is set by hardware when an Idle line is detected. An interrupt is generated if IDLEIE = 1 in the USART\_CR1 register. It is cleared by software, writing 1 to the IDLECF in the USART\_ICR register.

0: No Idle line is detected  
1: Idle line is detected

*Note: The IDLE bit will not be set again until the RXNE bit has been set (i.e. a new idle line occurs).*

*If mute mode is enabled (MME=1), IDLE is set if the USART is not mute (RWU=0), whatever the mute mode selected by the WAKE bit. If RWU=1, IDLE is not set.*

Bit 3 **ORE**: Overrun error

This bit is set by hardware when the data currently being received in the shift register is ready to be transferred into the RDR register while RXNE=1. It is cleared by a software, writing 1 to the ORECF, in the USART\_ICR register.

An interrupt is generated if RXNEIE=1 in the USART\_CR1 register, or EIE = 1 in the USART\_CR3 register.

0: No overrun error  
1: Overrun error is detected

*Note: When this bit is set, the RDR register content is not lost but the shift register is overwritten. An interrupt is generated if the ORE flag is set during multibuffer communication if the EIE bit is set.*

*This bit is permanently forced to 0 (no overrun detection) when the OVRDIS bit is set in the USART\_CR3 register.*

Bit 2 **NF**: START bit Noise detection flag

This bit is set by hardware when noise is detected on a received frame. It is cleared by software, writing 1 to the NFCF bit in the USART\_ICR register.

0: No noise is detected

1: Noise is detected

*Note: This bit does not generate an interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. An interrupt is generated when the NF flag is set during multibuffer communication if the EIE bit is set.*

*Note: When the line is noise-free, the NF flag can be disabled by programming the ONEBIT bit to 1 to increase the USART tolerance to deviations (Refer to [Section 31.5.5: Tolerance of the USART receiver to clock deviation on page 1028](#)).*

Bit 1 **FE**: Framing error

This bit is set by hardware when a de-synchronization, excessive noise or a break character is detected. It is cleared by software, writing 1 to the FECF bit in the USART\_ICR register.

In smartcard mode, in transmission, this bit is set when the maximum number of transmit attempts is reached without success (the card NACKs the data frame).

An interrupt is generated if EIE = 1 in the USART\_CR3 register.

0: No Framing error is detected

1: Framing error or break character is detected

Bit 0 **PE**: Parity error

This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by software, writing 1 to the PECE bit in the USART\_ICR register.

An interrupt is generated if PEIE = 1 in the USART\_CR1 register.

0: No parity error

1: Parity error

### 31.8.9 USART interrupt flag clear register (USART\_ICR)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CMCF	Res.
														rc_w1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	EOBCF	RTOCF	Res.	CTSCF	LBDCF	Res.	TCCF	Res.	IDLECF	ORECF	NCF	FECF	PECF
			rc_w1	rc_w1		rc_w1	rc_w1		rc_w1		rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:20 Reserved, must be kept at reset value.

Bits 19:18 Reserved, must be kept at reset value.

Bit 17 **CMCF**: Character match clear flag

Writing 1 to this bit clears the CMF flag in the USART\_ISR register.

Bits 16:13 Reserved, must be kept at reset value.

Bit 12 **EOBCF**: End of block clear flag

Writing 1 to this bit clears the EOBF flag in the USART\_ISR register.

*Note: If the USART does not support smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 11 **RTOCF**: Receiver timeout clear flag

Writing 1 to this bit clears the RTOF flag in the USART\_ISR register.

*Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 10 Reserved, must be kept at reset value.

Bit 9 **CTSCF**: CTS clear flag

Writing 1 to this bit clears the CTSIF flag in the USART\_ISR register.

*Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 8 **LBDGF**: LIN break detection clear flag

Writing 1 to this bit clears the LBDF flag in the USART\_ISR register.

*Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 31.4: USART implementation on page 1012](#).*

Bit 7 Reserved, must be kept at reset value.

Bit 6 **TCCF**: Transmission complete clear flag

Writing 1 to this bit clears the TC flag in the USART\_ISR register.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **IDLECF**: Idle line detected clear flag

Writing 1 to this bit clears the IDLE flag in the USART\_ISR register.

Bit 3 **ORECF**: Overrun error clear flag

Writing 1 to this bit clears the ORE flag in the USART\_ISR register.

Bit 2 **NCF**: Noise detected clear flag

Writing 1 to this bit clears the NF flag in the USART\_ISR register.

Bit 1 **FECF**: Framing error clear flag

Writing 1 to this bit clears the FE flag in the USART\_ISR register.

Bit 0 **PECF**: Parity error clear flag

Writing 1 to this bit clears the PE flag in the USART\_ISR register.

### 31.8.10 USART receive data register (USART\_RDR)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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.
RDR[8:0]															
							r	r	r	r	r	r	r	r	r

Bits 31:9 Reserved, must be kept at reset value.

Bits 8:0 **RDR[8:0]**: Receive data value

Contains the received data character.

The RDR register provides the parallel interface between the input shift register and the internal bus (see [Figure 329](#)).

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

### 31.8.11 USART transmit data register (USART\_TDR)

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDR[8:0]														
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:9 Reserved, must be kept at reset value.

Bits 8:0 **TDR[8:0]**: Transmit data value

Contains the data character to be transmitted.

The TDR register provides the parallel interface between the internal bus and the output shift register (see [Figure 329](#)).

When transmitting with the parity enabled (PCE bit set to 1 in the USART\_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

*Note: This register must be written only when TXE=1.*

### 31.8.12 USART register map

The table below gives the USART register map and reset values.

Table 206. USART register map and reset values

Offset	Register name reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18
0x00	USART_CR1	PSC	Res.	Res.	M1	EOBIE	RTOIE	DEAT4	DEAT3	DEAT2	DEAT1	DEAT0	DEDT4	DEDT3	19
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	USART_CR2	ADD[7:4]				ADD[3:0]				RTOEN	ABRMD1	ABRMD0	ABREN	MSBFIRST	DATANV
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 206. USART register map and reset values (continued)

Offset	Register name reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x08	<b>USART_CR3</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																																
0x0C	<b>USART_BRR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																
0x10	<b>USART_GTPR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																
0x14	<b>USART_RTOR</b>	BLEN[7:0]								GT[7:0]								PSC[7:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x18	<b>USART_RQR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																
0x1C	<b>USART_ISR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																
0x20	<b>USART_ICR</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																																
0x24	<b>USART_RDR</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																																
0x28	<b>USART_TDR</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 on page 69](#) for the register boundary addresses.

## 32 Serial peripheral interface / integrated interchip sound (SPI/I2S)

### 32.1 Introduction

The SPI/I<sup>2</sup>S interface can be used to communicate with external devices using the SPI protocol or the I<sup>2</sup>S audio protocol. SPI or I<sup>2</sup>S mode is selectable by software. SPI Motorola mode is selected by default after a device reset.

The serial peripheral interface (SPI) protocol supports half-duplex, full-duplex and simplex synchronous, serial communication with external devices. The interface can be configured as master and in this case it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multimaster configuration.

The integrated interchip sound (I<sup>2</sup>S) protocol is also a synchronous serial communication interface. It can operate in slave or master mode with half-duplex communication. Full-duplex operations are possible by combining two I<sup>2</sup>S blocks. It can address four different audio standards including the Philips I<sup>2</sup>S standard, the MSB- and LSB-justified standards and the PCM standard.

### 32.2 SPI main features

- Master or slave operation
- Full-duplex synchronous transfers on three lines
- Half-duplex synchronous transfer on two lines (with bidirectional data line)
- Simplex synchronous transfers on two lines (with unidirectional data line)
- 4 to 16-bit data size selection
- Multimaster mode capability
- 8 master mode baud rate prescalers up to  $f_{PCLK}/2$
- Slave mode frequency up to  $f_{PCLK}/2$
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- SPI Motorola support
- Hardware CRC feature for reliable communication:
  - CRC value can be transmitted as last byte in Tx mode
  - Automatic CRC error checking for last received byte
- Master mode fault, overrun flags with interrupt capability
- CRC Error flag
- Two 32-bit embedded Rx and Tx FIFOs with DMA capability
- Enhanced TI and NSS pulse modes support

### 32.3 I2S main features

- Half-duplex communication (only transmitter or receiver)
- Master or slave operations
- 8-bit programmable linear prescaler to reach accurate audio sample frequencies (from 8 kHz to 192 kHz)
- Data format may be 16-bit, 24-bit, or 32-bit
- Packet frame is fixed to 16-bit (16-bit data frame) or 32-bit (16-bit, 24-bit, 32-bit data frame) by audio channel
- Programmable clock polarity (steady state)
- Underrun flag in slave transmission mode, overrun flag in reception mode (master and slave) and Frame Error Flag in reception and transmitter mode (slave only)
- 16-bit register for transmission and reception with one data register for both channel sides
- Supported I<sup>2</sup>S protocols:
  - I<sup>2</sup>S Philips standard
  - MSB-justified standard (left-justified)
  - LSB-justified standard (right-justified)
  - PCM standard (with short and long frame synchronization on 16-bit channel frame or 16-bit data frame extended to 32-bit channel frame)
- Data direction is always MSB first
- DMA capability for transmission and reception (16-bit wide)
- Master clock can be output to drive an external audio component. The ratio is fixed at  $256 \times f_s$  for all I2S modes, and to  $128 \times f_s$  for all PCM modes (where  $f_s$  is the audio sampling frequency).

### 32.4 SPI/I2S implementation

The following table describes all the SPI instances and their features embedded in the devices.

**Table 207. STM32F75xxx and STM32F74xxx SPI and SPI/I2S implementation**

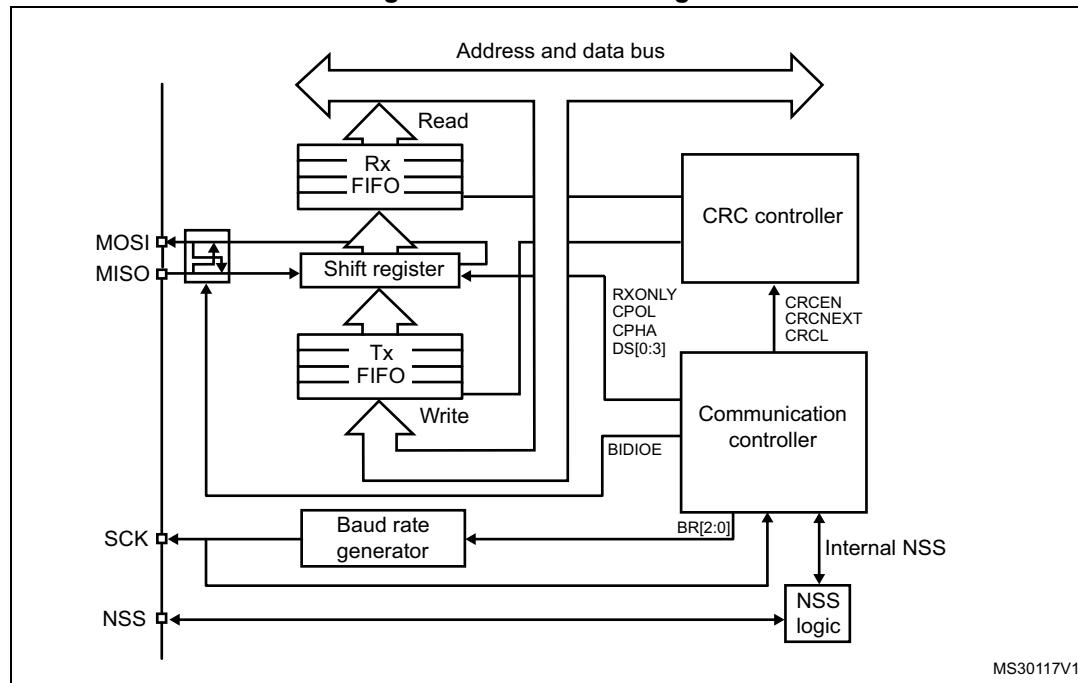
SPI Features	SPI2S1	SPI2S2	SPI2S3	SPI4	SPI5	SPI6
Enhanced NSSP & TI modes	Yes	Yes	Yes	Yes	Yes	Yes
Hardware CRC calculation	Yes	Yes	Yes	Yes	Yes	Yes
I <sup>2</sup> S support	Yes	Yes	Yes	No	No	No
Data size configurable	from 4 to 16 bits					
Rx/Tx FIFO size	32 bits					
Wake-up capability from Low-power Sleep	Yes	Yes	Yes	Yes	Yes	Yes

## 32.5 SPI functional description

### 32.5.1 General description

The SPI allows synchronous, serial communication between the MCU and external devices. Application software can manage the communication by polling the status flag or using dedicated SPI interrupt. The main elements of SPI and their interactions are shown in the following block diagram [Figure 354](#).

**Figure 354. 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 32.5.5: Slave select \(NSS\) pin management](#) for details.

The SPI bus allows the communication between one master device and one or more slave devices. The bus consists of at least two wires - one for the clock signal and the other for synchronous data transfer. Other signals can be added depending on the data exchange between SPI nodes and their slave select signal management.

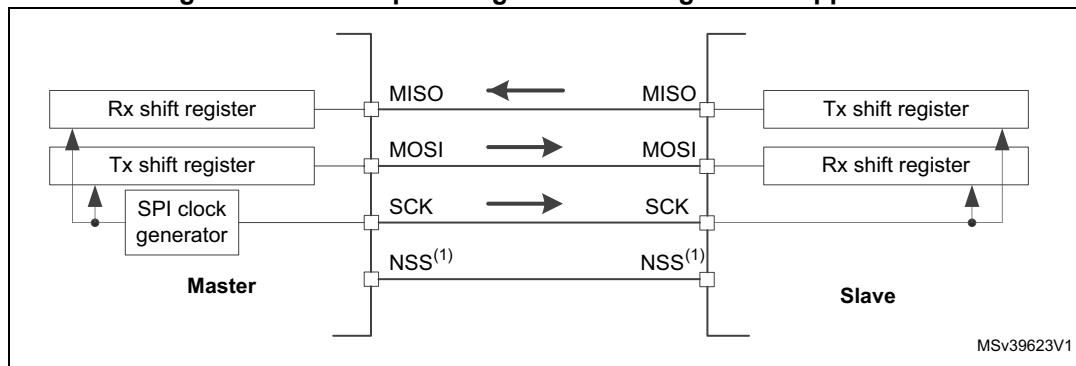
### 32.5.2 Communications between one master and one slave

The SPI allows the MCU to communicate using different configurations, depending on the device targeted and the application requirements. These configurations use 2 or 3 wires (with software NSS management) or 3 or 4 wires (with hardware NSS management). Communication is always initiated by the master.

#### Full-duplex communication

By default, the SPI is configured for full-duplex communication. In this configuration, the shift registers of the master and slave are linked using two unidirectional lines between the MOSI and the MISO pins. During SPI communication, data is shifted synchronously on the SCK clock edges provided by the master. The master transmits the data to be sent to the slave via the MOSI line and receives data from the slave via the MISO line. When the data frame transfer is complete (all the bits are shifted) the information between the master and slave is exchanged.

**Figure 355. Full-duplex single master/ single slave application**

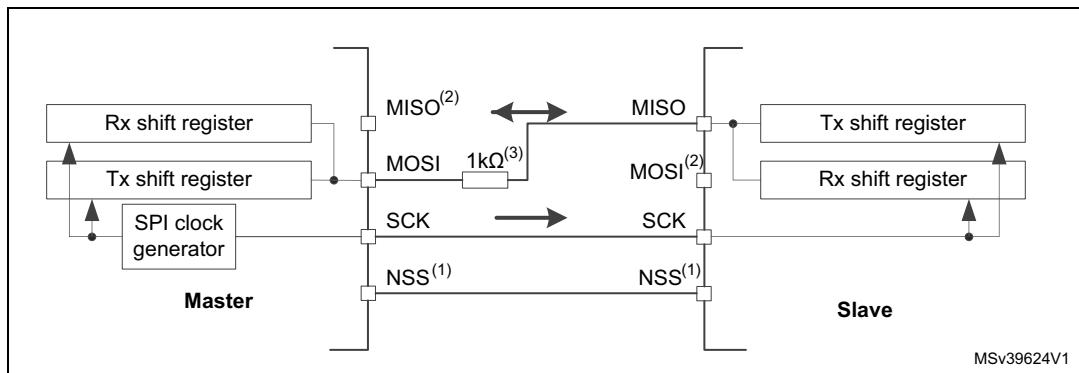


1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 32.5.5: Slave select \(NSS\) pin management](#).

#### Half-duplex communication

The SPI can communicate in half-duplex mode by setting the BIDIMODE bit in the SPIx\_CR1 register. In this configuration, one single cross connection line is used to link the shift registers of the master and slave together. During this communication, the data is synchronously shifted between the shift registers on the SCK clock edge in the transfer direction selected reciprocally by both master and slave with the BDIOE bit in their SPIx\_CR1 registers. In this configuration, the master's MISO pin and the slave's MOSI pin are free for other application uses and act as GPIOs.

Figure 356. Half-duplex single master/ single slave application



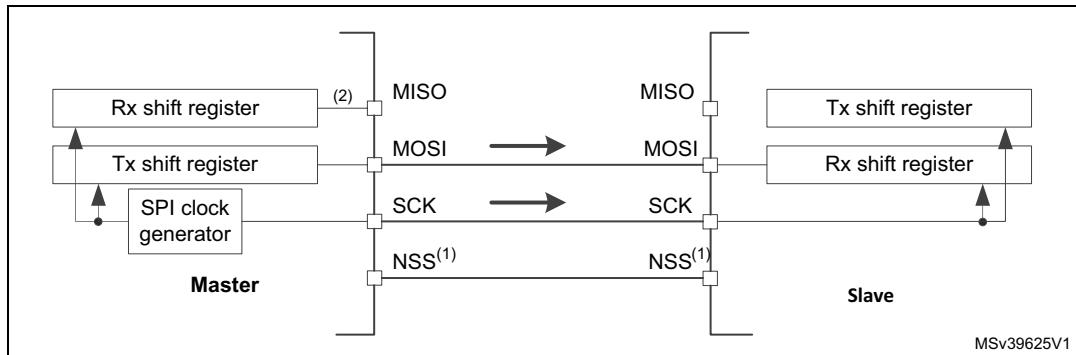
1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 32.5.5: Slave select \(NSS\) pin management](#).
2. In this configuration, the master's MISO pin and the slave's MOSI pin can be used as GPIOs.
3. A critical situation can happen when communication direction is changed not synchronously between two nodes working at bidirectional mode and new transmitter accesses the common data line while former transmitter still keeps an opposite value on the line (the value depends on SPI configuration and communication data). Both nodes then fight while providing opposite output levels on the common line temporary till next node changes its direction settings correspondingly, too. It is suggested to insert a serial resistance between MISO and MOSI pins at this mode to protect the outputs and limit the current blowing between them at this situation.

## Simplex communications

The SPI can communicate in simplex mode by setting the SPI in transmit-only or in receive-only using the RXONLY bit in the SPIx\_CR1 register. In this configuration, only one line is used for the transfer between the shift registers of the master and slave. The remaining MISO and MOSI pins pair is not used for communication and can be used as standard GPIOs.

- **Transmit-only mode (RXONLY=0):** The configuration settings are the same as for full-duplex. The application has to ignore the information captured on the unused input pin. This pin can be used as a standard GPIO.
- **Receive-only mode (RXONLY=1):** The application can disable the SPI output function by setting the RXONLY bit. In slave configuration, the MISO output is disabled and the pin can be used as a GPIO. The slave continues to receive data from the MOSI pin while its slave select signal is active (see [32.5.5: Slave select \(NSS\) pin management](#)). Received data events appear depending on the data buffer configuration. In the master configuration, the MOSI output is disabled and the pin can be used as a GPIO. The clock signal is generated continuously as long as the SPI is enabled. The only way to stop the clock is to clear the RXONLY bit or the SPE bit and wait until the incoming pattern from the MISO pin is finished and fills the data buffer structure, depending on its configuration.

**Figure 357. 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 32.5.5: Slave select \(NSS\) pin management](#).
2. An accidental input information is captured at the input of transmitter Rx shift register. All the events associated with the transmitter receive flow must be ignored in standard transmit only mode (e.g. OVR flag).
3. In this configuration, both the MISO pins can be used as GPIOs.

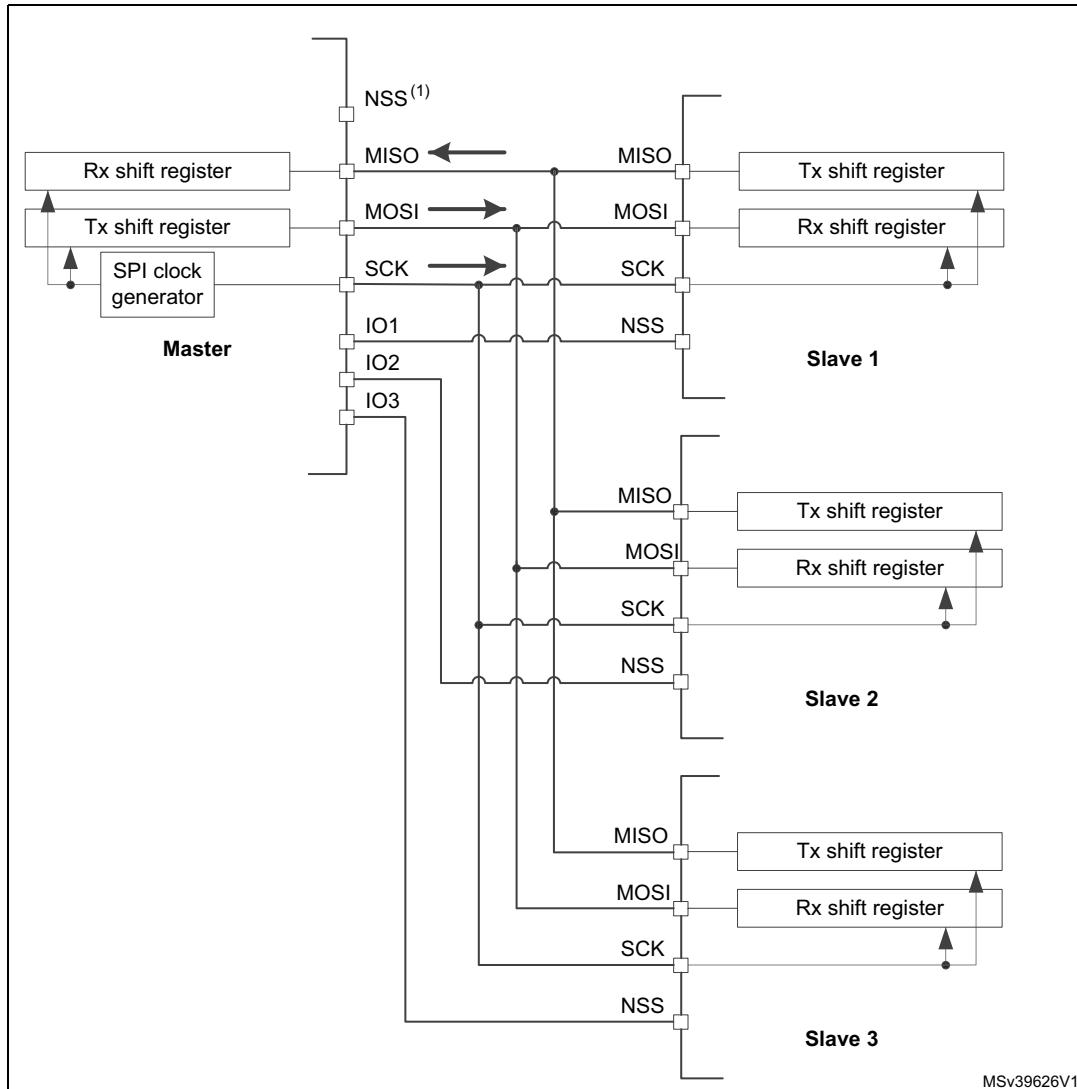
**Note:**

*Any simplex communication can be alternatively replaced by a variant of the half-duplex communication with a constant setting of the transaction direction (bidirectional mode is enabled while BDIO bit is not changed).*

### 32.5.3 Standard multislide 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 358](#)). 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 358. Master and three independent slaves



1. NSS pin is not used on master side at this configuration. It has to be managed internally ( $SSM=1$ ,  $SSI=1$ ) to prevent any MODF error.
2. As MISO pins of the slaves are connected together, all slaves must have the GPIO configuration of their MISO pin set as alternate function open-drain (see I/O alternate function input/output section (GPIO)).

### 32.5.4 Multimaster communication

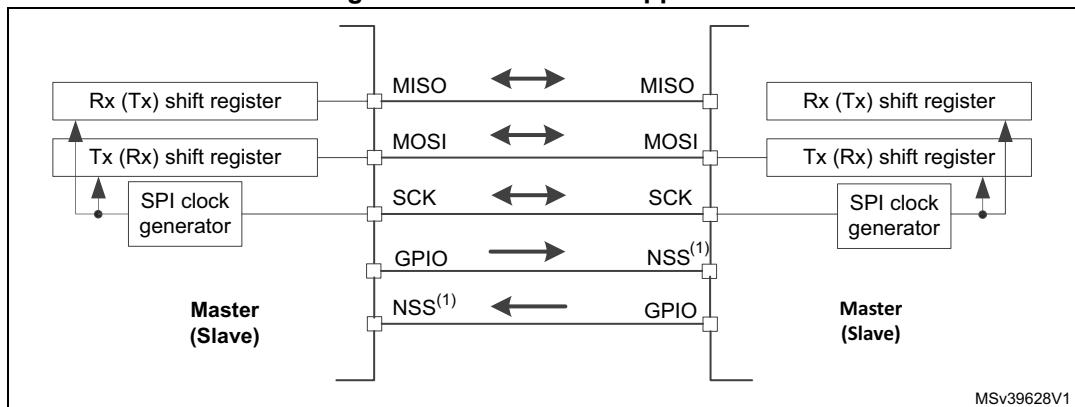
Unless SPI bus is not designed for a multimaster capability primarily, the user can use build in feature which detects a potential conflict between two nodes trying to master the bus at the same time. For this detection, NSS pin is used configured at hardware input mode.

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 359. Multimaster 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.

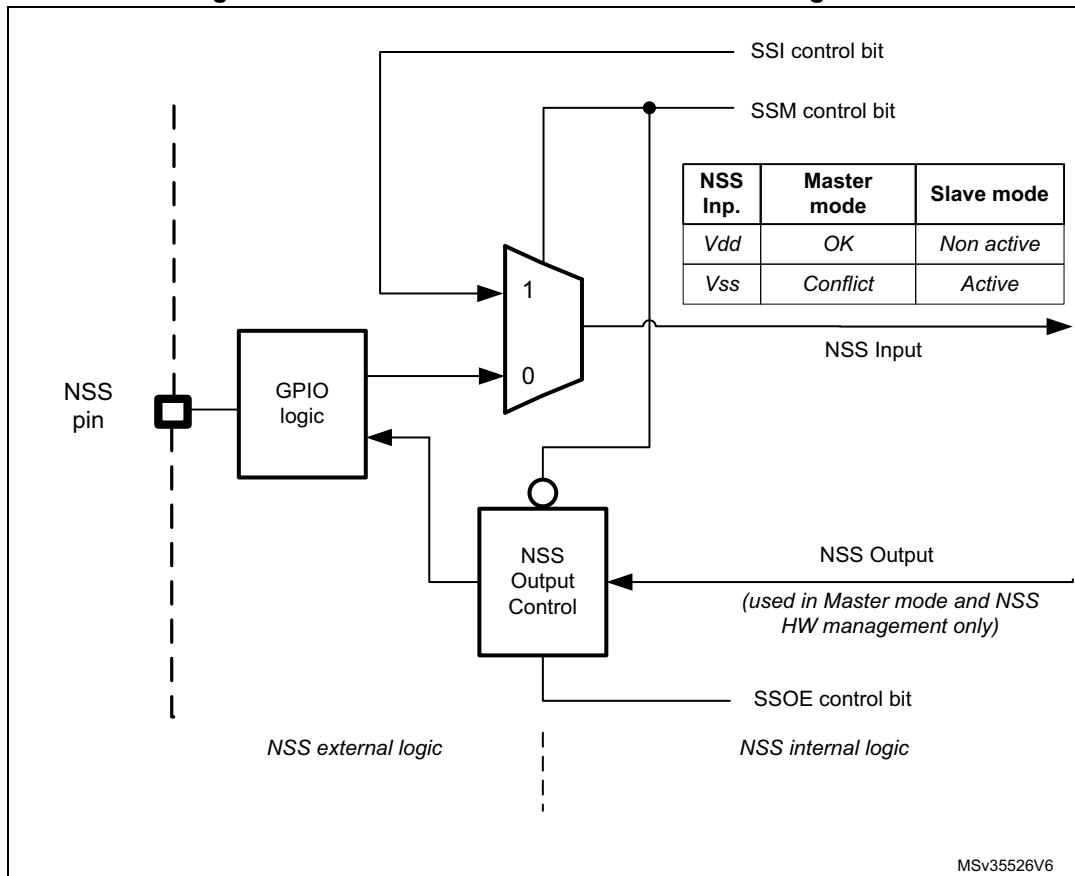
### 32.5.5 Slave select (NSS) pin management

In slave mode, the NSS works as a standard “chip select” input and lets the slave communicate with the master. In master mode, NSS can be used either as output or input. As an input it can prevent multimaster bus collision, and as an output it can drive a slave select signal of a single slave.

Hardware or software slave select management can be set using the SSM bit in the SPIx\_CR1 register:

- **Software NSS management (SSM = 1):** in this configuration, slave select information is driven internally by the SSI bit value in register SPIx\_CR1. The external NSS pin is free for other application uses.
- **Hardware NSS management (SSM = 0):** in this case, there are two possible configurations. The configuration used depends on the NSS output configuration (SSOE bit in register SPIx\_CR1).
  - **NSS output enable (SSM=0,SSOE = 1):** this configuration is only used when the MCU is set as master. The NSS pin is managed by the hardware. The NSS signal is driven low as soon as the SPI is enabled in master mode (SPE=1), and is kept low until the SPI is disabled (SPE =0). A pulse can be generated between continuous communications if NSS pulse mode is activated (NSSP=1). The SPI cannot work in multimaster configuration with this NSS setting.
  - **NSS output disable (SSM=0, SSOE = 0):** if the microcontroller is acting as the master on the bus, this configuration allows multimaster capability. If the NSS pin is pulled low in this mode, the SPI enters master mode fault state and the device is automatically reconfigured in slave mode. In slave mode, the NSS pin works as a standard “chip select” input and the slave is selected while NSS line is at low level.

Figure 360. Hardware/software slave select management



### 32.5.6 Communication formats

During SPI communication, receive and transmit operations are performed simultaneously. The serial clock (SCK) synchronizes the shifting and sampling of the information on the data lines. The communication format depends on the clock phase, the clock polarity and the data frame format. To be able to communicate together, the master and slaves devices must follow the same communication format.

#### Clock phase and polarity controls

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits in the SPIx\_CR1 register. The CPOL (clock polarity) bit controls the idle state value of the clock when no data is being transferred. This bit affects both master and slave modes. If CPOL is reset, the SCK pin has a low-level idle state. If CPOL is set, the SCK pin has a high-level idle state.

If the CPHA bit is set, the second edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set). Data are latched on each occurrence of this clock transition type. If the CPHA bit is reset, the first edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is set, rising edge if the CPOL bit is reset). Data are latched on each occurrence of this clock transition type.

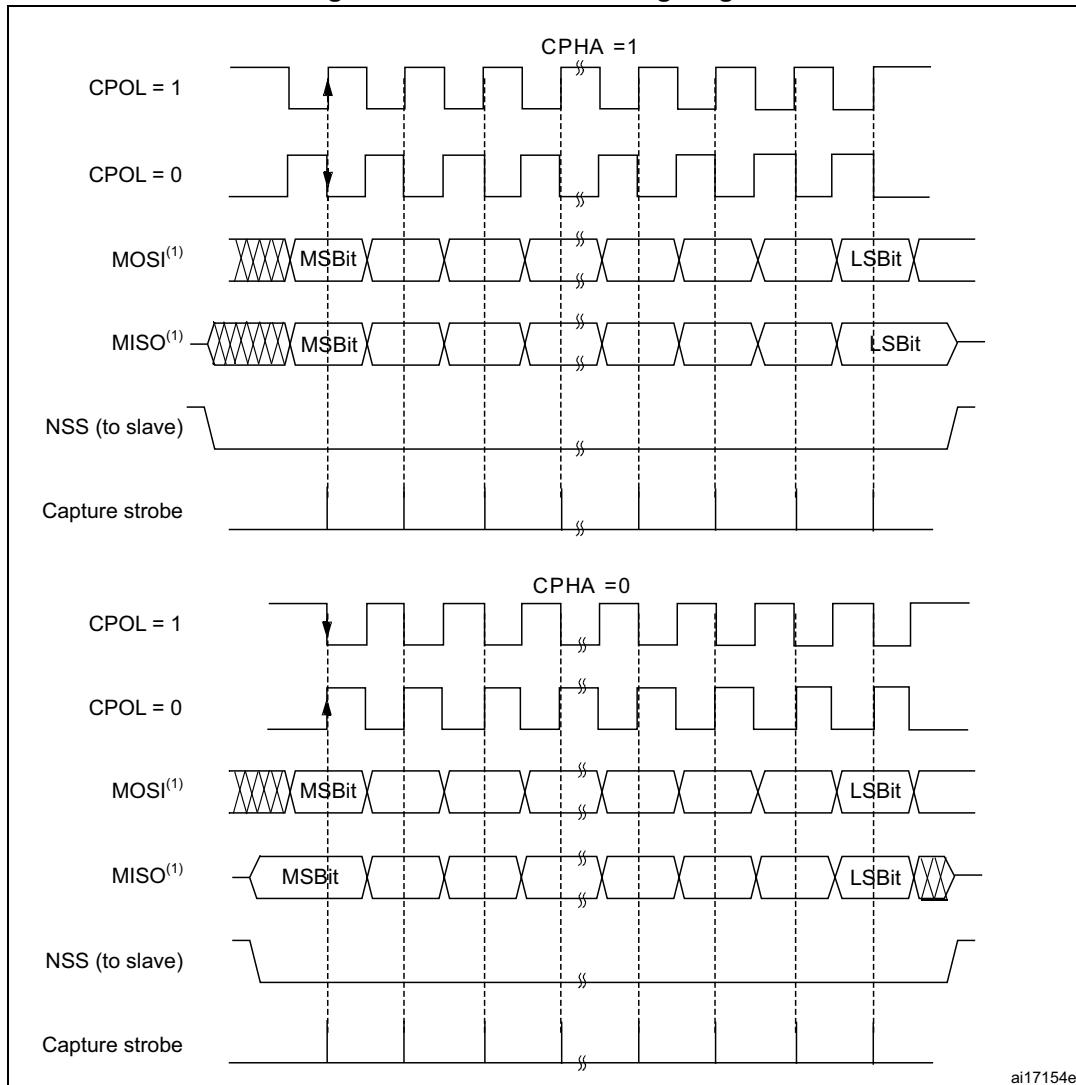
The combination of CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge.

*Figure 361*, 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).

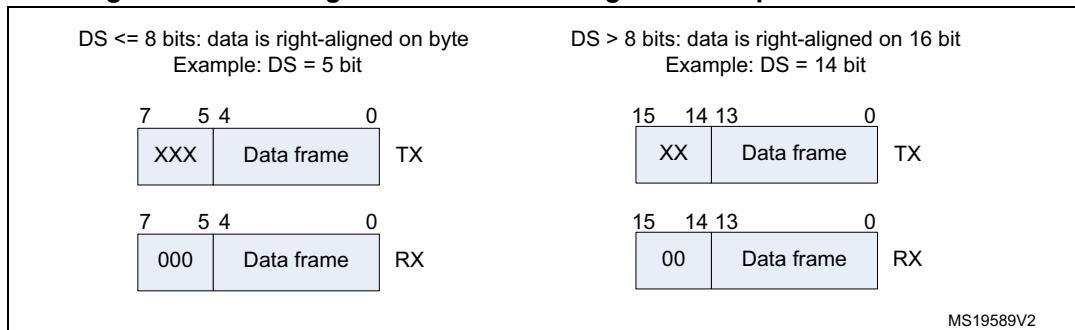
**Figure 361. Data clock timing diagram**



1. The order of data bits depends on LSBFIRST bit setting.

### Data frame format

The SPI shift register can be set up to shift out MSB-first or LSB-first, depending on the value of the LSBFIRST bit. The data frame size is chosen by using the DS bits. It can be set from 4-bit up to 16-bit length and the setting applies for both transmission and reception. Whatever the selected data frame size, read access to the FIFO must be aligned with the FRXTH level. When the SPIx\_DR register is accessed, data frames are always right-aligned into either a byte (if the data fits into a byte) or a half-word (see *Figure 362*). During communication, only bits within the data frame are clocked and transferred.

**Figure 362. Data alignment when data length is not equal to 8-bit or 16-bit**

**Note:** The minimum data length is 4 bits. If a data length of less than 4 bits is selected, it is forced to an 8-bit data frame size.

### 32.5.7 Configuration of SPI

The configuration procedure is almost the same for master and slave. For specific mode setups, follow the dedicated sections. When a standard communication is to be initialized, perform these steps:

1. Write proper GPIO registers: Configure GPIO for MOSI, MISO and SCK pins.
2. Write to the SPI\_CR1 register:
  - a) Configure the serial clock baud rate using the BR[2:0] bits (Note: 4).
  - b) Configure the CPOL and CPHA bits combination to define one of the four relationships between the data transfer and the serial clock (CPHA must be cleared in NSSP mode). (Note: 2 - except the case when CRC is enabled at TI mode).
  - c) Select simplex or half-duplex mode by configuring RXONLY or BIDIMODE and BIDIOE (RXONLY and BIDIMODE cannot be set at the same time).
  - d) Configure the LSBFIRST bit to define the frame format (Note: 2).
  - e) Configure the CRCL and CRCEN bits if CRC is needed (while SCK clock signal is at idle state).
  - f) Configure SSM and SSI (Notes: 2 & 3).
  - g) Configure the MSTR bit (in multimaster NSS configuration, avoid conflict state on NSS if master is configured to prevent MODF error).
3. Write to SPI\_CR2 register:
  - a) Configure the DS[3:0] bits to select the data length for the transfer.
  - b) Configure SSOE (Notes: 1 & 2 & 3).
  - c) Set the FRF bit if the TI protocol is required (keep NSSP bit cleared in TI mode).
  - d) Set the NSSP bit if the NSS pulse mode between two data units is required (keep CHPA and TI bits cleared in NSSP mode).
  - e) Configure the RXTH bit. The RXFIFO threshold must be aligned to the read access size for the SPIx\_DR register.
  - f) Initialize LDMA\_TX and LDMA\_RX bits if DMA is used in packed mode.
4. Write to SPI\_CRCPR register: Configure the CRC polynomial if needed.
5. Write proper DMA registers: Configure DMA streams dedicated for SPI Tx and Rx in DMA registers if the DMA streams are used.

- Note:
- (1) Step is not required in slave mode.
  - (2) Step is not required in TI mode.
  - (3) Step is not required in NSSP mode.
  - (4) The step is not required in slave mode except slave working at TI mode

### 32.5.8 Procedure for enabling SPI

It is recommended to enable the SPI slave before the master sends the clock. If not, undesired data transmission might occur. The data register of the slave must already contain data to be sent before starting communication with the master (either on the first edge of the communication clock, or before the end of the ongoing communication if the clock signal is continuous). The SCK signal must be settled at an idle state level corresponding to the selected polarity before the SPI slave is enabled.

The master at full-duplex (or in any transmit-only mode) starts to communicate when the SPI is enabled and TXFIFO is not empty, or with the next write to TXFIFO.

In any master receive only mode (RXONLY = 1 or BIDIMODE = 1 & BIDIOE = 0), master starts to communicate and the clock starts running immediately after SPI is enabled.

For handling DMA, follow the dedicated section.

### 32.5.9 Data transmission and reception procedures

#### RXFIFO and TXFIFO

All SPI data transactions pass through the 32-bit embedded FIFOs. This enables the SPI to work in a continuous flow, and prevents overruns when the data frame size is short. Each direction has its own FIFO called TXFIFO and RXFIFO. These FIFOs are used in all SPI modes except for receiver-only mode (slave or master) with CRC calculation enabled (see [Section 32.5.14: CRC calculation](#)).

The handling of FIFOs depends on the data exchange mode (duplex, simplex), data frame format (number of bits in the frame), access size performed on the FIFO data registers (8-bit or 16-bit), and whether or not data packing is used when accessing the FIFOs (see [Section 32.5.13: TI mode](#)).

A read access to the SPIx\_DR register returns the oldest value stored in RXFIFO that has not been read yet. A write access to the SPIx\_DR stores the written data in the TXFIFO at the end of a send queue. The read access must be always aligned with the RXFIFO threshold configured by the FRXTH bit in SPIx\_CR2 register. FTLVL[1:0] and FRLVL[1:0] bits indicate the current occupancy level of both FIFOs.

A read access to the SPIx\_DR register must be managed by the RXNE event. This event is triggered when data is stored in RXFIFO and the threshold (defined by FRXTH bit) is reached. When RXNE is cleared, RXFIFO is considered to be empty. In a similar way, write access of a data frame to be transmitted is managed by the TXE event. This event is triggered when the TXFIFO level is less than or equal to half of its capacity. Otherwise TXE is cleared and the TXFIFO is considered as full. In this way, RXFIFO can store up to four data frames, whereas TXFIFO can only store up to three when the data frame format is not greater than 8 bits. This difference prevents possible corruption of 3x 8-bit data frames already stored in the TXFIFO when software tries to write more data in 16-bit mode into TXFIFO. Both TXE and RXNE events can be polled or handled by interrupts. See [Figure 364](#) through [Figure 367](#).

Another way to manage the data exchange is to use DMA (see [Communication using DMA \(direct memory addressing\)](#)).

If the next data is received when the RXFIFO is full, an overrun event occurs (see description of OVR flag at [Section 32.5.10: SPI status flags](#)). An overrun event can be polled or handled by an interrupt.

The BSY bit being set indicates ongoing transaction of a current data frame. When the clock signal runs continuously, the BSY flag stays set between data frames at master but becomes low for a minimum duration of one SPI clock at slave between each data frame transfer.

### Sequence handling

A few data frames can be passed at single sequence to complete a message. When transmission is enabled, a sequence begins and continues while any data is present in the TXFIFO of the master. The clock signal is provided continuously by the master until TXFIFO becomes empty, then it stops waiting for additional data.

In receive-only modes, half-duplex (BIDIMODE=1, BIDIOE=0) or simplex (BIDIMODE=0, RXONLY=1) the master starts the sequence immediately when both SPI is enabled and receive-only mode is activated. The clock signal is provided by the master and it does not stop until either SPI or receive-only mode is disabled by the master. The master receives data frames continuously up to this moment.

While the master can provide all the transactions in continuous mode (SCK signal is continuous) it has to respect slave capability to handle data flow and its content at anytime. When necessary, the master must slow down the communication and provide either a slower clock or separate frames or data sessions with sufficient delays. Be aware there is no underflow error signal for master or slave in SPI mode, and data from the slave is always transacted and processed by the master even if the slave could not prepare it correctly in time. It is preferable for the slave to use DMA, especially when data frames are shorter and bus rate is high.

Each sequence must be encased by the NSS pulse in parallel with the multislide system to select just one of the slaves for communication. In a single slave system it is not necessary to control the slave with NSS, but it is often better to provide the pulse here too, to synchronize the slave with the beginning of each data sequence. NSS can be managed by both software and hardware (see [Section 32.5.5: Slave select \(NSS\) pin management](#)).

When the BSY bit is set it signifies an ongoing data frame transaction. When the dedicated frame transaction is finished, the RXNE flag is raised. The last bit is just sampled and the complete data frame is stored in the RXFIFO.

### Procedure for disabling the SPI

When SPI is disabled, it is mandatory to follow the disable procedures described in this paragraph. It is important to do this before the system enters a low-power mode when the peripheral clock is stopped. Ongoing transactions can be corrupted in this case. In some modes the disable procedure is the only way to stop continuous communication running.

Master in full-duplex or transmit only mode can finish any transaction when it stops providing data for transmission. In this case, the clock stops after the last data transaction. Special care must be taken in packing mode when an odd number of data frames are transacted to prevent some dummy byte exchange (refer to [Data packing](#) section). Before the SPI is disabled in these modes, the user must follow standard disable procedure. When

the SPI is disabled at the master transmitter while a frame transaction is ongoing or next data frame is stored in TXFIFO, the SPI behavior is not guaranteed.

When the master is in any receive only mode, the only way to stop the continuous clock is to disable the peripheral by SPE=0. This must occur in specific time window within last data frame transaction just between the sampling time of its first bit and before its last bit transfer starts (in order to receive a complete number of expected data frames and to prevent any additional “dummy” data reading after the last valid data frame). Specific procedure must be followed when disabling SPI in this mode.

Data received but not read remains stored in RXFIFO when the SPI is disabled, and must be processed the next time the SPI is enabled, before starting a new sequence. To prevent having unread data, ensure that RXFIFO is empty when disabling the SPI, by using the correct disabling procedure, or by initializing all the SPI registers with a software reset via the control of a specific register dedicated to peripheral reset (see the SPIiRST bits in the RCC\_APBiRSTR registers).

Standard disable procedure is based on pulling BSY status together with FTLVL[1:0] to check if a transmission session is fully completed. This check can be done in specific cases, too, when it is necessary to identify the end of ongoing transactions, for example:

- When NSS signal is managed by software and master has to provide proper end of NSS pulse for slave, or
- When transactions’ streams from DMA or FIFO are completed while the last data frame or CRC frame transaction is still ongoing in the peripheral bus.

The correct disable procedure is (except when receive only mode is used):

1. Wait until FTLVL[1:0] = 00 (no more data to transmit).
2. Wait until BSY=0 (the last data frame is processed).
3. Disable the SPI (SPE=0).
4. Read data until FRLVL[1:0] = 00 (read all the received data).

The correct disable procedure for certain receive only modes is:

1. Interrupt the receive flow by disabling SPI (SPE=0) in the specific time window while the last data frame is ongoing.
2. Wait until BSY=0 (the last data frame is processed).
3. Read data until FRLVL[1:0] = 00 (read all the received data).

**Note:**

*If packing mode is used and an odd number of data frames with a format less than or equal to 8 bits (fitting into one byte) has to be received, FRXTH must be set when FRLVL[1:0] = 01, in order to generate the RXNE event to read the last odd data frame and to keep good FIFO pointer alignment.*

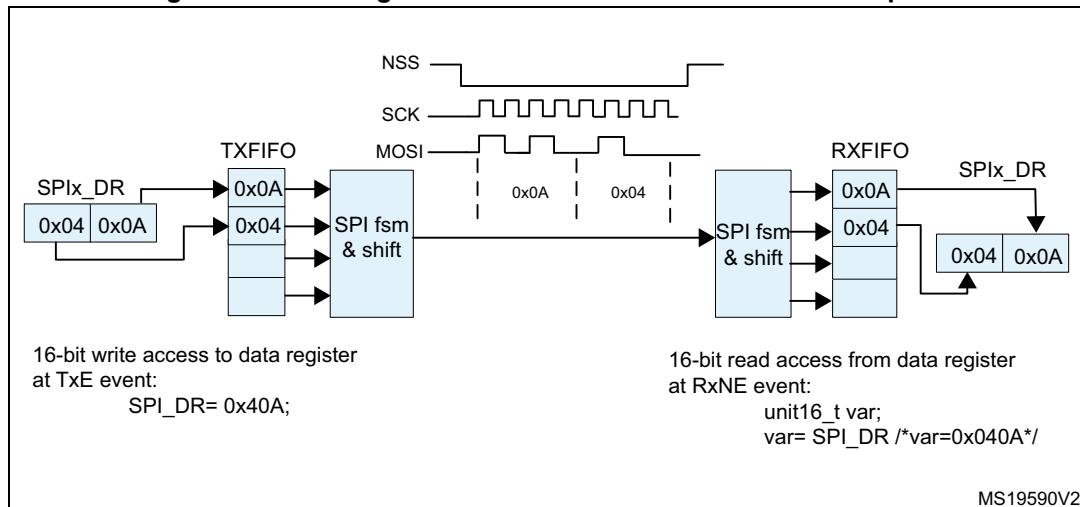
### Data packing

When the data frame size fits into one byte (less than or equal to 8 bits), data packing is used automatically when any read or write 16-bit access is performed on the SPIx\_DR register. The double data frame pattern is handled in parallel in this case. At first, the SPI operates using the pattern stored in the LSB of the accessed word, then with the other half stored in the MSB. [Figure 363](#) provides an example of data packing mode sequence handling. Two data frames are sent after the single 16-bit access the SPIx\_DR register of the transmitter. This sequence can generate just one RXNE event in the receiver if the RXFIFO threshold is set to 16 bits (FRXTH=0). The receiver then has to access both data frames by a single 16-bit read of SPIx\_DR as a response to this single RXNE event. The

RxFIFO threshold setting and the following read access must be always kept aligned at the receiver side, as data can be lost if it is not in line.

A specific problem appears if an odd number of such “fit into one byte” data frames must be handled. On the transmitter side, writing the last data frame of any odd sequence with an 8-bit access to SPIx\_DR is enough. The receiver has to change the Rx\_FIFO threshold level for the last data frame received in the odd sequence of frames in order to generate the RXNE event.

**Figure 363. Packing data in FIFO for transmission and reception**



1. In this example: Data size DS[3:0] is 4-bit configured, CPOL=0, CPHA=1 and LSBFIRST =0. The Data storage is always right aligned while the valid bits are performed on the bus only, the content of LSB byte goes first on the bus, the unused bits are not taken into account on the transmitter side and padded by zeros at the receiver side.

### Communication using DMA (direct memory addressing)

To operate at its maximum speed and to facilitate the data register read/write process required to avoid overrun, the SPI features a DMA capability, which implements a simple request/acknowledge protocol.

A DMA access is requested when the TXDMAEN or RXDMAEN enable bit in the SPIx\_CR2 register is set. Separate requests must be issued to the Tx and Rx buffers.

- In transmission, a DMA request is issued each time TXE is set to 1. The DMA then writes to the SPIx\_DR register.
- In reception, a DMA request is issued each time RXNE is set to 1. The DMA then reads the SPIx\_DR register.

See [Figure 364](#) through [Figure 367](#).

When the SPI is used only to transmit data, it is possible to enable only the SPI Tx DMA channel. In this case, the OVR flag is set because the data received is not read. When the SPI is used only to receive data, it is possible to enable only the SPI Rx DMA channel.

In transmission mode, when the DMA has written all the data to be transmitted (the TCIF flag is set in the DMA\_ISR register), the BSY flag can be monitored to ensure that the SPI communication is complete. This is required to avoid corrupting the last transmission before disabling the SPI or entering the Stop mode. The software must first wait until FTLVL[1:0]=00 and then until BSY=0.

When starting communication using DMA, to prevent DMA channel management raising error events, these steps must be followed in order:

1. Enable DMA Rx buffer in the RXDMAEN bit in the SPI\_CR2 register, if DMA Rx is used.
2. Enable DMA streams for Tx and Rx in DMA registers, if the streams are used.
3. Enable DMA Tx buffer in the TXDMAEN bit in the SPI\_CR2 register, if DMA Tx is used.
4. Enable the SPI by setting the SPE bit.

To close communication it is mandatory to follow these steps in order:

1. Disable DMA streams for Tx and Rx in the DMA registers, if the streams are used.
2. Disable the SPI by following the SPI disable procedure.
3. Disable DMA Tx and Rx buffers by clearing the TXDMAEN and RXDMAEN bits in the SPI\_CR2 register, if DMA Tx and/or DMA Rx are used.

### Packing with DMA

If the transfers are managed by DMA (TXDMAEN and RXDMAEN set in the SPIx\_CR2 register) packing mode is enabled/disabled automatically depending on the PSIZE value configured for SPI TX and the SPI RX DMA channel. If the DMA channel PSIZE value is equal to 16-bit and SPI data size is less than or equal to 8-bit, then packing mode is enabled. The DMA then automatically manages the write operations to the SPIx\_DR register.

If data packing mode is used and the number of data to transfer is not a multiple of two, the LDMA\_TX/LDMA\_RX bits must be set. The SPI then considers only one data for the transmission or reception to serve the last DMA transfer (for more details refer to [Data packing on page 1087](#).)

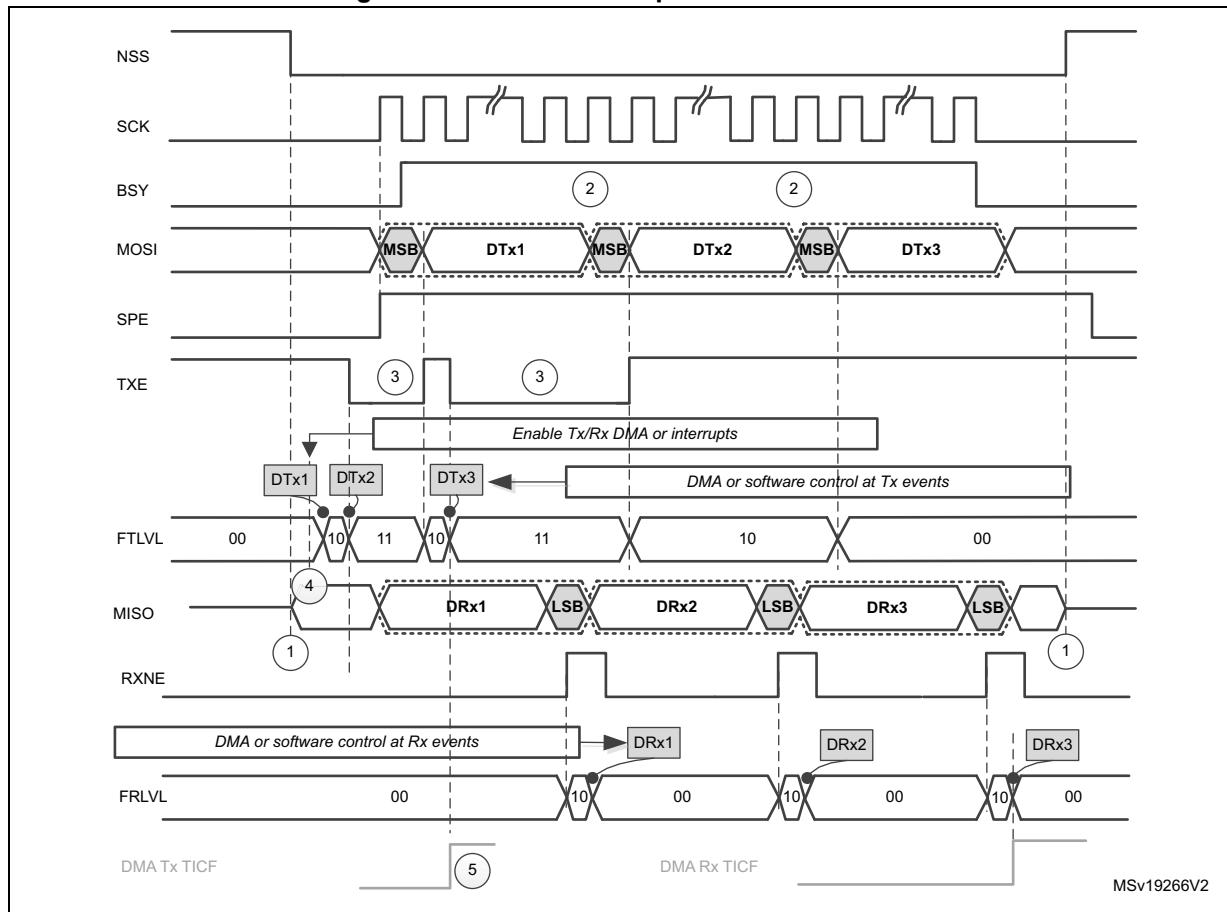
## Communication diagrams

Some typical timing schemes are explained in this section. These schemes are valid no matter if the SPI events are handled by polling, interrupts or DMA. For simplicity, the LSBFIRST=0, CPOL=0 and CPHA=1 setting is used as a common assumption here. No complete configuration of DMA streams is provided.

The following numbered notes are common for [Figure 364 on page 1091](#) through [Figure 367 on page 1094](#):

1. The slave starts to control MISO line as NSS is active and SPI is enabled, and is disconnected from the line when one of them is released. Sufficient time must be provided for the slave to prepare data dedicated to the master in advance before its transaction starts.  
At the master, the SPI peripheral takes control at MOSI and SCK signals (occasionally at NSS signal as well) only if SPI is enabled. If SPI is disabled the SPI peripheral is disconnected from GPIO logic, so the levels at these lines depends on GPIO setting exclusively.
2. At the master, BSY stays active between frames if the communication (clock signal) is continuous. At the slave, BSY signal always goes down for at least one clock cycle between data frames.
3. The TXE signal is cleared only if TXFIFO is full.
4. The DMA arbitration process starts just after the TXDMAEN bit is set. The TXE interrupt is generated just after the TXEIE is set. As the TXE signal is at an active level, data transfers to TxFIFO start, until TxFIFO becomes full or the DMA transfer completes.
5. If all the data to be sent can fit into TxFIFO, the DMA Tx TCIF flag can be raised even before communication on the SPI bus starts. This flag always rises before the SPI transaction is completed.
6. The CRC value for a package is calculated continuously frame by frame in the SPIx\_TXCRCR and SPIx\_RXCRCR registers. The CRC information is processed after the entire data package has completed, either automatically by DMA (Tx channel must be set to the number of data frames to be processed) or by SW (the user must handle CRCNEXT bit during the last data frame processing).  
While the CRC value calculated in SPIx\_TXCRCR is simply sent out by transmitter, received CRC information is loaded into Rx FIFO and then compared with the SPIx\_RXCRCR register content (CRC error flag can be raised here if any difference). This is why the user must take care to flush this information from the FIFO, either by software reading out all the stored content of Rx FIFO, or by DMA when the proper number of data frames is preset for Rx channel (number of data frames + number of CRC frames) (see the settings at the example assumption).
7. In data packed mode, TxE and RxNE events are paired and each read/write access to the FIFO is 16 bits wide until the number of data frames are even. If the Tx FIFO is  $\frac{3}{4}$  full FTLVL status stays at FIFO full level. That is why the last odd data frame cannot be stored before the Tx FIFO becomes  $\frac{1}{2}$  full. This frame is stored into Tx FIFO with an 8-bit access either by software or automatically by DMA when LDMA\_TX control is set.
8. To receive the last odd data frame in packed mode, the Rx threshold must be changed to 8-bit when the last data frame is processed, either by software setting FRXTH=1 or automatically by a DMA internal signal when LDMA\_RX is set.

Figure 364. Master full-duplex communication



Assumptions for master full-duplex communication example:

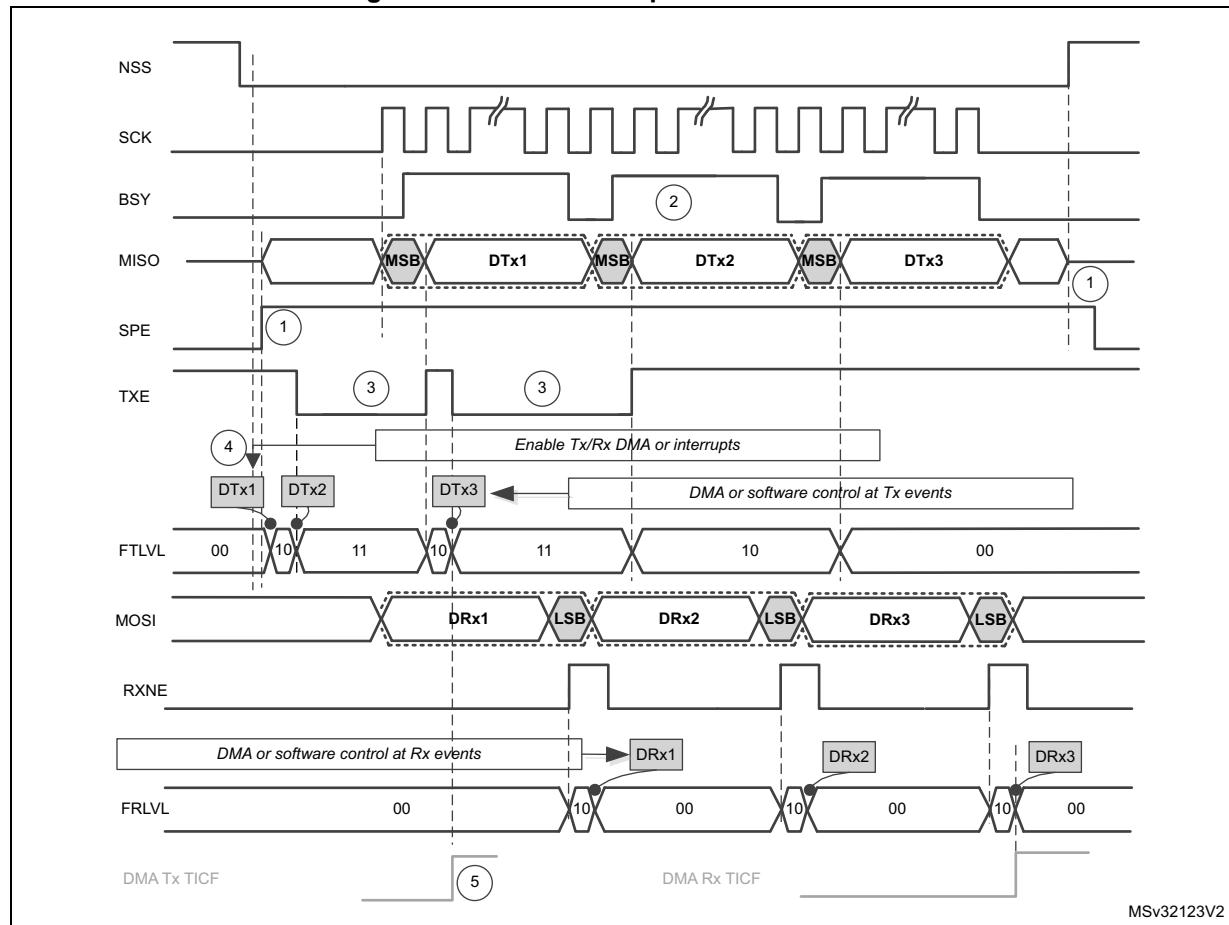
- Data size > 8 bit

If DMA is used:

- Number of Tx frames transacted by DMA is set to 3
- Number of Rx frames transacted by DMA is set to 3

See also : [Communication diagrams on page 1090](#) for details about common assumptions and notes.

Figure 365. Slave full-duplex communication



Assumptions for slave full-duplex communication example:

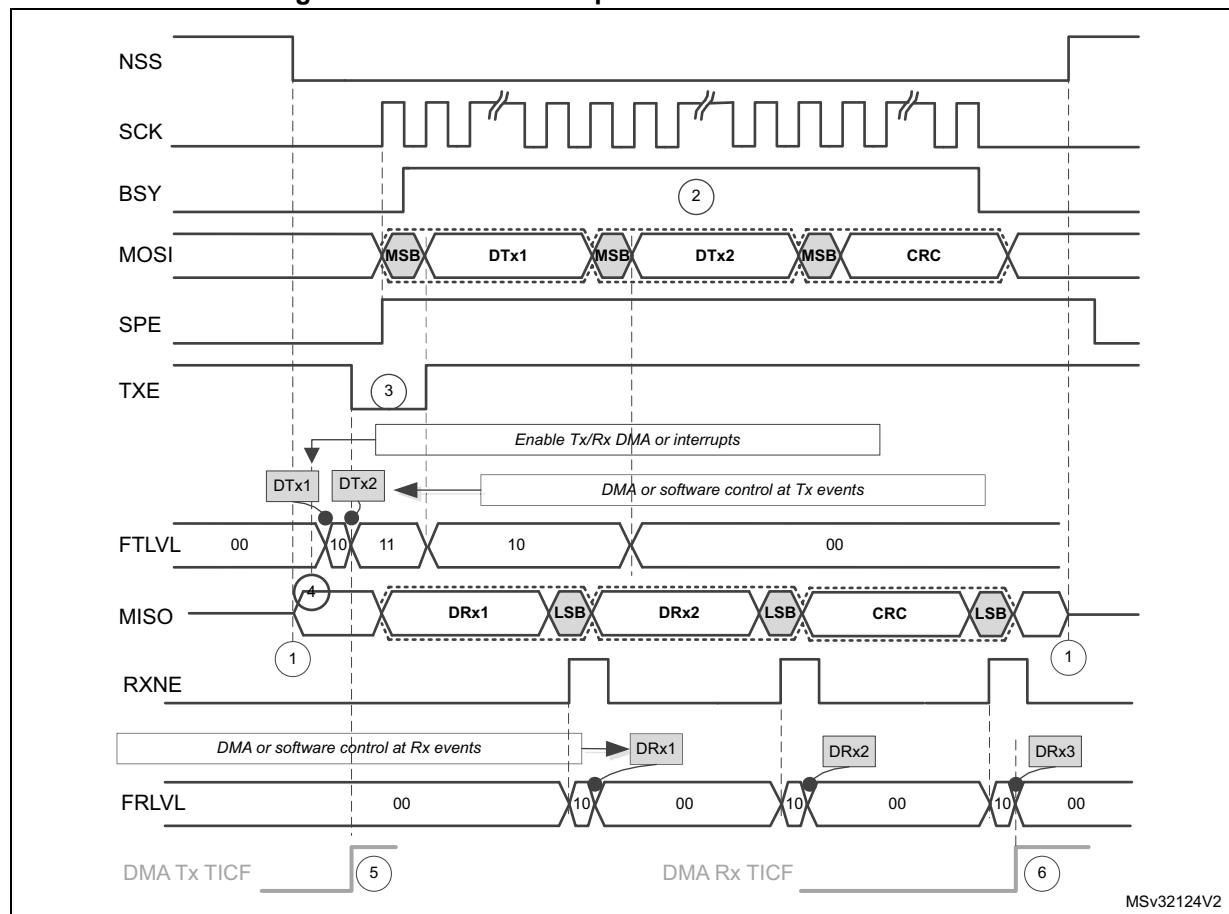
- Data size > 8 bit

If DMA is used:

- Number of Tx frames transacted by DMA is set to 3
- Number of Rx frames transacted by DMA is set to 3

See also [Communication diagrams on page 1090](#) for details about common assumptions and notes.

Figure 366. Master full-duplex communication with CRC



Assumptions for master full-duplex communication with CRC example:

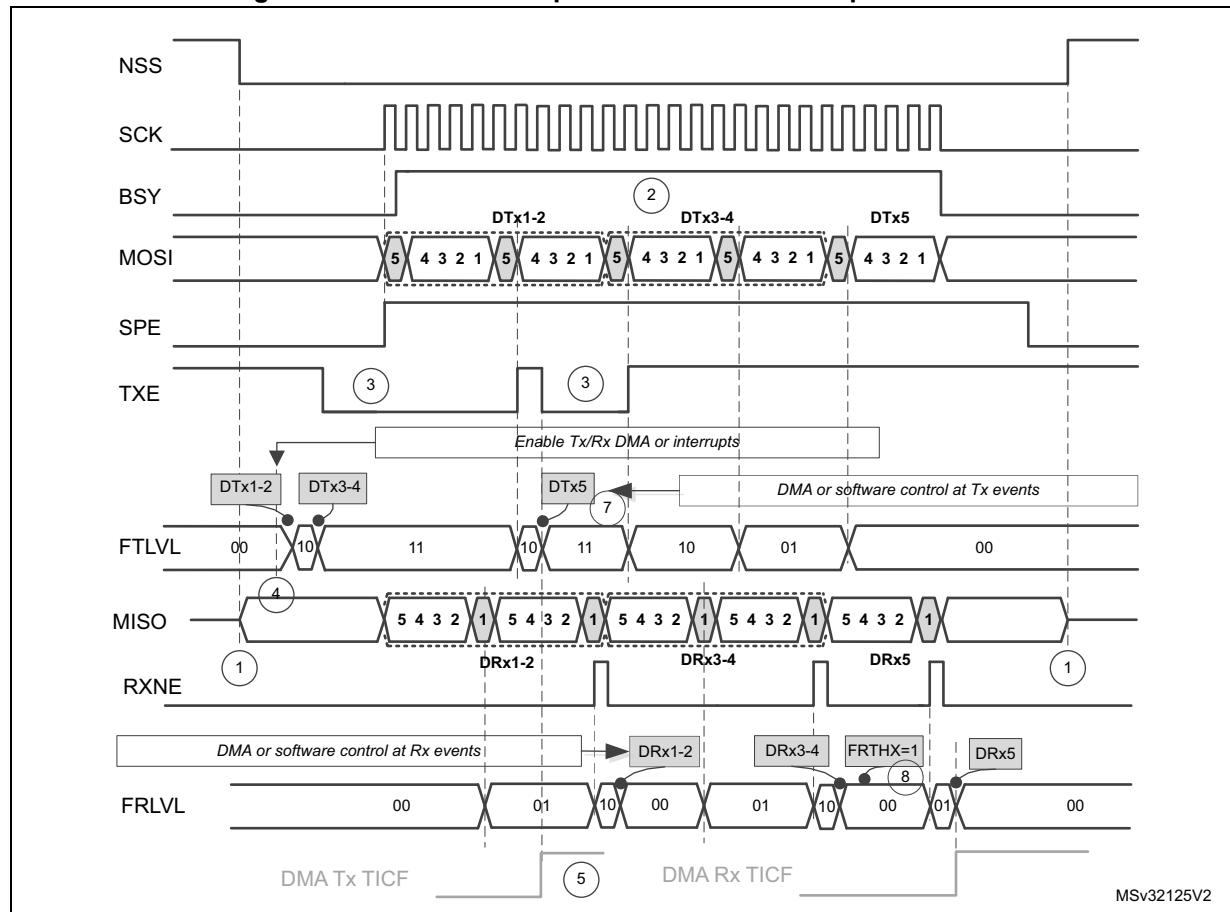
- Data size = 16 bit
- CRC enabled

If DMA is used:

- Number of Tx frames transacted by DMA is set to 2
- Number of Rx frames transacted by DMA is set to 3

See also : [Communication diagrams on page 1090](#) for details about common assumptions and notes.

Figure 367. Master full-duplex communication in packed mode



Assumptions for master full-duplex communication in packed mode example:

- Data size = 5 bit
- Read/write FIFO is performed mostly by 16-bit access
- FRXTH=0

If DMA is used:

- Number of Tx frames to be transacted by DMA is set to 3
- Number of Rx frames to be transacted by DMA is set to 3
- PSIZE for both Tx and Rx DMA channel is set to 16-bit
- LDMA\_TX=1 and LDMA\_RX=1

See also : [Communication diagrams on page 1090](#) for details about common assumptions and notes.

### 32.5.10 SPI status flags

Three status flags are provided for the application to completely monitor the state of the SPI bus.

#### Tx buffer empty flag (TXE)

The TXE flag is set when transmission TXFIFO has enough space to store data to send. TXE flag is linked to the TXFIFO level. The flag goes high and stays high until the TXFIFO level is lower or equal to 1/2 of the FIFO depth. An interrupt can be generated if the TXEIE bit in the SPIx\_CR2 register is set. The bit is cleared automatically when the TXFIFO level becomes greater than 1/2.

#### Rx buffer not empty (RXNE)

The RXNE flag is set depending on the FRXTH bit value in the SPIx\_CR2 register:

- If FRXTH is set, RXNE goes high and stays high until the RXFIFO level is greater or equal to 1/4 (8-bit).
- If FRXTH is cleared, RXNE goes high and stays high until the RXFIFO level is greater than or equal to 1/2 (16-bit).

An interrupt can be generated if the RXNEIE bit in the SPIx\_CR2 register is set.

The RXNE is cleared by hardware automatically when the above conditions are no longer true.

#### Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect).

When BSY is set, it indicates that a data transfer is in progress on the SPI (the SPI bus is busy).

The BSY flag can be used in certain modes to detect the end of a transfer so that the software can disable the SPI or its peripheral clock before entering a low-power mode which does not provide a clock for the peripheral. This avoids corrupting the last transfer.

The BSY flag is also useful for preventing write collisions in a multimaster system.

The BSY flag is cleared under any one of the following conditions:

- When the SPI is correctly disabled
- When a fault is detected in Master mode (MODF bit set to 1)
- In Master mode, when it finishes a data transmission and no new data is ready to be sent
- In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.

Note:

*When the next transmission can be handled immediately by the master (e.g. if the master is in Receive-only mode or its Transmit FIFO is not empty), communication is continuous and the BSY flag remains set to '1' between transfers on the master side. Although this is not the case with a slave, it is recommended to use always the TXE and RXNE flags (instead of the BSY flags) to handle data transmission or reception operations.*

### 32.5.11 SPI error flags

An SPI interrupt is generated if one of the following error flags is set and interrupt is enabled by setting the ERRIE bit.

#### Overrun flag (OVR)

An overrun condition occurs when data is received by a master or slave and the RXFIFO has not enough space to store this received data. This can happen if the software or the DMA did not have enough time to read the previously received data (stored in the RXFIFO) or when space for data storage is limited e.g. the RXFIFO is not available when CRC is enabled in receive only mode so in this case the reception buffer is limited into a single data frame buffer (see [Section 32.5.14: CRC calculation](#)).

When an overrun condition occurs, the newly received value does not overwrite the previous one in the RXFIFO. The newly received value is discarded and all data transmitted subsequently is lost. Clearing the OVR bit is done by a read access to the SPI\_DR register followed by a read access to the SPI\_SR register.

#### Mode fault (MODF)

Mode fault occurs when the master device has its internal NSS signal (NSS pin in NSS hardware mode, or SSI bit in NSS software mode) pulled low. This automatically sets the MODF bit. Master mode fault affects the SPI interface in the following ways:

- The MODF bit is set and an SPI interrupt is generated if the ERRIE bit is set.
- The SPE bit is cleared. This blocks all output from the device and disables the SPI interface.
- The MSTR bit is cleared, thus forcing the device into slave mode.

Use the following software sequence to clear the MODF bit:

1. Make a read or write access to the SPIx\_SR register while the MODF bit is set.
2. Then write to the SPIx\_CR1 register.

To avoid any multiple slave conflicts in a system comprising several MCUs, the NSS pin must be pulled high during the MODF bit clearing sequence. The SPE and MSTR bits can be restored to their original state after this clearing sequence. As a security, hardware does not allow the SPE and MSTR bits to be set while the MODF bit is set. In a slave device the MODF bit cannot be set except as the result of a previous multimaster conflict.

#### CRC error (CRCERR)

This flag is used to verify the validity of the value received when the CRCEN bit in the SPIx\_CR1 register is set. The CRCERR flag in the SPIx\_SR register is set if the value received in the shift register does not match the receiver SPIx\_RXCRCR value. The flag is cleared by the software.

#### TI mode frame format error (FRE)

A TI mode frame format error is detected when an NSS pulse occurs during an ongoing communication when the SPI is operating in slave mode and configured to conform to the TI mode protocol. When this error occurs, the FRE flag is set in the SPIx\_SR register. The SPI is not disabled when an error occurs, the NSS pulse is ignored, and the SPI waits for the next NSS pulse before starting a new transfer. The data may be corrupted since the error detection may result in the loss of two data bytes.

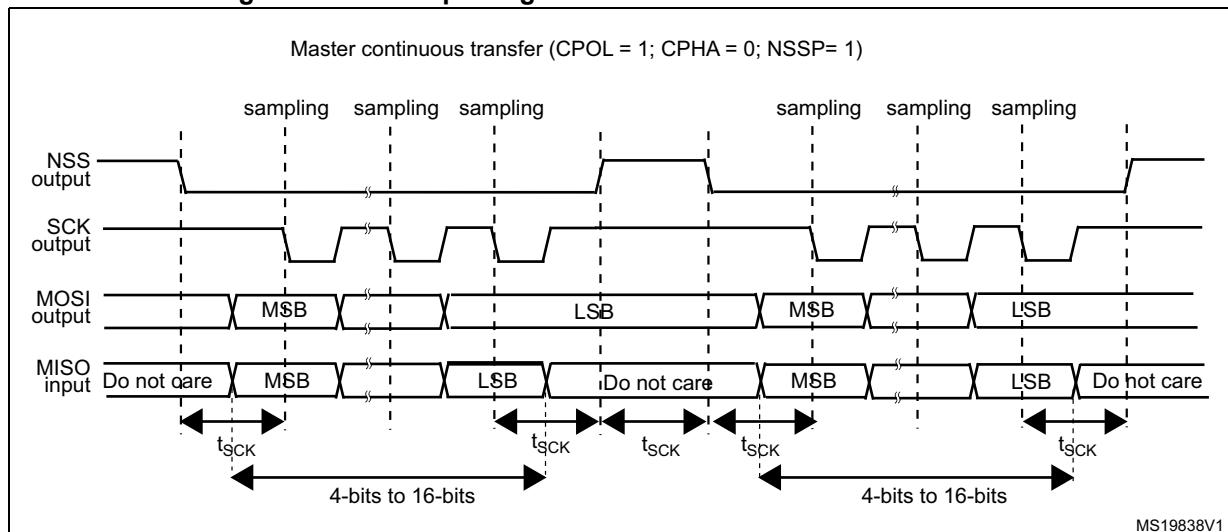
The FRE flag is cleared when SPIx\_SR register is read. If the ERRIE bit is set, an interrupt is generated on the NSS error detection. In this case, the SPI should be disabled because data consistency is no longer guaranteed and communications should be reinitiated by the master when the slave SPI is enabled again.

### 32.5.12 NSS pulse mode

This mode is activated by the NSSP bit in the SPIx\_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx\_CR1 CPHA = 0, CPOL setting is ignored). When activated, an NSS pulse is generated between two consecutive data frame transfers when NSS stays at high level for the duration of one clock period at least. This mode allows the slave to latch data. NSSP pulse mode is designed for applications with a single master-slave pair.

*Figure 368* illustrates NSS pin management when NSSP pulse mode is enabled.

**Figure 368. NSSP pulse generation in Motorola SPI master mode**



**Note:** Similar behavior is encountered when CPOL = 0. In this case the sampling edge is the *rising* edge of SCK, and NSS assertion and deassertion refer to this sampling edge.

### 32.5.13 TI mode

#### TI protocol in master mode

The SPI interface is compatible with the TI protocol. The FRF bit of the SPIx\_CR2 register can be used to configure the SPI to be compliant with this protocol.

The clock polarity and phase are forced to conform to the TI protocol requirements whatever the values set in the SPIx\_CR1 register. NSS management is also specific to the TI protocol which makes the configuration of NSS management through the SPIx\_CR1 and SPIx\_CR2 registers (SSM, SSI, SSOE) impossible in this case.

In slave mode, the SPI baud rate prescaler is used to control the moment when the MISO pin state changes to HiZ when the current transaction finishes (see *Figure 369*). Any baud rate can be used, making it possible to determine this moment with optimal flexibility. However, the baud rate is generally set to the external master clock baud rate. The delay for the MISO signal to become HiZ ( $t_{release}$ ) depends on internal resynchronization and on the

baud rate value set in through the BR[2:0] bits in the SPIx\_CR1 register. It is given by the formula:

$$\frac{t_{\text{baud\_rate}}}{2} + 4 \times t_{\text{pclk}} < t_{\text{release}} < \frac{t_{\text{baud\_rate}}}{2} + 6 \times t_{\text{pclk}}$$

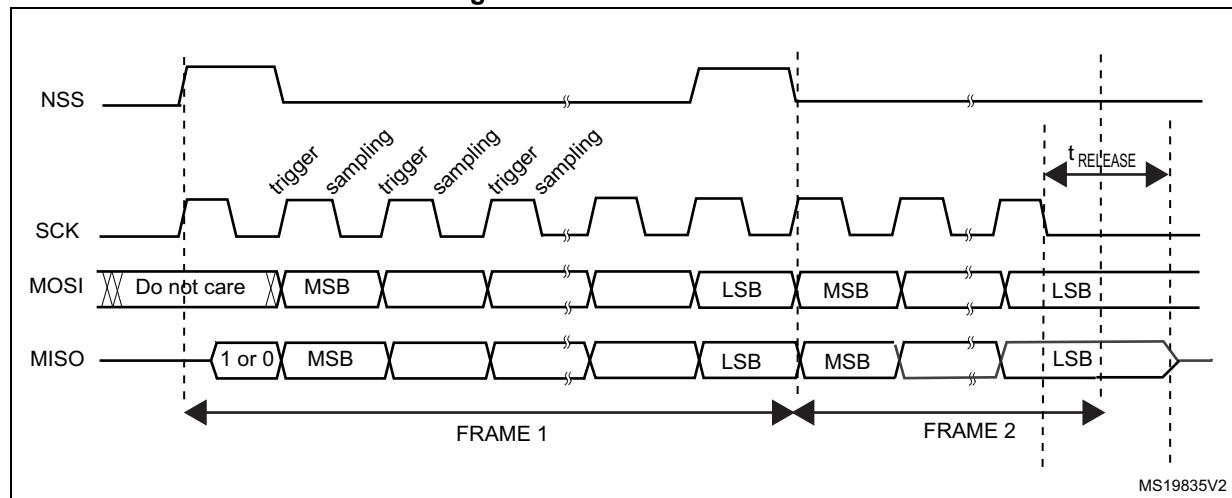
If the slave detects a misplaced NSS pulse during a data frame transaction the TIFRE flag is set.

If the data size is equal to 4-bits or 5-bits, the master in full-duplex mode or transmit-only mode uses a protocol with one more dummy data bit added after LSB. TI NSS pulse is generated above this dummy bit clock cycle instead of the LSB in each period.

This feature is not available for Motorola SPI communications (FRF bit set to 0).

*Figure 369: TI mode transfer* shows the SPI communication waveforms when TI mode is selected.

Figure 369. TI mode transfer



### 32.5.14 CRC calculation

Two separate CRC calculators are implemented in order to check the reliability of transmitted and received data. The SPI offers CRC8 or CRC16 calculation independently of the frame data length, which can be fixed to 8-bit or 16-bit. For all the other data frame lengths, no CRC is available.

#### CRC principle

CRC calculation is enabled by setting the CRCEN bit in the SPIx\_CR1 register before the SPI is enabled (SPE = 1). The CRC value is calculated using an odd programmable polynomial on each bit. The calculation is processed on the sampling clock edge defined by the CPHA and CPOL bits in the SPIx\_CR1 register. The calculated CRC value is checked automatically at the end of the data block as well as for transfer managed by CPU or by the DMA. When a mismatch is detected between the CRC calculated internally on the received data and the CRC sent by the transmitter, a CRCERR flag is set to indicate a data corruption error. The right procedure for handling the CRC calculation depends on the SPI configuration and the chosen transfer management.

**Note:** *The polynomial value should only be odd. No even values are supported.*

### CRC transfer managed by CPU

Communication starts and continues normally until the last data frame has to be sent or received in the SPIx\_DR register. Then CRCNEXT bit has to be set in the SPIx\_CR1 register to indicate that the CRC frame transaction follows after the transaction of the currently processed data frame. The CRCNEXT bit must be set before the end of the last data frame transaction. CRC calculation is frozen during CRC transaction.

The received CRC is stored in the RXFIFO like a data byte or word. That is why in CRC mode only, the reception buffer has to be considered as a single 16-bit buffer used to receive only one data frame at a time.

A CRC-format transaction usually takes one more data frame to communicate at the end of data sequence. However, when setting an 8-bit data frame checked by 16-bit CRC, two more frames are necessary to send the complete CRC.

When the last CRC data is received, an automatic check is performed comparing the received value and the value in the SPIx\_RXCRC register. Software has to check the CRCERR flag in the SPIx\_SR register to determine if the data transfers were corrupted or not. Software clears the CRCERR flag by writing '0' to it.

After the CRC reception, the CRC value is stored in the RXFIFO and must be read in the SPIx\_DR register in order to clear the RXNE flag.

### CRC transfer managed by DMA

When SPI communication is enabled with CRC communication and DMA mode, the transmission and reception of the CRC at the end of communication is automatic (with the exception of reading CRC data in receive only mode). The CRCNEXT bit does not have to be handled by the software. The counter for the SPI transmission DMA channel has to be set to the number of data frames to transmit excluding the CRC frame. On the receiver side, the received CRC value is handled automatically by DMA at the end of the transaction but user must take care to flush out received CRC information from RXFIFO as it is always loaded into it. In full-duplex mode, the counter of the reception DMA channel can be set to the number of data frames to receive including the CRC, which means, for example, in the specific case of an 8-bit data frame checked by 16-bit CRC:

$$\text{DMA\_RX} = \text{Numb\_of\_data} + 2$$

In receive only mode, the DMA reception channel counter should contain only the amount of data transferred, excluding the CRC calculation. Then based on the complete transfer from DMA, all the CRC values must be read back by software from FIFO as it works as a single buffer in this mode.

At the end of the data and CRC transfers, the CRCERR flag in the SPIx\_SR register is set if corruption occurred during the transfer.

If packing mode is used, the LDMA\_RX bit needs managing if the number of data is odd.

### Resetting the SPIx\_TXCRC and SPIx\_RXCRC values

The SPIx\_TXCRC and SPIx\_RXCRC values are cleared automatically when new data is sampled after a CRC phase. This allows the use of DMA circular mode (not available in receive-only mode) in order to transfer data without any interruption, (several data blocks covered by intermediate CRC checking phases).

If the SPI is disabled during a communication the following sequence must be followed:

1. Disable the SPI
2. Clear the CRCEN bit
3. Enable the CRCEN bit
4. Enable the SPI

Note:

*When the SPI interface is configured as a slave, the NSS internal signal needs to be kept low during transaction of the CRC phase once the CRCNEXT signal is released. That is why the CRC calculation cannot be used at NSS Pulse mode when NSS hardware mode should be applied at slave normally.*

*At TI mode, despite the fact that clock phase and clock polarity setting is fixed and independent on SPIx\_CR1 register, the corresponding setting CPOL=0 CPHA=1 has to be kept at the SPIx\_CR1 register anyway if CRC is applied. In addition, the CRC calculation has to be reset between sessions by SPI disable sequence with re-enable the CRCEN bit described above at both master and slave side, else CRC calculation can be corrupted at this specific mode.*

## 32.6 SPI interrupts

During SPI communication an interrupt can be generated by the following events:

- Transmit TXFIFO ready to be loaded
- Data received in Receive RXFIFO
- Master mode fault
- Overrun error
- TI frame format error
- CRC protocol error

Interrupts can be enabled and disabled separately.

**Table 208. SPI interrupt requests**

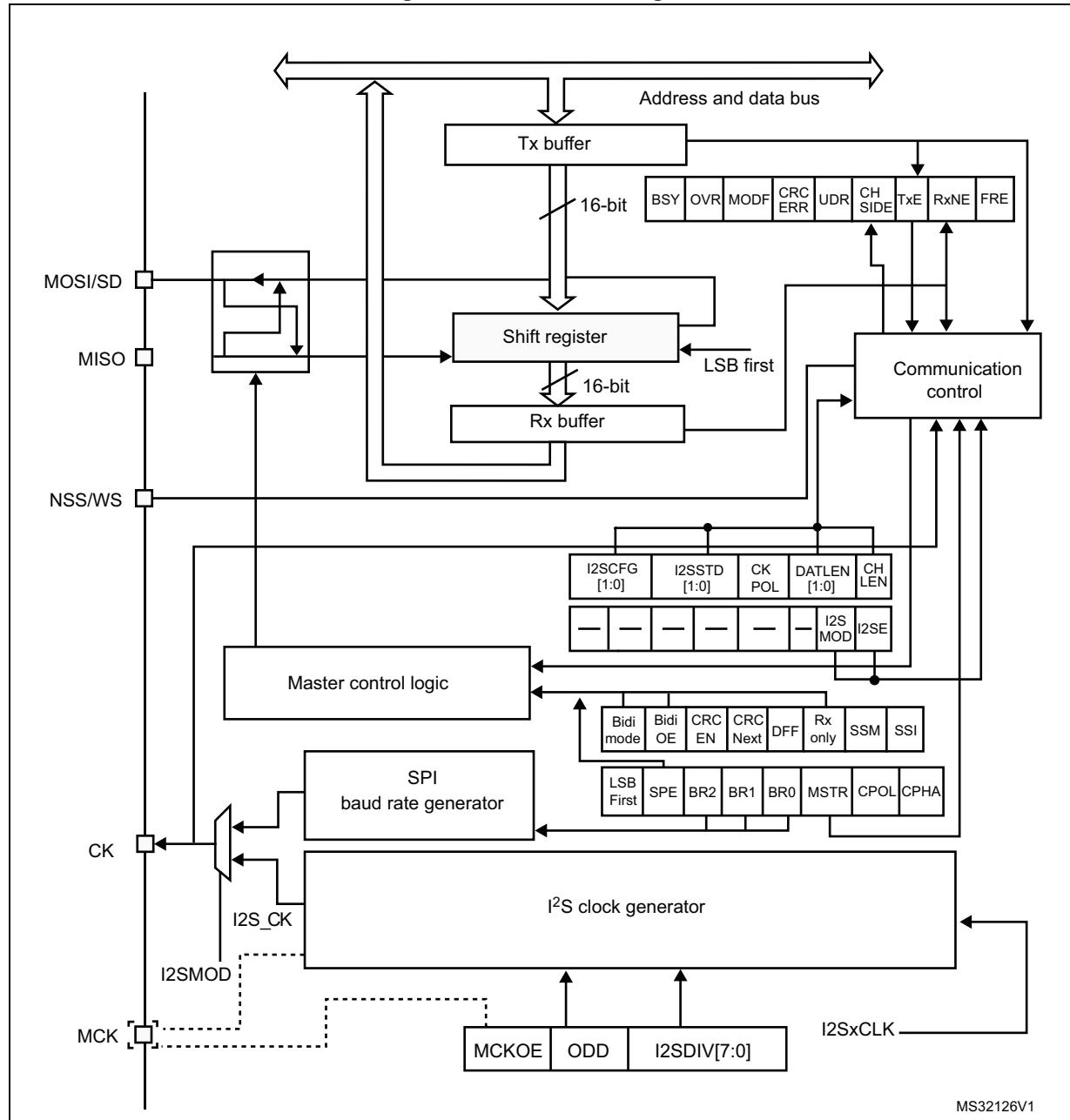
Interrupt event	Event flag	Enable Control bit
Transmit TXFIFO ready to be loaded	TXE	TXEIE
Data received in RXFIFO	RXNE	RXNEIE
Master Mode fault event	MODF	ERRIE
Overrun error	OVR	
TI frame format error	FRE	
CRC protocol error	CRCERR	

## 32.7 I2S functional description

### 32.7.1 I2S general description

The block diagram of the I2S is shown in [Figure 370](#).

**Figure 370. I2S block diagram**



1. MCK is mapped on the MISO pin.

The SPI can function as an audio I2S interface when the I2S capability is enabled (by setting the I2SMOD bit in the SPIx\_I2SCFGR register). This interface mainly uses the same pins, flags and interrupts as the SPI.

The I2S shares three common pins with the SPI:

- SD: Serial Data (mapped on the MOSI pin) to transmit or receive the two time-multiplexed data channels (in half-duplex mode only).
- WS: Word Select (mapped on the NSS pin) is the data control signal output in master mode and input in slave mode.
- CK: Serial Clock (mapped on the SCK pin) is the serial clock output in master mode and serial clock input in slave mode.

An additional pin can be used when a master clock output is needed for some external audio devices:

- MCK: Master Clock (mapped separately) is used, when the I2S is configured in master mode (and when the MCKOE bit in the SPIx\_I2SPR register is set), to output this additional clock generated at a preconfigured frequency rate equal to  $256 \times f_S$  for all I2S modes, and to  $128 \times f_S$  for all PCM modes, where  $f_S$  is the audio sampling frequency.

The I2S uses its own clock generator to produce the communication clock when it is set in master mode. This clock generator is also the source of the master clock output. Two additional registers are available in I<sup>2</sup>S mode. One is linked to the clock generator configuration SPIx\_I2SPR and the other one is a generic I2S configuration register SPIx\_I2SCFGR (audio standard, slave/master mode, data format, packet frame, clock polarity, etc.).

The SPIx\_CR1 register and all CRC registers are not used in the 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 I2S uses the same SPI register for data transfer (SPIx\_DR) in 16-bit wide mode.

### 32.7.2 I2S full duplex

*Figure 371* shows how to perform full-duplex communications using two SPI/I2S instances. In this case, the WS and CK IOs of both SPI/I2S must be connected together.

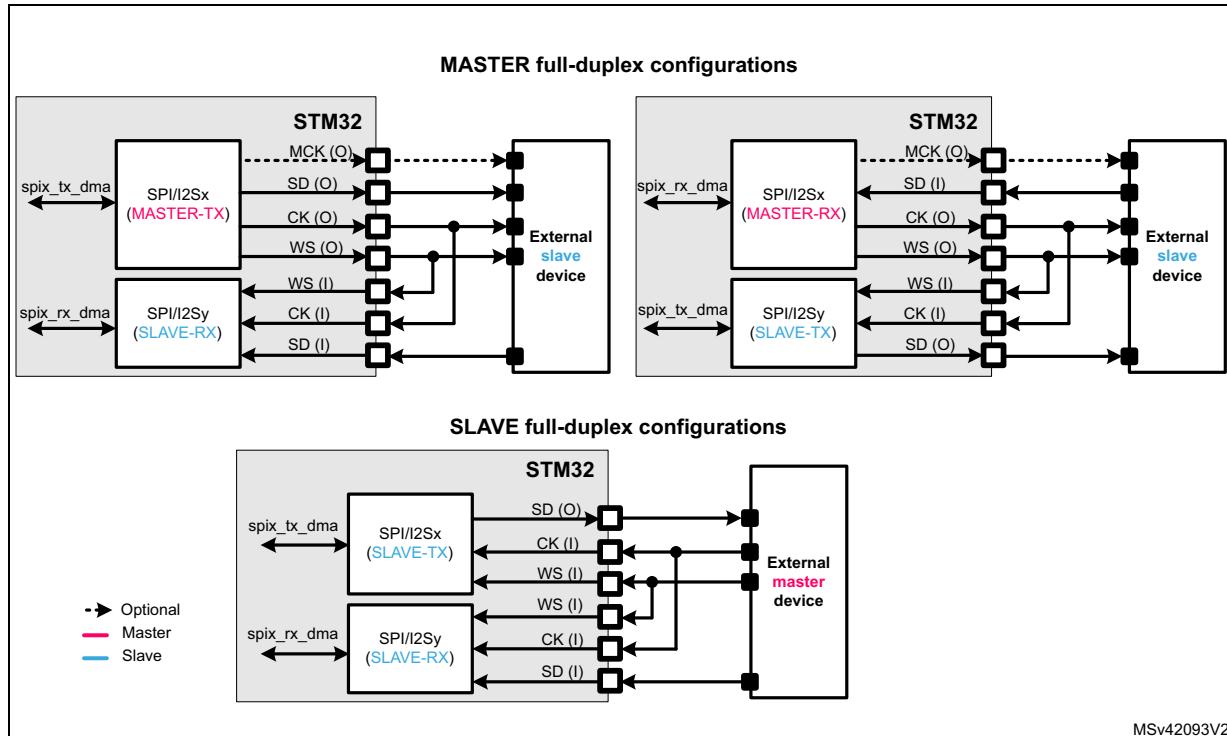
For the master full-duplex mode, one of the SPI/I2S block must be programmed in master (I2SCFG = '10' or '11'), and the other SPI/I2S block must be programmed in slave (I2SCFG = '00' or '01'). The MCK can be generated or not, depending on the application needs.

For the slave full-duplex mode, both SPI/I2S blocks must be programmed in slave. One of them in the slave receiver (I2SCFG = '01'), and the other in the slave transmitter (I2SCFG = '00'). The master external device then provides the bit clock (CK) and the frame synchronization (WS).

Note that the full-duplex mode can be used for all the supported standards: I<sup>2</sup>S Philips, MSB-justified, LSB-justified and PCM.

For the full-duplex mode, both SPI/I2S instances must use the same standard, with the same parameters: I2SMOD, I2SSTD, CKPOL, PCMSYNC, DATLEN and CHLEN must contain the same value on both instances.

Figure 371. Full-duplex communication



### 32.7.3 Supported audio protocols

The three-line bus has to handle only audio data generally time-multiplexed on two channels: the right channel and the left channel. However there is only one 16-bit register for transmission or reception. So, it is up to the software to write into the data register the appropriate value corresponding to each channel side, or to read the data from the data register and to identify the corresponding channel by checking the CHSIDE bit in the SPIx\_SR register. Channel left is always sent first followed by the channel right (CHSIDE has no meaning for the PCM protocol).

Four data and packet frames are available. Data may be sent with a format of:

- 16-bit data packed in a 16-bit frame
- 16-bit data packed in a 32-bit frame
- 24-bit data packed in a 32-bit frame
- 32-bit data packed in a 32-bit frame

When using 16-bit data extended on 32-bit packet, the first 16 bits (MSB) are the significant bits, the 16-bit LSB is forced to 0 without any need for software action or DMA request (only one read/write operation).

The 24-bit and 32-bit data frames need two CPU read or write operations to/from the SPIx\_DR register or two DMA operations if the DMA is preferred for the application. For 24-bit data frame specifically, the 8 non-significant bits are extended to 32 bits with 0-bits (by hardware).

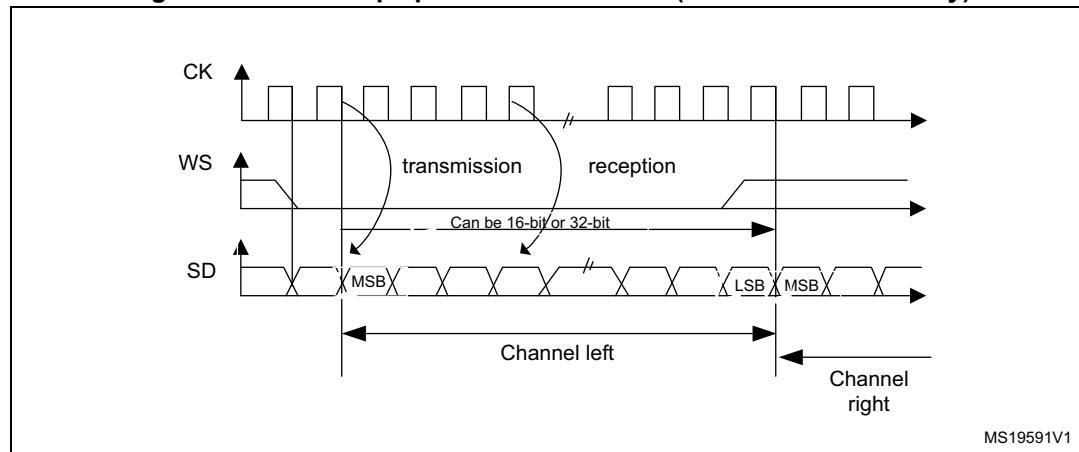
For all data formats and communication standards, the most significant bit is always sent first (MSB first).

The I<sup>2</sup>S interface supports four audio standards, configurable using the I2SSSTD[1:0] and PCMSYNC bits in the SPIx\_I2SCFGR register.

### I<sup>2</sup>S Philips standard

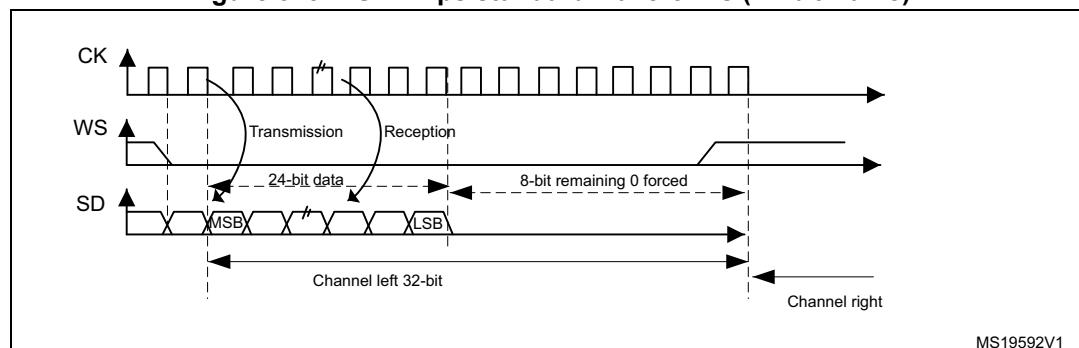
For this standard, the WS signal is used to indicate which channel is being transmitted. It is activated one CK clock cycle before the first bit (MSB) is available.

**Figure 372. I<sup>2</sup>S Philips protocol waveforms (16/32-bit full accuracy)**



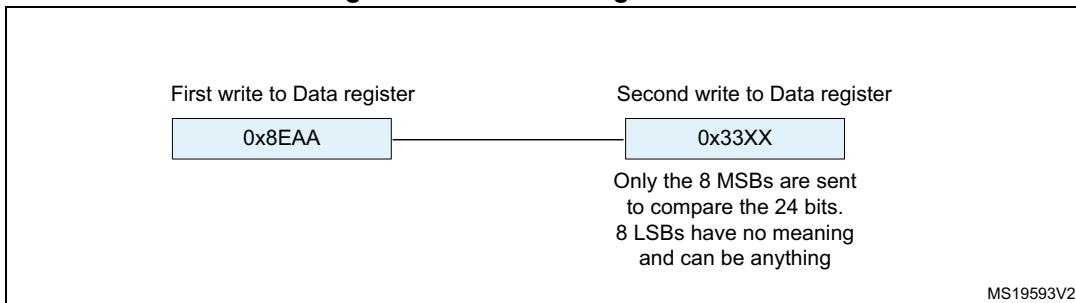
Data are latched on the falling edge of CK (for the transmitter) and are read on the rising edge (for the receiver). The WS signal is also latched on the falling edge of CK.

**Figure 373. I<sup>2</sup>S Philips standard waveforms (24-bit frame)**

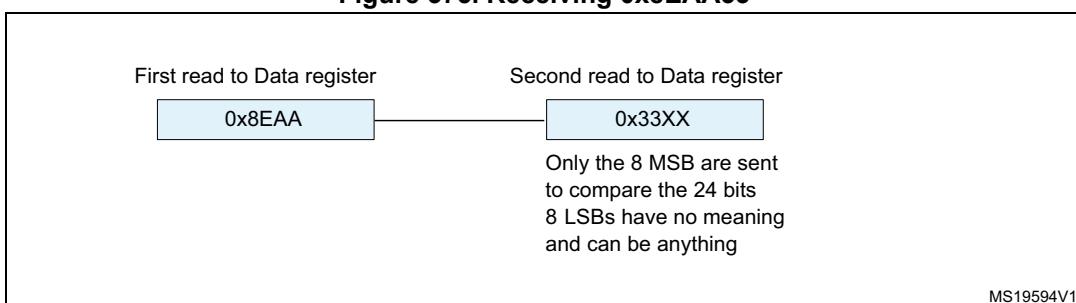
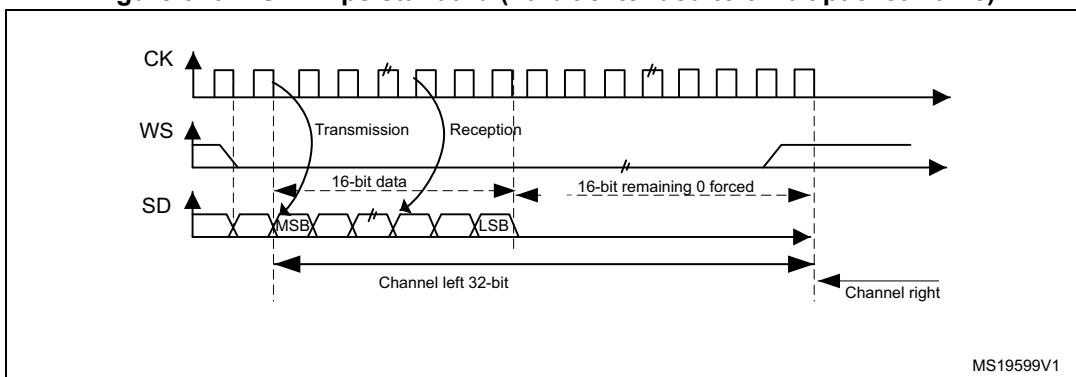


This mode needs two write or read operations to/from the SPIx\_DR register.

- In transmission mode:  
If 0x8EAA33 has to be sent (24-bit):

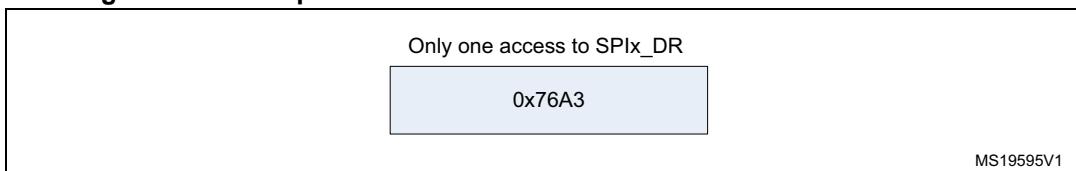
**Figure 374. Transmitting 0x8EAA33**

- In reception mode:  
If data 0x8EAA33 is received:

**Figure 375. Receiving 0x8EAA33****Figure 376. I<sup>2</sup>S Philips standard (16-bit extended to 32-bit packet frame)**

When 16-bit data frame extended to 32-bit channel frame is selected during the I2S configuration phase, only one access to the SPIx\_DR register is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format.

If the data to transmit or the received data are 0x76A3 (0x76A30000 extended to 32-bit), the operation shown in [Figure 377](#) is required.

**Figure 377. 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.

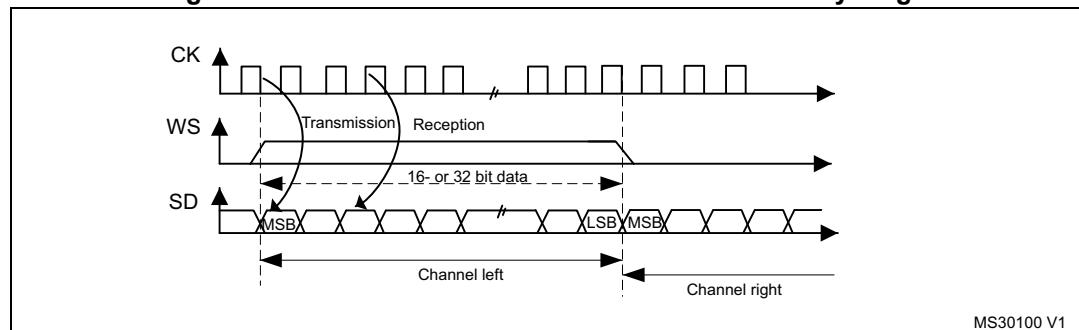
For reception, the RXNE flag is set and its interrupt, if allowed, is generated when the first 16 MSB half-word is received.

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

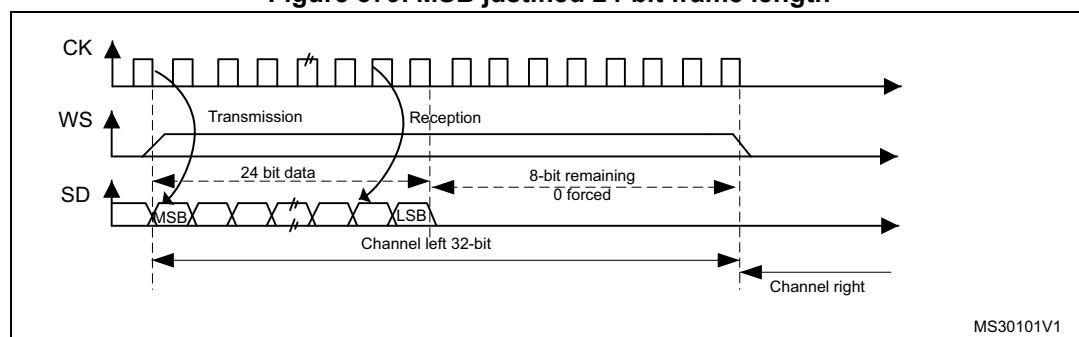
For this standard, the WS signal is generated at the same time as the first data bit, which is the MSBit.

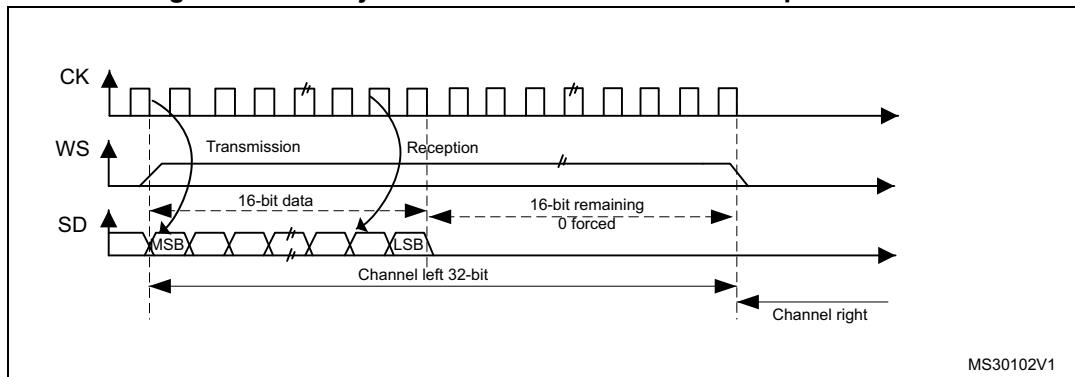
**Figure 378. MSB Justified 16-bit or 32-bit full-accuracy length**



Data are latched on the falling edge of CK (for transmitter) and are read on the rising edge (for the receiver).

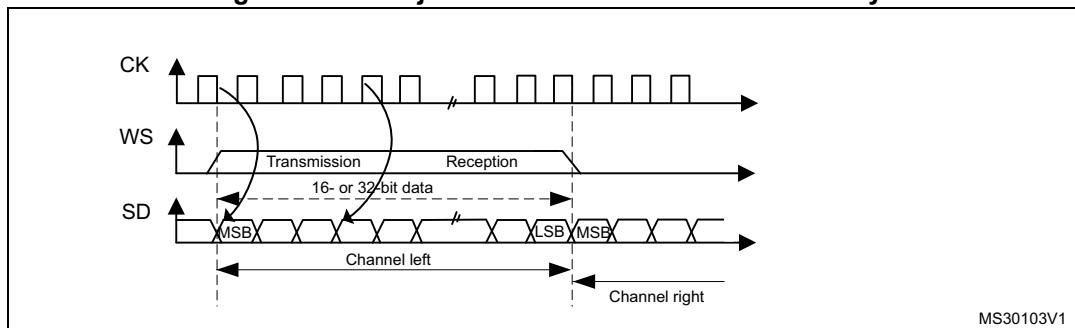
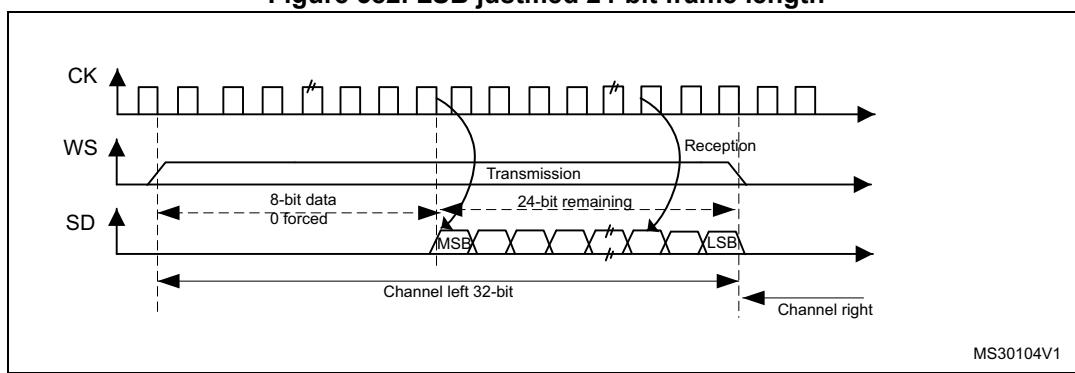
**Figure 379. MSB justified 24-bit frame length**



**Figure 380. MSB justified 16-bit extended to 32-bit packet frame****LSB justified standard**

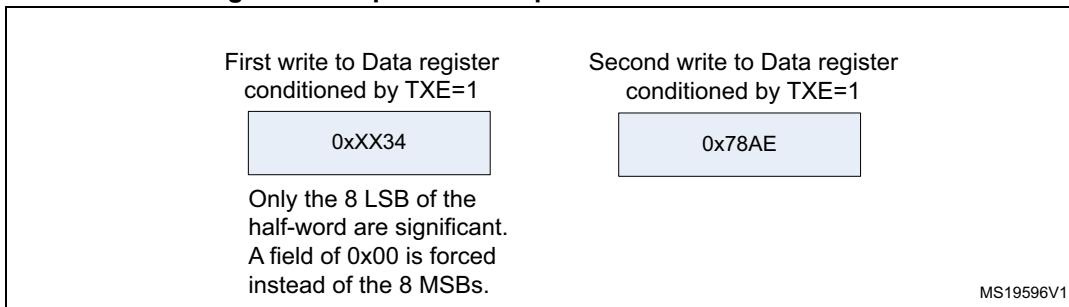
This standard is similar to the MSB justified standard (no difference for the 16-bit and 32-bit full-accuracy frame formats).

The sampling of the input and output signals is the same as for the I<sup>2</sup>S Philips standard.

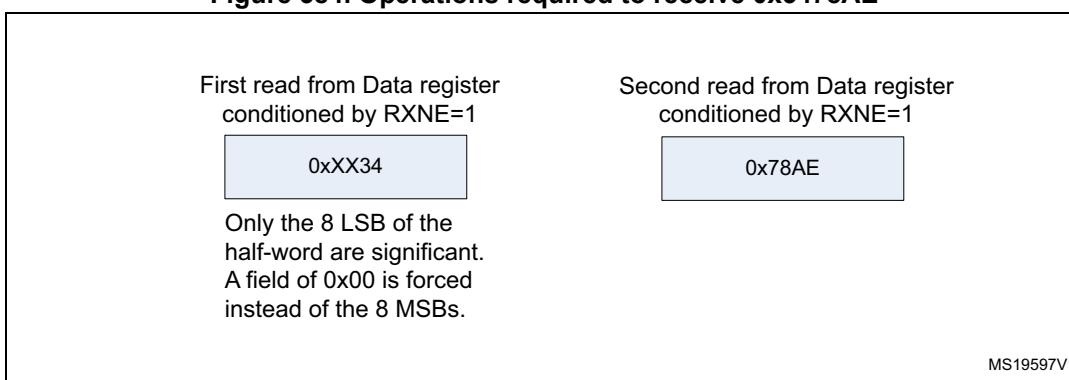
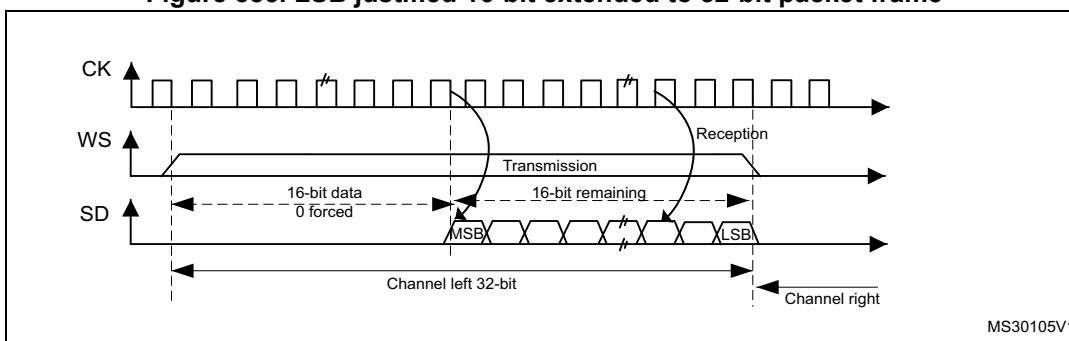
**Figure 381. LSB justified 16-bit or 32-bit full-accuracy****Figure 382. LSB justified 24-bit frame length**

- In transmission mode:

If data 0x3478AE have to be transmitted, two write operations to the SPIx\_DR register are required by software or by DMA. The operations are shown below.

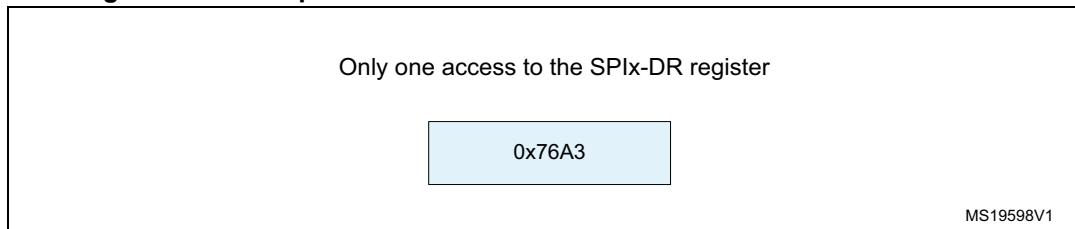
**Figure 383. Operations required to transmit 0x3478AE**

- In reception mode:  
If data 0x3478AE are received, two successive read operations from the SPIx\_DR register are required on each RXNE event.

**Figure 384. Operations required to receive 0x3478AE****Figure 385. LSB justified 16-bit extended to 32-bit packet frame**

When 16-bit data frame extended to 32-bit channel frame is selected during the I2S configuration phase, Only one access to the SPIx\_DR register is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format. In this case it corresponds to the half-word MSB.

If the data to transmit or the received data are 0x76A3 (0x0000 76A3 extended to 32-bit), the operation shown in [Figure 386](#) is required.

**Figure 386. Example of 16-bit data frame extended to 32-bit channel frame**

In transmission mode, when a TXE event occurs, the application has to write the data to be transmitted (in this case 0x76A3). The 0x000 field is transmitted first (extension on 32-bit). The TXE flag is set again as soon as the effective data (0x76A3) is sent on SD.

In reception mode, RXNE is asserted as soon as the significant half-word is received (and not the 0x0000 field).

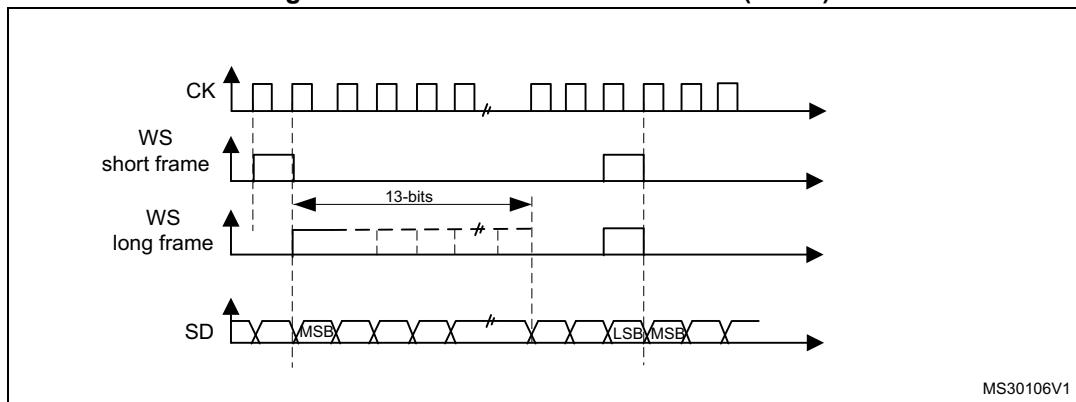
In this way, more time is provided between two write or read operations to prevent underrun or overrun conditions.

### PCM standard

For the PCM standard, there is no need to use channel-side information. The two PCM modes (short and long frame) are available and configurable using the PCMSYNC bit in SPIx\_I2SCFGR register.

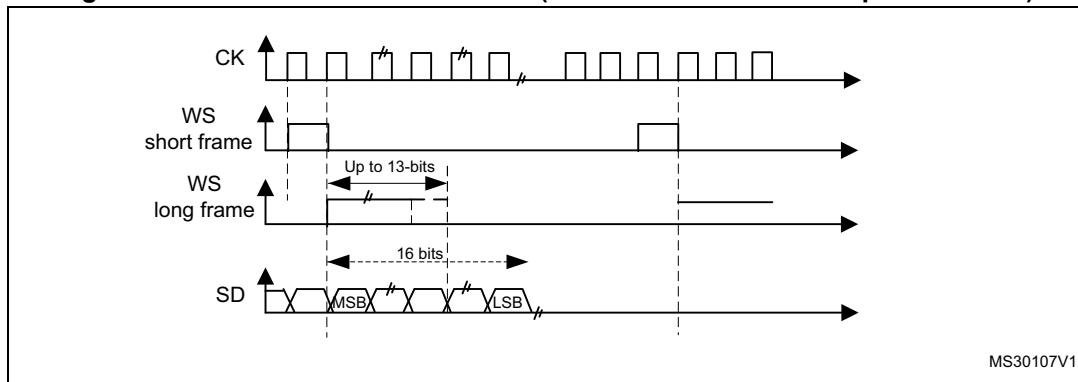
In PCM mode, the output signals (WS, SD) are sampled on the rising edge of CK signal. The input signals (WS, SD) are captured on the falling edge of CK.

Note that CK and WS are configured as output in MASTER mode.

**Figure 387. 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.

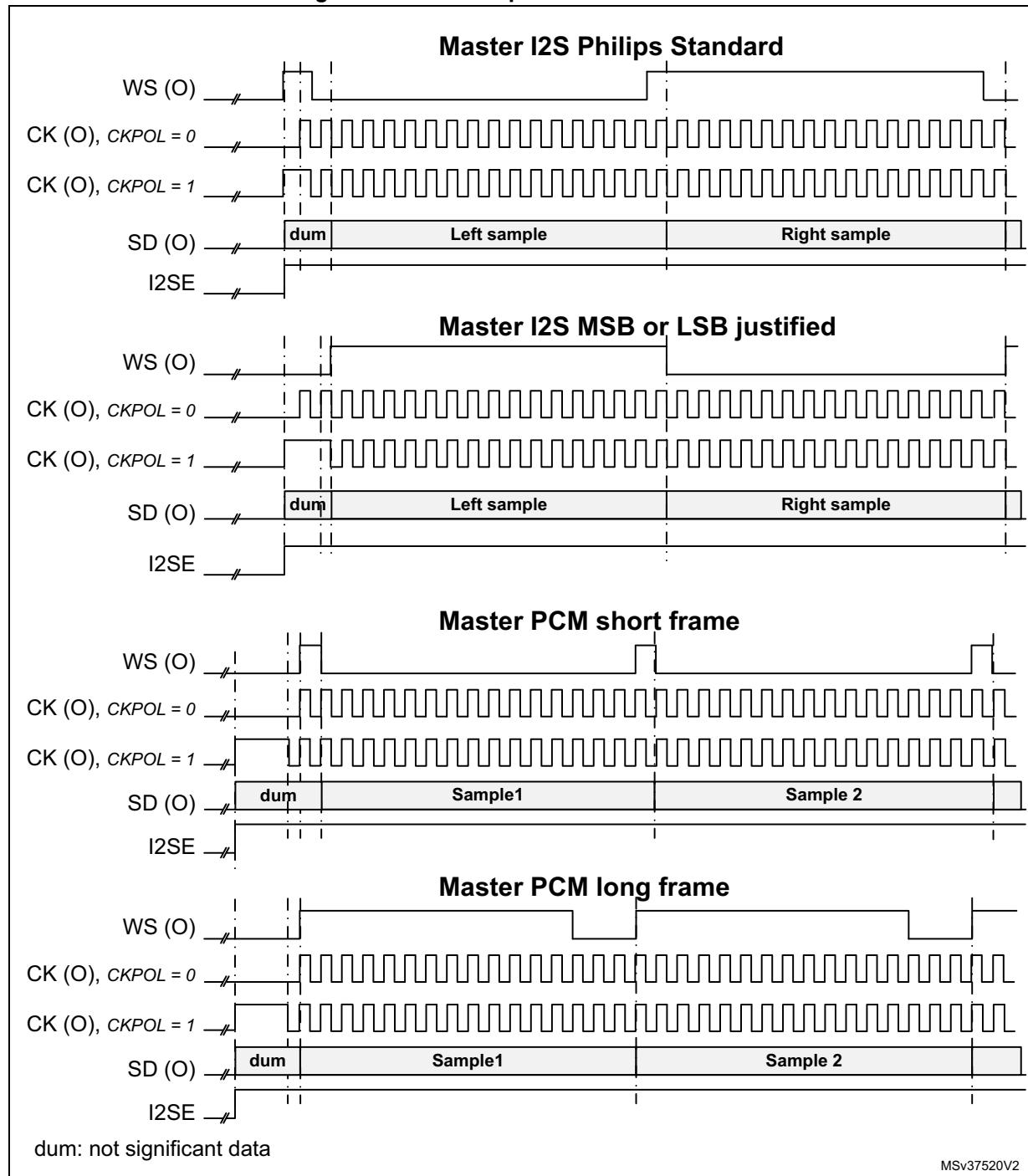
**Figure 388. PCM standard waveforms (16-bit extended to 32-bit packet frame)**

**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.

### 32.7.4 Start-up description

The [Figure 389](#) shows how the serial interface is handled in MASTER mode, when the SPI/I2S is enabled (via I2SE bit). It shows as well the effect of CKPOL on the generated signals.

Figure 389. Start sequence in master mode



In slave mode, the way the frame synchronization is detected, depends on the value of ASTRTEN bit.

If ASTRTEN = 0, when the audio interface is enabled (I2SE = 1), then the hardware waits for the appropriate transition on the incoming WS signal, using the CK signal.

The appropriate transition is a falling edge on WS signal when I<sup>2</sup>S Philips Standard is used, or a rising edge for other standards. The falling edge is detected by sampling first WS to 1 and then to 0, and vice-versa for the rising edge detection.

If ASTRTEN = 1, the user has to enable the audio interface before the WS becomes active. This means that the I2SE bit must be set to 1 when WS = 1 for I<sup>2</sup>S Philips standard, or when WS = 0 for other standards.

### 32.7.5 Clock generator

The I<sup>2</sup>S bit rate determines the data flow on the I<sup>2</sup>S data line and the I<sup>2</sup>S clock signal frequency.

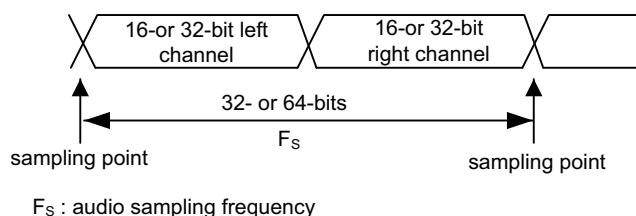
I<sup>2</sup>S bit rate = number of bits per channel × number of channels × sampling audio frequency

For a 16-bit audio, left and right channel, the I<sup>2</sup>S bit rate is calculated as follows:

$$\text{I}^2\text{S bit rate} = 16 \times 2 \times f_s$$

It is: I<sup>2</sup>S bit rate = 32 × 2 × f<sub>s</sub> if the packet length is 32-bit wide.

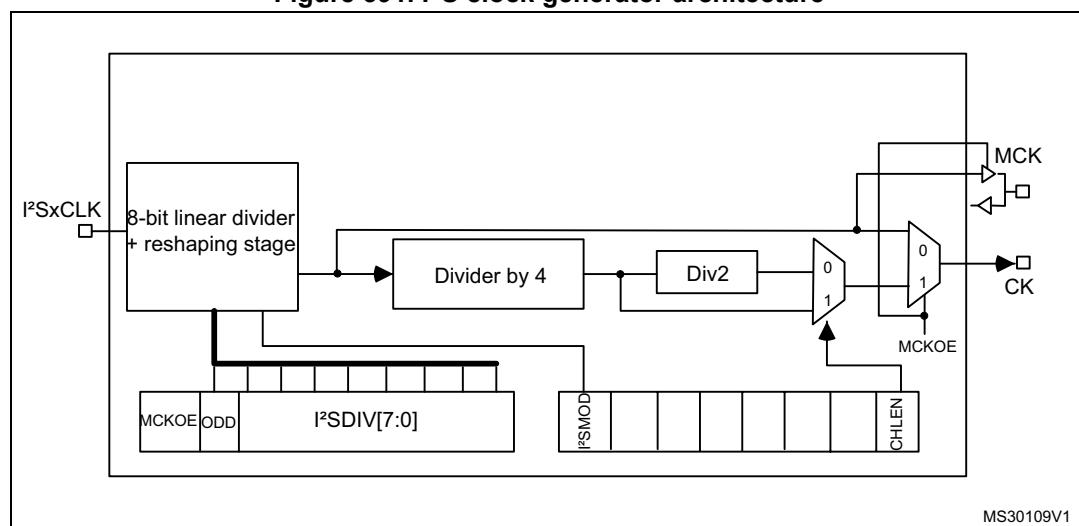
**Figure 390. Audio sampling frequency definition**



MS30108V1

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 391. I<sup>2</sup>S clock generator architecture**



MS30109V1

1. Where x can be 2 or 3.

*Figure 391* presents the communication clock architecture. The I2SxCLK clock is provided by the reset and clock controller (RCC) of the product. The I2SxCLK clock can be asynchronous with respect to the SPI/I2S APB clock.

---

**Warning:** In addition, it is mandatory to keep the I2SxCLK frequency higher or equal to the APB clock used by the SPI/I2S block. If this condition is not respected the SPI/I2S does not work properly.

---

The audio sampling frequency may be 192 kHz, 96 kHz, 48 kHz, 44.1 kHz, 32 kHz, 22.05 kHz, 16 kHz, 11.025 kHz or 8 kHz (or any other value within this range).

In order to reach the desired frequency, the linear divider needs to be programmed according to the formulas below:

#### For I<sup>2</sup>S modes:

When the master clock is generated (MCKOE in the SPIx\_I2SPR register is set):

$$F_S = \frac{F_{I2SxCLK}}{256 \times ((2 \times I2SDIV) + ODD)}$$

When the master clock is disabled (MCKOE bit cleared):

$$F_S = \frac{F_{I2SxCLK}}{32 \times (CHLEN + 1) \times ((2 \times I2SDIV) + ODD)}$$

CHLEN = 0 when the channel frame is 16-bit wide and,

CHLEN = 1 when the channel frame is 32-bit wide.

#### For PCM modes:

When the master clock is generated (MCKOE in the SPIx\_I2SPR register is set):

$$F_S = \frac{F_{I2SxCLK}}{128 \times ((2 \times I2SDIV) + ODD)}$$

When the master clock is disabled (MCKOE bit cleared):

$$F_S = \frac{F_{I2SxCLK}}{16 \times (CHLEN + 1) \times ((2 \times I2SDIV) + ODD)}$$

CHLEN = 0 when the channel frame is 16-bit wide and,

CHLEN = 1 when the channel frame is 32-bit wide.

Where  $F_S$  is the audio sampling frequency, and  $F_{I2SxCLK}$  is the frequency of the kernel clock provided to the SPI/I2S block.

**Note:** I<sub>2</sub>S DIV must be strictly higher than 1.

The following table provides example precision values for different clock configurations.

**Note:** Other configurations are possible that allow optimum clock precision.

**Table 209. Audio-frequency precision using 48 MHz clock derived from HSE<sup>(1)</sup>**

SYSCLK (MHz)	Data length	I <sub>2</sub> S DIV	I <sub>2</sub> S ODD	MCLK	Target fs (Hz)	Real fs (kHz)	Error
48	16	8	0	No	96000	93750	2.3438%
48	32	4	0	No	96000	93750	2.3438%
48	16	15	1	No	48000	48387.0968	0.8065%
48	32	8	0	No	48000	46875	2.3438%
48	16	17	0	No	44100	44117.647	0.0400%
48	32	8	1	No	44100	44117.647	0.0400%
48	16	23	1	No	32000	31914.8936	0.2660%
48	32	11	1	No	32000	32608.696	1.9022%
48	16	34	0	No	22050	22058.8235	0.0400%
48	32	17	0	No	22050	22058.8235	0.0400%
48	16	47	0	No	16000	15957.4468	0.2660%
48	32	23	1	No	16000	15957.447	0.2660%
48	16	68	0	No	11025	11029.4118	0.0400%
48	32	34	0	No	11025	11029.412	0.0400%
48	16	94	0	No	8000	7978.7234	0.2660%
48	32	47	0	No	8000	7978.7234	0.2660%
48	16	2	0	Yes	48000	46875	2.3430%
48	32	2	0	Yes	48000	46875	2.3430%
48	16	2	0	Yes	44100	46875	6.2925%
48	32	2	0	Yes	44100	46875	6.2925%
48	16	3	0	Yes	32000	31250	2.3438%
48	32	3	0	Yes	32000	31250	2.3438%
48	16	4	1	Yes	22050	20833.333	5.5178%
48	32	4	1	Yes	22050	20833.333	5.5178%
48	16	6	0	Yes	16000	15625	2.3438%
48	32	6	0	Yes	16000	15625	2.3438%
48	16	8	1	Yes	11025	11029.4118	0.0400%
48	32	8	1	Yes	11025	11029.4118	0.0400%
48	16	11	1	Yes	8000	8152.17391	1.9022%
48	32	11	1	Yes	8000	8152.17391	1.9022%

1. This table gives only example values for different clock configurations. Other configurations allowing optimum clock precision are possible.

### 32.7.6 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 DAC/ADC audio component (the I2SDIV and ODD values should be computed depending on the state of the MCK output, for more details refer to [Section 32.7.5: 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 TXIE bit in the SPIx\_CR2 register is set.

For more details about the write operations depending on the I<sup>2</sup>S standard mode selected, refer to [Section 32.7.3: Supported audio protocols](#)).

To ensure a continuous audio data transmission, it is mandatory to write the SPIx\_DR register with the next data to transmit before the end of the current transmission.

To switch off the I<sup>2</sup>S, by clearing I2SE, it is mandatory to wait for TXE = 1 and BSY = 0.

### Reception sequence

The operating mode is the same as for transmission mode except for the point 3 (refer to the procedure described in [Section 32.7.6: I<sup>2</sup>S master mode](#)), where the configuration should set the master reception mode through the I2SCFG[1:0] bits.

Whatever the data or channel length, the audio data are received by 16-bit packets. This means that each time the Rx buffer is full, the RXNE flag is set and an interrupt is generated if the RXNEIE bit is set in SPIx\_CR2 register. Depending on the data and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the Rx buffer.

Clearing the RXNE bit is performed by reading the SPIx\_DR register.

CHSIDE is updated after each reception. It is sensitive to the WS signal generated by the I<sup>2</sup>S cell.

For more details about the read operations depending on the I<sup>2</sup>S standard mode selected, refer to [Section 32.7.3: Supported audio protocols](#).

If data are received while the previously received data have not been read yet, an overrun is generated and the OVR flag is set. If the ERRIE bit is set in the SPIx\_CR2 register, an interrupt is generated to indicate the error.

To switch off the I<sup>2</sup>S, specific actions are required to ensure that the I<sup>2</sup>S completes the transfer cycle properly without initiating a new data transfer. The sequence depends on the configuration of the data and channel lengths, and on the audio protocol mode selected. In the case of:

- 16-bit data length extended on 32-bit channel length (DATLEN = 00 and CHLEN = 1) using the LSB justified mode (I2SSTD = 10)
  - a) Wait for the second to last RXNE = 1 (n – 1)
  - b) Then wait 17 I<sup>2</sup>S clock cycles (using a software loop)
  - c) Disable the I<sup>2</sup>S (I2SE = 0)
- 16-bit data length extended on 32-bit channel length (DATLEN = 00 and CHLEN = 1) in MSB justified, I<sup>2</sup>S or PCM modes (I2SSTD = 00, I2SSTD = 01 or I2SSTD = 11, respectively)
  - a) Wait for the last RXNE
  - b) Then wait 1 I<sup>2</sup>S clock cycle (using a software loop)
  - c) Disable the I<sup>2</sup>S (I2SE = 0)
- For all other combinations of DATLEN and CHLEN, whatever the audio mode selected through the I2SSTD bits, carry out the following sequence to switch off the I<sup>2</sup>S:
  - a) Wait for the second to last RXNE = 1 (n – 1)
  - b) Then wait one I<sup>2</sup>S clock cycle (using a software loop)
  - c) Disable the I<sup>2</sup>S (I2SE = 0)

*Note:* The BSY flag is kept low during transfers.

### 32.7.7 I<sup>2</sup>S slave mode

For the slave configuration, the I<sup>2</sup>S can be configured in transmission or reception mode. The operating mode is following mainly the same rules as described for the I<sup>2</sup>S master

configuration. In slave mode, there is no clock to be generated by the I2S interface. The clock and WS signals are input from the external master connected to the I2S interface. There is then no need, for the user, to configure the clock.

The configuration steps to follow are listed below:

1. Set the I2SMOD bit in the SPIx\_I2SCFGR register to select I<sup>2</sup>S mode and choose the I<sup>2</sup>S standard through the I2SSTD[1:0] bits, the data length through the DATLEN[1:0] bits and the number of bits per channel for the frame configuring the CHLEN bit. Select also the mode (transmission or reception) for the slave through the I2SCFG[1:0] bits in SPIx\_I2SCFGR register.
2. If needed, select all the potential interrupt sources and the DMA capabilities by writing the SPIx\_CR2 register.
3. The I2SE bit in SPIx\_I2SCFGR register must be set.

### Transmission sequence

The transmission sequence begins when the external master device sends the clock and when the NSS\_WS signal requests the transfer of data. The slave has to be enabled before the external master starts the communication. The I2S data register has to be loaded before the master initiates the communication.

For the I2S, MSB justified and LSB justified modes, the first data item to be written into the data register corresponds to the data for the left channel. When the communication starts, the data are transferred from the Tx buffer to the shift register. The TXE flag is then set in order to request the right channel data to be written into the I2S data register.

The CHSIDE flag indicates which channel is to be transmitted. Compared to the master transmission mode, in slave mode, CHSIDE is sensitive to the WS signal coming from the external master. This means that the slave needs to be ready to transmit the first data before the clock is generated by the master. WS assertion corresponds to left channel transmitted first.

*Note:* *The I2SE has to be written at least two PCLK cycles before the first clock of the master comes on the CK line.*

The data half-word is parallel-loaded into the 16-bit shift register (from the internal bus) during the first bit transmission, and then shifted out serially to the MOSI/SD pin MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPIx\_CR2 register is set.

Note that the TXE flag should be checked to be at 1 before attempting to write the Tx buffer.

For more details about the write operations depending on the I<sup>2</sup>S standard mode selected, refer to [Section 32.7.3: Supported audio protocols](#).

To secure a continuous audio data transmission, it is mandatory to write the SPIx\_DR register with the next data to transmit before the end of the current transmission. An underrun flag is set and an interrupt may be generated if the data are not written into the SPIx\_DR register before the first clock edge of the next data communication. This indicates to the software that the transferred data are wrong. If the ERRIE bit is set into the SPIx\_CR2 register, an interrupt is generated when the UDR flag in the SPIx\_SR register goes high. In this case, it is mandatory to switch off the I2S and to restart a data transfer starting from the left channel.

To switch off the I2S, by clearing the I2SE bit, it is mandatory to wait for TXE = 1 and BSY = 0.

### Reception sequence

The operating mode is the same as for the transmission mode except for the point 1 (refer to the procedure described in [Section 32.7.7: I<sup>2</sup>S slave mode](#)), where the configuration should set the master reception mode using the I2SCFG[1:0] bits in the SPIx\_I2SCFGR register.

Whatever the data length or the channel length, the audio data are received by 16-bit packets. This means that each time the RX buffer is full, the RXNE flag in the SPIx\_SR register is set and an interrupt is generated if the RXNEIE bit is set in the SPIx\_CR2 register. Depending on the data length and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the RX buffer.

The CHSIDE flag is updated each time data are received to be read from the SPIx\_DR register. It is sensitive to the external WS line managed by the external master component.

Clearing the RXNE bit is performed by reading the SPIx\_DR register.

For more details about the read operations depending the I<sup>2</sup>S standard mode selected, refer to [Section 32.7.3: Supported audio protocols](#).

If data are received while the preceding received data have not yet been read, an overrun is generated and the OVR flag is set. If the bit ERRIE is set in the SPIx\_CR2 register, an interrupt is generated to indicate the error.

To switch off the I2S in reception mode, I2SE has to be cleared immediately after receiving the last RXNE = 1.

*Note:* *The external master components should have the capability of sending/receiving data in 16-bit or 32-bit packets via an audio channel.*

## 32.7.8 I2S status flags

Three status flags are provided for the application to fully monitor the state of the I2S bus.

### Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect). It indicates the state of the communication layer of the I2S.

When BSY is set, it indicates that the I2S is busy communicating. There is one exception in master receive mode (I2SCFG = 11) where the BSY flag is kept low during reception.

The BSY flag is useful to detect the end of a transfer if the software needs to disable the I2S. This avoids corrupting the last transfer. For this, the procedure described below must be strictly respected.

The BSY flag is set when a transfer starts, except when the I2S is in master receiver mode.

The BSY flag is cleared:

- When a transfer completes (except in master transmit mode, in which the communication is supposed to be continuous)
- When the I2S is disabled

When communication is continuous:

- In master transmit mode, the BSY flag is kept high during all the transfers
- In slave mode, the BSY flag goes low for one I2S clock cycle between each transfer

*Note:* *Do not use the BSY flag to handle each data transmission or reception. It is better to use the TXE and RXNE flags instead.*

### Tx buffer empty flag (TXE)

When set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can then be loaded into it. The TXE flag is reset when the Tx buffer already contains data to be transmitted. It is also reset when the I2S is disabled (I2SE bit is reset).

### RX buffer not empty (RXNE)

When set, this flag indicates that there are valid received data in the RX Buffer. It is reset when SPIx\_DR register is read.

### Channel Side flag (CHSIDE)

In transmission mode, this flag is refreshed when TXE goes high. It indicates the channel side to which the data to transfer on SD has to belong. In case of an underrun error event in slave transmission mode, this flag is not reliable and I2S needs to be switched off and switched on before resuming the communication.

In reception mode, this flag is refreshed when data are received into SPIx\_DR. It indicates from which channel side data have been received. Note that in case of error (like OVR) this flag becomes meaningless and the I2S should be reset by disabling and then enabling it (with configuration if it needs changing).

This flag has no meaning in the PCM standard (for both Short and Long frame modes).

When the OVR or UDR flag in the SPIx\_SR is set and the ERRIE bit in SPIx\_CR2 is also set, an interrupt is generated. This interrupt can be cleared by reading the SPIx\_SR status register (once the interrupt source has been cleared).

## 32.7.9 I2S error flags

There are three error flags for the I2S cell.

### Underrun flag (UDR)

In slave transmission mode this flag is set when the first clock for data transmission appears while the software has not yet loaded any value into SPIx\_DR. It is available when the I2SMOD bit in the SPIx\_I2SCFGR register is set. An interrupt may be generated if the ERRIE bit in the SPIx\_CR2 register is set.

The UDR bit is cleared by a read operation on the SPIx\_SR register.

### Overrun flag (OVR)

This flag is set when data are received and the previous data have not yet been read from the SPIx\_DR register. As a result, the incoming data are lost. An interrupt may be generated if the ERRIE bit is set in the SPIx\_CR2 register.

In this case, the receive buffer contents are not updated with the newly received data from the transmitter device. A read operation to the SPIx\_DR register returns the previous correctly received data. All other subsequently transmitted half-words are lost.

Clearing the OVR bit is done by a read operation on the SPIx\_DR register followed by a read access to the SPIx\_SR register.

### Frame error flag (FRE)

This flag can be set by hardware only if the I2S is configured in Slave mode. It is set if the external master is changing the WS line while the slave is not expecting this change. If the

synchronization is lost, the following steps are required to recover from this state and resynchronize the external master device with the I2S slave device:

1. Disable the I2S.
2. Re-enable the I2S interface again (Keeping ASTRTEN=0).

Desynchronization between master and slave devices may be due to noisy environment on the CK communication clock or on the WS frame synchronization line. An error interrupt can be generated if the ERRIE bit is set. The desynchronization flag (FRE) is cleared by software when the status register is read.

### 32.7.10 DMA features

In I<sup>2</sup>S mode, the DMA works in exactly the same way as it does in SPI mode. There is no difference except that the CRC feature is not available in I<sup>2</sup>S mode since there is no data transfer protection system.

## 32.8 I2S interrupts

*Table 210* provides the list of I2S interrupts.

**Table 210. I2S interrupt requests**

Interrupt event	Event flag	Enable control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Overrun error	OVR	ERRIE
Underrun error	UDR	
Frame error flag	FRE	

## 32.9 SPI and I2S registers

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit). SPI\_DR in addition can be accessed by 8-bit access.

### 32.9.1 SPI control register 1 (SPIx\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDIOE	CRC EN	CRCN EXT	CRCL	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **BIDI MODE**: Bidirectional data mode enable.

This bit enables half-duplex communication using common single bidirectional data line.  
Keep RXONLY bit clear when bidirectional mode is active.

0: 2-line unidirectional data mode selected

1: 1-line bidirectional data mode selected

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bit 14 **BIDIOE**: Output enable in bidirectional mode

This bit combined with the BIDI MODE bit selects the direction of transfer in bidirectional mode.  
0: Output disabled (receive-only mode)  
1: Output enabled (transmit-only mode)

*Note: In master mode, the MOSI pin is used and in slave mode, the MISO pin is used.  
This bit is not used in I<sup>2</sup>S mode.*

Bit 13 **CRCEN**: Hardware CRC calculation enable

0: CRC calculation disabled  
1: CRC calculation enabled

*Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation.  
This bit is not used in I<sup>2</sup>S mode.*

Bit 12 **CRCNEXT**: Transmit CRC next

0: Next transmit value is from Tx buffer.  
1: Next transmit value is from Tx CRC register.

*Note: This bit has to be written as soon as the last data is written in the SPIx\_DR register.  
This bit is not used in I<sup>2</sup>S mode.*

Bit 11 **CRCL**: CRC length

This bit is set and cleared by software to select the CRC length.  
0: 8-bit CRC length  
1: 16-bit CRC length

*Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation.  
This bit is not used in I<sup>2</sup>S mode.*

Bit 10 **RXONLY:** Receive only mode enabled.

This bit enables simplex communication using a single unidirectional line to receive data exclusively. Keep BIDIMODE bit clear when receive only mode is active. This bit is also useful in a multislave system in which this particular slave is not accessed, the output from the accessed slave is not corrupted.

- 0: Full-duplex (Transmit and receive)
- 1: Output disabled (Receive-only mode)

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bit 9 **SSM:** Software slave management

When the SSM bit is set, the NSS pin input is replaced with the value from the SSI bit.

- 0: Software slave management disabled
- 1: Software slave management enabled

*Note: This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 8 **SSI:** Internal slave select

This bit has an effect only when the SSM bit is set. The value of this bit is forced onto the NSS pin and the I/O value of the NSS pin is ignored.

*Note: This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 7 **LSBFIRST:** Frame format

- 0: data is transmitted / received with the MSB first
- 1: data is transmitted / received with the LSB first

*Note: 1. This bit should not be changed when communication is ongoing.*

*2. This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 6 **SPE:** SPI enable

- 0: Peripheral disabled
- 1: Peripheral enabled

*Note: When disabling the SPI, follow the procedure described in [Procedure for disabling the SPI on page 1086](#).*

*This bit is not used in I<sup>2</sup>S mode.*

Bits 5:3 **BR[2:0]:** Baud rate control

- 000: f<sub>PCLK</sub>/2
- 001: f<sub>PCLK</sub>/4
- 010: f<sub>PCLK</sub>/8
- 011: f<sub>PCLK</sub>/16
- 100: f<sub>PCLK</sub>/32
- 101: f<sub>PCLK</sub>/64
- 110: f<sub>PCLK</sub>/128
- 111: f<sub>PCLK</sub>/256

*Note: These bits should not be changed when communication is ongoing.*

*These bits are not used in I<sup>2</sup>S mode.*

Bit 2 **MSTR:** Master selection

- 0: Slave configuration
- 1: Master configuration

*Note: This bit should not be changed when communication is ongoing.*

*This bit is not used in I<sup>2</sup>S mode.*

Bit 1 **CPOL:** Clock polarity

- 0: CK to 0 when idle
- 1: CK to 1 when idle

*Note: This bit should not be changed when communication is ongoing.*

*This bit is not used in SPI TI mode except the case when CRC is applied at TI mode.*

Bit 0 **CPHA:** Clock phase

- 0: The first clock transition is the first data capture edge
- 1: The second clock transition is the first data capture edge

*Note: This bit should not be changed when communication is ongoing.*

*This bit is not used in SPI TI mode except the case when CRC is applied at TI mode.*

### 32.9.2 SPI control register 2 (SPIx\_CR2)

Address offset: 0x04

Reset value: 0x0700

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.		LDMA_TX	LDMA_RX	FRXTH	DS[3:0]				TXEIE	RXNEIE	ERRIE	FRF	NSSP	SSOE	TXDMAEN	RXDMAEN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **LDMA\_TX:** Last DMA transfer for transmission

This bit is used in data packing mode, to define if the total number of data to transmit by DMA is odd or even. It has significance only if the TXDMAEN bit in the SPIx\_CR2 register is set and if packing mode is used (data length <= 8-bit and write access to SPIx\_DR is 16-bit wide). It has to be written when the SPI is disabled (SPE = 0 in the SPIx\_CR1 register).

- 0: Number of data to transfer is even
- 1: Number of data to transfer is odd

*Note: Refer to [Procedure for disabling the SPI on page 1086](#) if the CRCEN bit is set.*

*This bit is not used in I<sup>2</sup>S mode.*

Bit 13 **LDMA\_RX:** Last DMA transfer for reception

This bit is used in data packing mode, to define if the total number of data to receive by DMA is odd or even. It has significance only if the RXDMAEN bit in the SPIx\_CR2 register is set and if packing mode is used (data length <= 8-bit and write access to SPIx\_DR is 16-bit wide). It has to be written when the SPI is disabled (SPE = 0 in the SPIx\_CR1 register).

- 0: Number of data to transfer is even
- 1: Number of data to transfer is odd

*Note: Refer to [Procedure for disabling the SPI on page 1086](#) if the CRCEN bit is set.*

*This bit is not used in I<sup>2</sup>S mode.*

Bit 12 **FRXTH:** FIFO reception threshold

This bit is used to set the threshold of the RXFIFO that triggers an RXNE event

- 0: RXNE event is generated if the FIFO level is greater than or equal to 1/2 (16-bit)
- 1: RXNE event is generated if the FIFO level is greater than or equal to 1/4 (8-bit)

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bits 11:8 **DS[3:0]**: Data size

These bits configure the data length for SPI transfers.

- 0000: Not used
- 0001: Not used
- 0010: Not used
- 0011: 4-bit
- 0100: 5-bit
- 0101: 6-bit
- 0110: 7-bit
- 0111: 8-bit
- 1000: 9-bit
- 1001: 10-bit
- 1010: 11-bit
- 1011: 12-bit
- 1100: 13-bit
- 1101: 14-bit
- 1110: 15-bit
- 1111: 16-bit

If software attempts to write one of the “Not used” values, they are forced to the value “0111” (8-bit)

*Note: These bits are not used in I<sup>2</sup>S mode.*

Bit 7 **TXEIE**: Tx buffer empty interrupt enable

0: TXE interrupt masked

1: TXE interrupt not masked. Used to generate an interrupt request when the TXE flag is set.

Bit 6 **RXNEIE**: RX buffer not empty interrupt enable

0: RXNE interrupt masked

1: RXNE interrupt not masked. Used to generate an interrupt request when the RXNE flag is set.

Bit 5 **ERRIE**: Error interrupt enable

This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode and UDR, OVR, and FRE in I<sup>2</sup>S mode).

0: Error interrupt is masked

1: Error interrupt is enabled

Bit 4 **FRF**: Frame format

0: SPI Motorola mode

1 SPI TI mode

*Note: This bit must be written only when the SPI is disabled (SPE=0).*

*This bit is not used in I<sup>2</sup>S mode.*

Bit 3 **NSSP**: NSS pulse management

This bit is used in master mode only. It allows the SPI to generate an NSS pulse between two consecutive data when doing continuous transfers. In the case of a single data transfer, it forces the NSS pin high level after the transfer.

It has no meaning if CPHA = '1', or FRF = '1'.

0: No NSS pulse

1: NSS pulse generated

*Note: 1. This bit must be written only when the SPI is disabled (SPE=0).*

*2. This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 2 **SSOE:** SS output enable

0: SS output is disabled in master mode and the SPI interface can work in multimaster configuration

1: SS output is enabled in master mode and when the SPI interface is enabled. The SPI interface cannot work in a multimaster environment.

*Note: This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 1 **TXDMAEN:** Tx buffer DMA enable

When this bit is set, a DMA request is generated whenever the TXE flag is set.

0: Tx buffer DMA disabled

1: Tx buffer DMA enabled

Bit 0 **RXDMAEN:** Rx buffer DMA enable

When this bit is set, a DMA request is generated whenever the RXNE flag is set.

0: Rx buffer DMA disabled

1: Rx buffer DMA enabled

### 32.9.3 SPI status register (SPIx\_SR)

Address offset: 0x08

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	FTLVL[1:0]		FRLVL[1:0]		FRE	BSY	OVR	MODF	CRCE RR	UDR	CHSIDE	TXE	RXNE
			r	r	r	r	r	r	r	r	rc_w0	r	r	r	r

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:11 **FTLVL[1:0]:** FIFO transmission level

These bits are set and cleared by hardware.

00: FIFO empty

01: 1/4 FIFO

10: 1/2 FIFO

11: FIFO full (considered as FULL when the FIFO threshold is greater than 1/2)

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bits 10:9 **FRLVL[1:0]:** FIFO reception level

These bits are set and cleared by hardware.

00: FIFO empty

01: 1/4 FIFO

10: 1/2 FIFO

11: FIFO full

*Note: These bits are not used in I<sup>2</sup>S mode and in SPI receive-only mode while CRC calculation is enabled.*

Bit 8 **FRE:** Frame format error

This flag is used for SPI in TI slave mode and I<sup>2</sup>S slave mode. Refer to [Section 32.5.11: SPI error flags](#) and [Section 32.7.9: I2S error flags](#).

This flag is set by hardware and reset when SPIx\_SR is read by software.

0: No frame format error

1: A frame format error occurred

**Bit 7 BSY:** Busy flag

- 0: SPI (or I2S) not busy
  - 1: SPI (or I2S) is busy in communication or Tx buffer is not empty
- This flag is set and cleared by hardware.

*Note: The BSY flag must be used with caution: refer to [Section 32.5.10: SPI status flags and Procedure for disabling the SPI](#) on page 1086.*

**Bit 6 OVR:** Overrun flag

- 0: No overrun occurred
  - 1: Overrun occurred
- This flag is set by hardware and reset by a software sequence. Refer to [I2S error flags on page 1119](#) for the software sequence.

**Bit 5 MODF:** Mode fault

- 0: No mode fault occurred
- 1: Mode fault occurred

This flag is set by hardware and reset by a software sequence. Refer to [Section : Mode fault \(MODF\) on page 1096](#) for the software sequence.

*Note: This bit is not used in I<sup>2</sup>S mode.*

**Bit 4 CRCERR:** CRC error flag

- 0: CRC value received matches the SPIx\_RXCRCR value
- 1: CRC value received does not match the SPIx\_RXCRCR value

*Note: This flag is set by hardware and cleared by software writing 0.*

*This bit is not used in I<sup>2</sup>S mode.*

**Bit 3 UDR:** Underrun flag

- 0: No underrun occurred
- 1: Underrun occurred

This flag is set by hardware and reset by a software sequence. Refer to [I2S error flags on page 1119](#) for the software sequence.

*Note: This bit is not used in SPI mode.*

**Bit 2 CHSIDE:** Channel side

- 0: Channel Left has to be transmitted or has been received
- 1: Channel Right has to be transmitted or has been received

*Note: This bit is not used in SPI mode. It has no significance in PCM mode.*

**Bit 1 TXE:** Transmit buffer empty

- 0: Tx buffer not empty
- 1: Tx buffer empty

**Bit 0 RXNE:** Receive buffer not empty

- 0: Rx buffer empty
- 1: Rx buffer not empty

### 32.9.4 SPI data register (SPIx\_DR)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **DR[15:0]**: Data register

Data received or to be transmitted

The data register serves as an interface between the Rx and Tx FIFOs. When the data register is read, RxFIFO is accessed while the write to data register accesses TxFIFO (See [Section 32.5.9: Data transmission and reception procedures](#)).

*Note:* *Data is always right-aligned. Unused bits are ignored when writing to the register, and read as zero when the register is read. The Rx threshold setting must always correspond with the read access currently used.*

### 32.9.5 SPI CRC polynomial register (SPIx\_CRCPR)

Address offset: 0x10

Reset value: 0x0007

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRCPOLY[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CRCPOLY[15:0]**: CRC polynomial register

This register contains the polynomial for the CRC calculation.

The CRC polynomial (0x0007) is the reset value of this register. Another polynomial can be configured as required.

*Note:* *The polynomial value should be odd only. No even value is supported.*

### 32.9.6 SPI Rx CRC register (SPIx\_RXCRCR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXCRC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **RXCRC[15:0]**: Rx CRC register

When CRC calculation is enabled, the RXCRC[15:0] bits contain the computed CRC value of the subsequently received bytes. This register is reset when the CRCEN bit in SPIx\_CR1 register is written to 1. The CRC is calculated serially using the polynomial programmed in the SPIx\_CRCPR register.

Only the 8 LSB bits are considered when the CRC frame format is set to be 8-bit length (CRCL bit in the SPIx\_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit CRC frame format is selected (CRCL bit in the SPIx\_CR1 register is set). CRC calculation is done based on any CRC16 standard.

*Note: A read to this register when the BSY Flag is set could return an incorrect value.*

*These bits are not used in I<sup>2</sup>S mode.*

**32.9.7 SPI Tx CRC register (SPIx\_TXCRCR)**

Address offset: 0x18

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXCRC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **TXCRC[15:0]**: Tx CRC register

When CRC calculation is enabled, the TXCRC[7:0] bits contain the computed CRC value of the subsequently transmitted bytes. This register is reset when the CRCEN bit of SPIx\_CR1 is written to 1. The CRC is calculated serially using the polynomial programmed in the SPIx\_CRCPR register.

Only the 8 LSB bits are considered when the CRC frame format is set to be 8-bit length (CRCL bit in the SPIx\_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit CRC frame format is selected (CRCL bit in the SPIx\_CR1 register is set). CRC calculation is done based on any CRC16 standard.

*Note: A read to this register when the BSY flag is set could return an incorrect value.*

*These bits are not used in I<sup>2</sup>S mode.*

**32.9.8 SPIx\_I2S configuration register (SPIx\_I2SCFGR)**

Address offset: 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	ASTR TEN	I2SMOD	I2SE	I2SCFG[1:0]		PCMSYNC	Res.	I2SSSTD[1:0]		CKPOL	DATLEN[1:0]		CHLEN
			rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw

Bits 15:13 Reserved, must be kept at reset value.

Bit 12 **ASTRTEM**: Asynchronous start enable.

0: The Asynchronous start is disabled.

When the I<sup>2</sup>S is enabled in slave mode, the hardware starts the transfer when the I<sup>2</sup>S clock is received and an appropriate transition 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 hardware starts the transfer when the I<sup>2</sup>S clock is received and the appropriate level is detected on the WS signal.

*Note: The appropriate **transition** is a falling edge on WS signal when I<sup>2</sup>S Philips Standard is used, or a rising edge for other standards.*

*The appropriate **level** is a low level on WS signal when I<sup>2</sup>S Philips Standard is used, or a high level for other standards.*

*Please refer to [Section 32.7.4: Start-up description](#) for additional information.*

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 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[1:0]**: I<sup>2</sup>S configuration mode

00: Slave - transmit

01: Slave - receive

10: Master - transmit

11: Master - receive

*Note: These bits should be configured when the I<sup>2</sup>S is disabled.*

*They are not used in SPI mode.*

Bit 7 **PCMSYNC**: PCM frame synchronization

0: Short frame synchronization

1: Long frame synchronization

*Note: This bit has a meaning only if I2SSTD = 11 (PCM standard is used).*

*It is not used in SPI mode.*

Bit 6 Reserved, must be kept at reset value.

Bits 5:4 **I2SSTD[1:0]**: 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 32.7.3 on page 1103](#)

*Note: For correct operation, these bits should be configured when the I<sup>2</sup>S is disabled.*

*They are not used in SPI mode.*

Bit 3 **CKPOL**: Inactive state clock polarity

- 0: I2S clock inactive state is low level
- 1: I2S clock inactive state is high level

*Note: For correct operation, this bit should be configured when the I2S is disabled.*

*It is not used in SPI mode.*

*The bit CKPOL does not affect the CK edge sensitivity used to receive or transmit the SD and WS signals.*

Bits 2:1 **DATLEN[1:0]**: Data length to be transferred

- 00: 16-bit data length
- 01: 24-bit data length
- 10: 32-bit data length
- 11: Not allowed

*Note: For correct operation, these bits should be configured when the I2S is disabled.*

*They are not used in SPI mode.*

Bit 0 **CHLEN**: Channel length (number of bits per audio channel)

- 0: 16-bit wide
- 1: 32-bit wide

The bit write operation has a meaning only if DATLEN = 00 otherwise the channel length is fixed to 32-bit by hardware whatever the value filled in.

*Note: For correct operation, this bit should be configured when the I2S is disabled.*

*It is not used in SPI mode.*

### 32.9.9 SPIx\_I2S prescaler register (SPIx\_I2SPR)

Address offset: 0x20

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	MCKOE	ODD	I2SDIV[7:0]							

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **MCKOE**: Master clock output enable

- 0: Master clock output is disabled
- 1: Master clock output is enabled

*Note: This bit should be configured when the I2S is disabled. It is used only when the I2S is in master mode.*

*It is not used in SPI mode.*

Bit 8 **ODD**: Odd factor for the prescaler

- 0: Real divider value is = I2SDIV \*2
  - 1: Real divider value is = (I2SDIV \* 2) + 1
- Refer to [Section 32.7.4 on page 1110](#).

*Note: This bit should be configured when the I2S is disabled. It is used only when the I2S is in master mode.*

*It is not used in SPI mode.*

Bits 7:0 **I2SDIV[7:0]**: I2S linear prescaler

I2SDIV [7:0] = 0 or I2SDIV [7:0] = 1 are forbidden values.

Refer to [Section 32.7.4 on page 1110](#).

*Note: These bits should be configured when the I2S is disabled. They are used only when the I2S is in master mode.*

*They are not used in SPI mode.*

### 32.9.10 SPI/I2S register map

*Table 211* shows the SPI/I2S register map and reset values.

**Table 211. SPI/I2S register map and reset values**

Offset	Register name reset value	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	SPIx_CR1	BIDIMODE															
	Reset value	0	LDMA_TX	0	LDMA_RX	0	CRCEN	0	CRCNEXT	0	CRCL	0	RXONLY	0	SSM	0	0
0x04	SPIx_CR2	Res.	Res.	Res.	Res.	DS[3:0]											
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	SPIx_SR	Res.	Res.	Res.	Res.	FRLVL[1:0]	FTLVL[1:0]	1	1	BSY	TXEIE	0	LSBFIRST	0			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	RXNEIE	0	SPE	BR [2:0]	
0x0C	SPIx_DR																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	SPIx_CRCPR												CRCPOLY[15:0]				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	SPIx_RXCRCR												RXCRC[15:0]				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	SPIx_TXCRCR												TXCRC[15:0]				
	Reset value	0	0	0	0	I2SMOD	I2SE	0	0	0	0	0	0	0	0	0	0
0x1C	SPIx_I2SCFGR	Res.	Res.	Res.	Res.	ASTRTEN	0	I2SCFG[1:0]	PCMSYNC	0	0	0	OVR	0	RXNEIE	0	SPE
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	SPIx_I2SPR	Res.	Res.	Res.	Res.	MCKOE	0	ODD	0	0	0	0	I2STD	0	ERRIE	0	5
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 33 Serial audio interface (SAI)

### 33.1 Introduction

The SAI interface (serial audio interface) offers a wide set of audio protocols due to its flexibility and wide range of configurations. Many audio protocols can be addressed thanks to SAI “free protocol mode”. The free protocol mode allows the application to define the audio frame shape, number of slots, slot size, data size, and so on. For example, the SAI supports I2S standards, LSB- or MSB-justified, PCM/DSP, TDM, and AC'97 protocols. 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 subblocks. Each block has its own clock generator and I/O line controller.

The SAI works in master or slave configuration. The audio subblocks are either receiver or transmitter and work synchronously or not (with respect to the other one).

The SAI can be connected with other SAIs to work synchronously.

### 33.2 SAI main features

- Two independent audio subblocks, which can be transmitters or receivers with their respective FIFO
- 8-word integrated FIFOs for each audio subblock
- Synchronous or asynchronous mode between the audio subblocks
- Possible synchronization between multiple SAIs
- Master or slave configuration independent for both audio subblocks
- Clock generator for each audio block to target independent audio frequency sampling when both audio subblocks 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

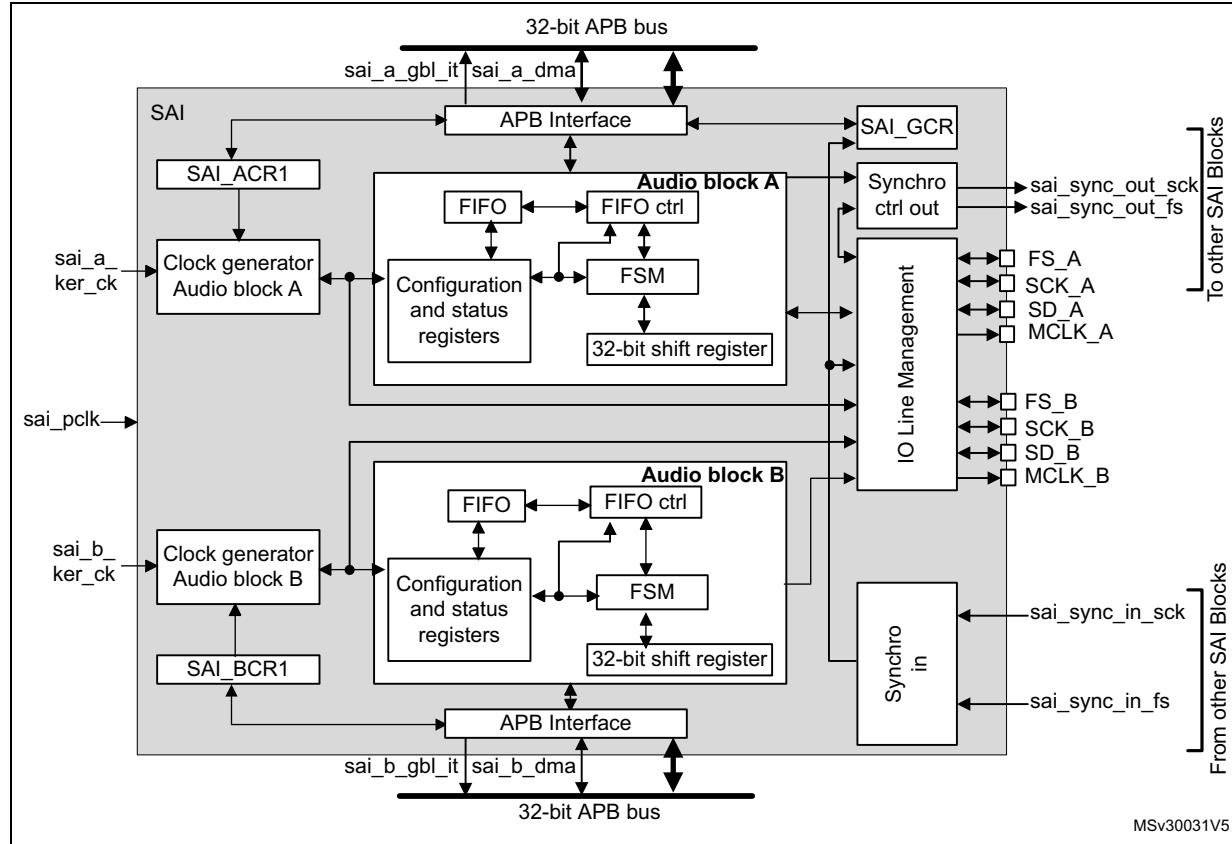
- Interrupt sources when enabled:
  - Errors
  - FIFO requests
- 2-channel DMA interface

### 33.3 SAI functional description

#### 33.3.1 SAI block diagram

*Figure 392* shows the SAI block diagram while *Table 212* and *Table 213* list SAI internal and external signals.

**Figure 392. SAI functional block diagram**



The SAI is mainly composed of two audio subblocks 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 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 subblocks 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 subblock 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 is 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 subblock.*

### 33.3.2 SAI pins and internal signals

**Table 212. SAI internal input/output signals**

Internal signal name	Signal type	Description
sai_a_gbl_it/ sai_b_gbl_it	Output	Audio block A and B global interrupts
sai_a_dma, sai_b_dma	Input/output	Audio block A and B DMA acknowledges and requests
sai_sync_out_sck, sai_sync_out_fs	Output	Internal clock and frame synchronization output signals exchanged with other SAI blocks
sai_sync_in_sck, sai_sync_in_fs	Input	Internal clock and frame synchronization input signals exchanged with other SAI blocks
sai_a_ker_ck/ sai_b_ker_ck	Input	Audio block A/B kernel clock
sai_pclk	Input	APB clock

**Table 213. SAI input/output pins**

Pin name	Pin type	Comments
SAI_SCK_A/B	Input/output	Audio block A/B bit clock
SAI_MCLK_A/B	Output	Audio block A/B master clock
SAI_SD_A/B	Input/output	Data line for block A/B
SAI_FS_A/B	Input/output	Frame synchronization line for audio block A/B

### 33.3.3 Main SAI modes

Each audio subblock of the SAI can be configured to be master or slave via the 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 the 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 subblock is configured in asynchronous mode, then the SCK\_x and FS\_x pins are configured as inputs.
- If the SAI subblock is configured to operate synchronously with another SAI interface or with the second audio subblock, the corresponding SCK\_x and FS\_x pins are left free to be used as general purpose I/Os.

In slave mode, the 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 block can use a different audio protocol. When PRTCFG[1:0] of the SAI\_xCR1 register is set to 0, the free protocol mode is selected and each SAI subblock can emulate I2S standards, LSB- or MSB-justified, PCM/DSP, TDM, or AC'97 protocols.

Each audio subblock 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, the SAI\_SD\_x pin is 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 is enabled by the SAIEN bit in the SAI\_xCR1 register. As soon as this bit is active, the transmitter or the receiver is sensitive to the activity on the clock, data, and synchronization lines in slave mode.

In master Tx mode, enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO. However, FS signal generation is conditioned by the presence of data in the FIFO. After the FIFO receives the first data to transmit, this data is output to external slaves. If there is no data to transmit in the FIFO, 0 values are then sent in the audio frame with an underrun flag generation.

In slave mode, the audio frame starts when the audio block is enabled and when a start of frame is detected.

In Slave Tx mode, no underrun event is possible on the first frame after the audio block is enabled, because the mandatory operating sequence in this case is:

1. Write into the SAI\_xDR (by software or by DMA).
2. Wait until the FIFO threshold (FLH) flag is different from 0b000 (FIFO empty).
3. Enable the audio block in slave transmitter mode.

### 33.3.4 SAI synchronization mode

There are two levels of synchronization, either at audio subblock level or at SAI level.

#### Internal synchronization

An audio subblock can be configured to operate synchronously with the second audio subblock 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 register).

*Note:* Due to internal resynchronization stages, PCLK APB frequency must be higher than twice the bit rate clock frequency.

### External synchronization

The audio subblocks can also be configured to operate synchronously with another SAI. This can be done as follows:

1. The SAI, which is configured as the source from which the other SAI is synchronized, has to define which of its audio subblocks 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 receives the synchronization signals, has to select which SAI provides the synchronization by setting the proper value on SYNCIN[1:0] bits. For each of the two SAI audio subblocks, the user must then specify if it operates synchronously with the other SAI via the SYNCEN bit.

*Note:* The SYNCIN[1:0] and SYNCOUT[1:0] bits are located in the SAI\_GCR register, and the SYNCEN bits in the SAI\_xCR1 register.

If both audio subblocks 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 'Reserved' must not be used.

**Table 214. External synchronization selection**

Block instance	SYNCIN= 3	SYNCIN= 2	SYNCIN= 1	SYNCIN= 0
SAI1	Reserved	Reserved	SAI2 sync.	Reserved
SAI2	Reserved	Reserved	Reserved	SAI1 sync.

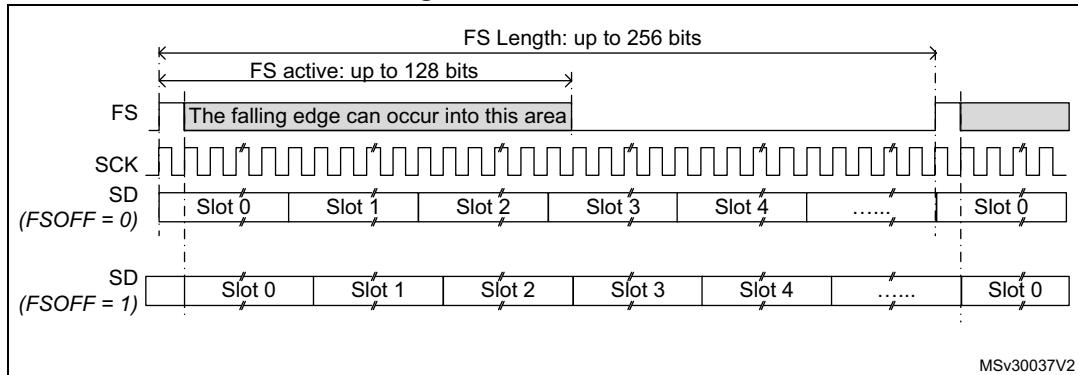
### 33.3.5 Audio data size

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 is sent first, depending on the configuration of the LSBFIRST bit in the SAI\_xCR1 register.

### 33.3.6 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 to target the different audio protocols with their own specificities concerning this frame synchronization behavior. This reconfigurability is done using the SAI\_xFRCR register. [Figure 393](#) illustrates this flexibility.

**Figure 393. 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 configuring the 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 are extended to 0 or the SD line is released to high-Z depending on the state of bit TRIS in the SAI\_xCR2 register (refer to [FS signal role](#)). In reception mode, the remaining bit is ignored.

If bit NODIV is cleared, (FRL+1) must be equal to a power of 2, from 8 to 256, to ensure that an audio frame contains an integer number of MCLK pulses per bit clock cycle.

If bit NODIV is set, the (FRL+1) field can take any value from 8 to 256. Refer to [Section 33.3.8: 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 ongoing audio frame. In this case, an error is generated. For more details, refer to [Section 33.3.13: 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

The FS POL bit in the SAI\_xFRCR register sets the active polarity of the FS pin from which a frame is started. The start of the frame is edge sensitive.

In slave mode, the audio block waits for a valid frame to start transmitting or receiving. The start of the frame is synchronized to this signal. It is effective only if the start of the frame is not detected during an ongoing communication and assimilated to an anticipated start of frame (refer to [Section 33.3.13: Error flags](#)).

In master mode, the frame synchronization is sent continuously each time an audio frame is complete until the SAIEN bit in the SAI\_xCR1 register is cleared. If no data are present in the FIFO at the end of the previous audio frame, an underrun condition is managed as described in [Section 33.3.13: Error flags](#)), but the audio communication flow is not interrupted.

## Frame synchronization active level length

The FS ALL[6:0] bits of the SAI\_xFRCR register enable the configuration of 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.

## 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). The FSOFF bit in the SAI\_xFRCR register enables the possibility to choose between two configurations.

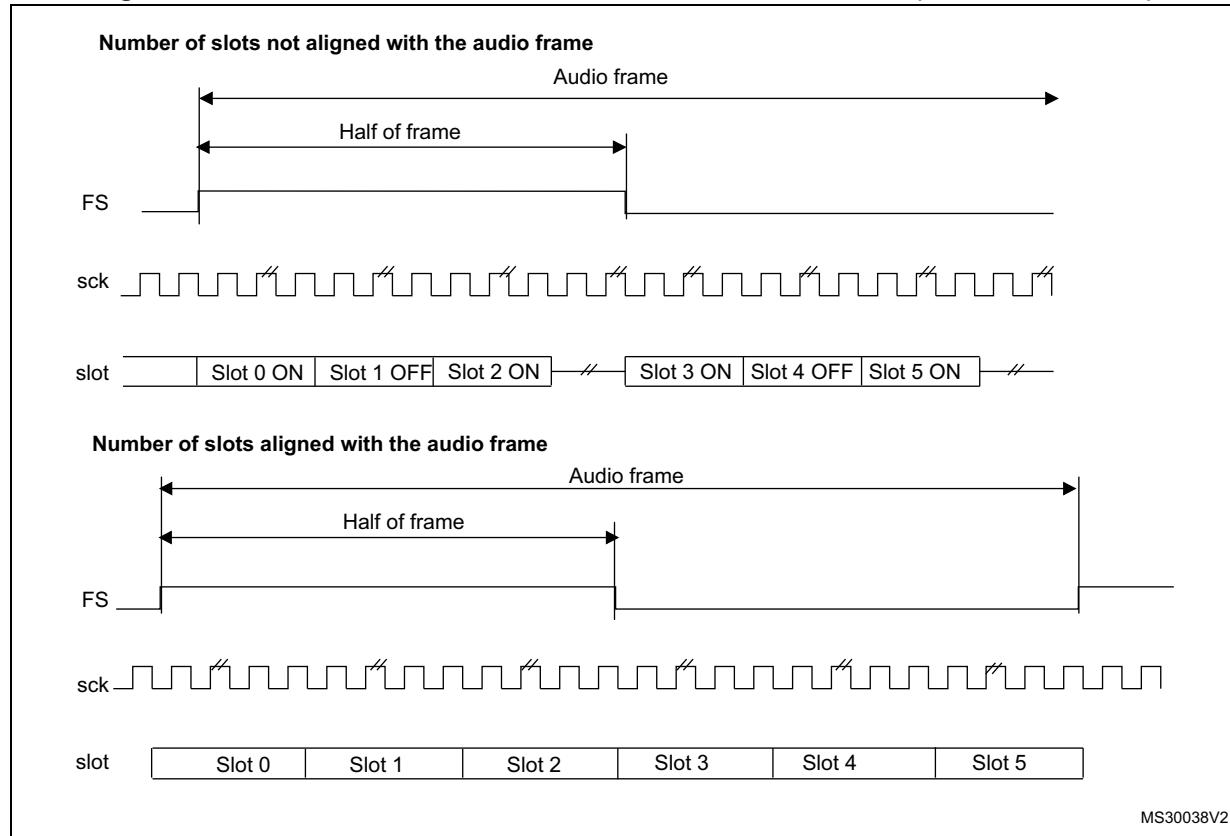
## FS signal role

The FS signal can have a different meaning depending on the FS function. FS DEF bit in the SAI\_xFRCR register selects which meaning it has:

- 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 or 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.

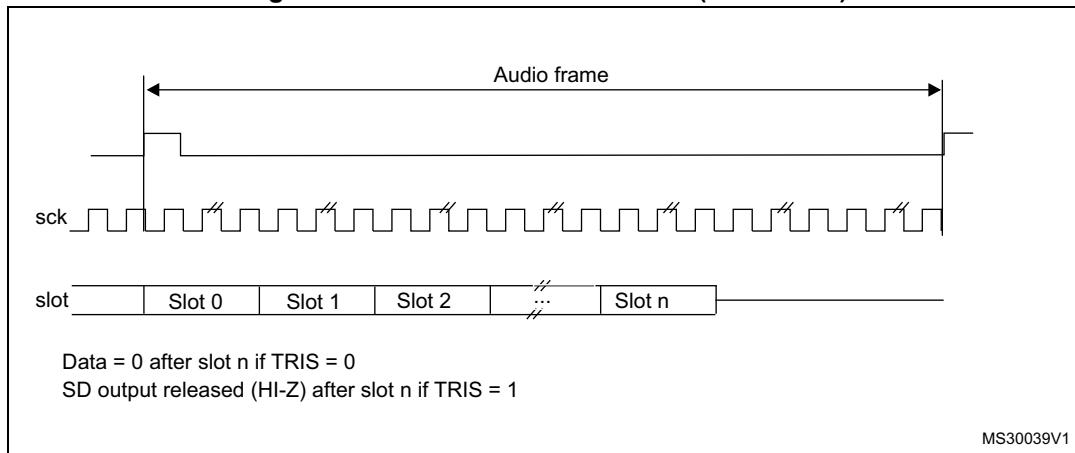
Otherwise, if TRIS = 1, the SD line is released to high-Z. In reception mode, the remaining bit clock cycles are not considered until the channel side changes.

**Figure 394. FS role is start of frame + channel side identification (FSDEF = TRIS = 1)**

1. The frame length must be even.

If the 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 is forced to 0 until the end of frame in case of transmitter,
- If TRIS = 1, the line is released to high-Z during the transfer of these remaining bits. In reception mode, these bits are discarded.

**Figure 395. 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 is released and left free for other purposes.

### 33.3.7 Slot configuration

The slot is the basic element in the audio frame. The number of slots in the audio frame is equal to NBSLOT[3:0] + 1.

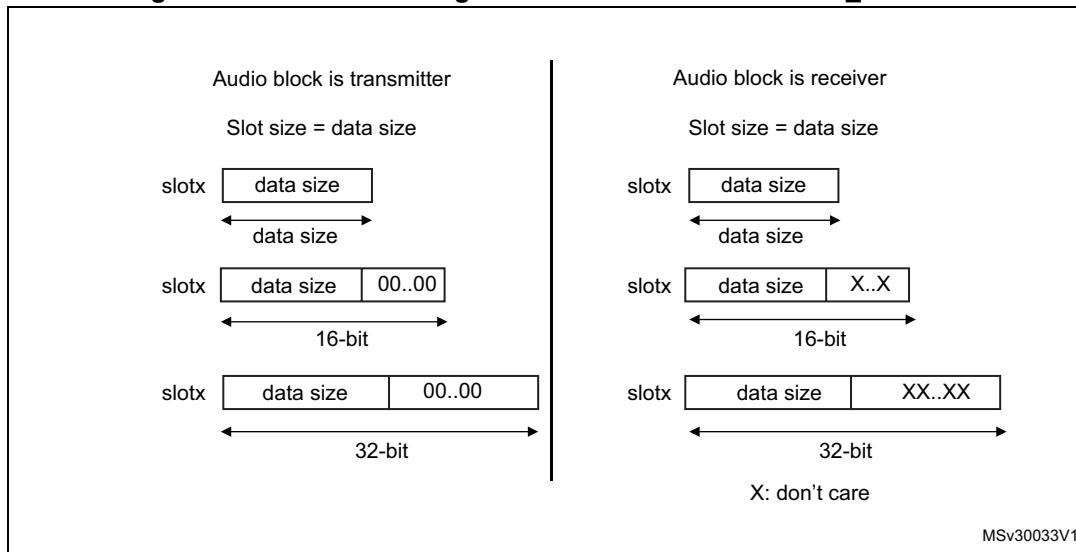
The maximum number of slots per audio frame is fixed at 16.

For AC'97 protocol or SPDIF (when bit PRTCFG[1:0] = 10 or PRTCFG[1:0] = 01), the number of slots is automatically set to target the protocol specification, and the value of NBSLOT[3:0] is ignored.

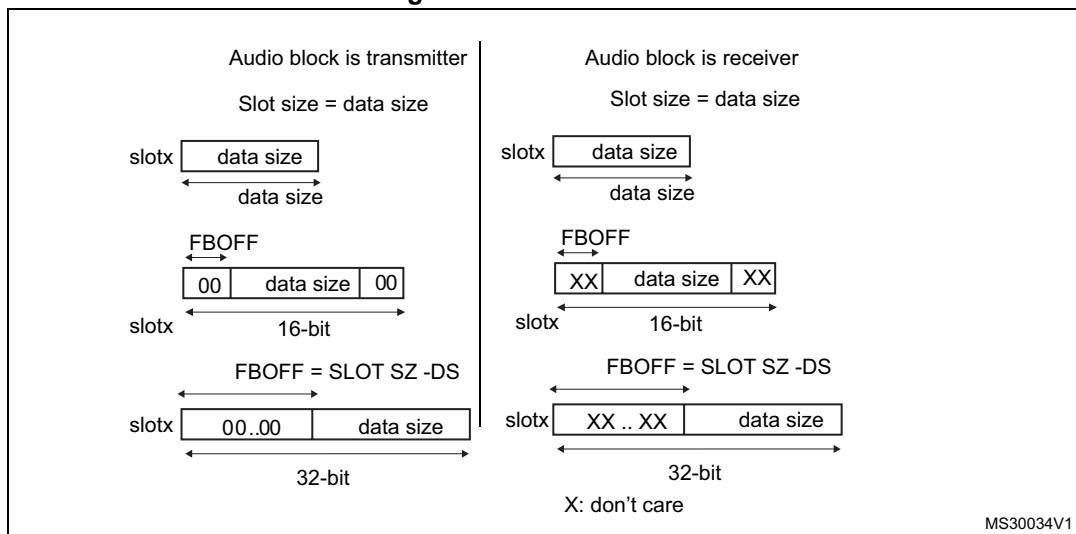
Each slot can be defined as a valid slot, or not, by setting SLOTEN[15:0] bits of the SAI\_xSLOTR register.

When an invalid slot is transferred, the SD data line is either forced to 0 or released to high-Z depending on the TRIS bit configuration (refer to [Output data line management on an inactive slot](#)) in transmitter mode. In receiver mode, the received value from the end of this slot is ignored. Consequently, there is 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 396](#). The size of the slots is selected by configuring the SLOTSZ[1:0] bits in the SAI\_xSLOTR register. The size is applied identically for each slot in an audio frame.

**Figure 396. 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 are 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 397. First bit offset**

It is mandatory to respect the following conditions to avoid bad SAI behavior:

- $\text{FBOFF} \leq (\text{SLOTSZ} - \text{DS})$
- $\text{DS} \leq \text{SLOTSZ}$
- $\text{NBSLOT} \times \text{SLOTSZ} \leq \text{FRL}$  (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 33.3.10: AC'97 link controller](#).

### 33.3.8 SAI clock generator

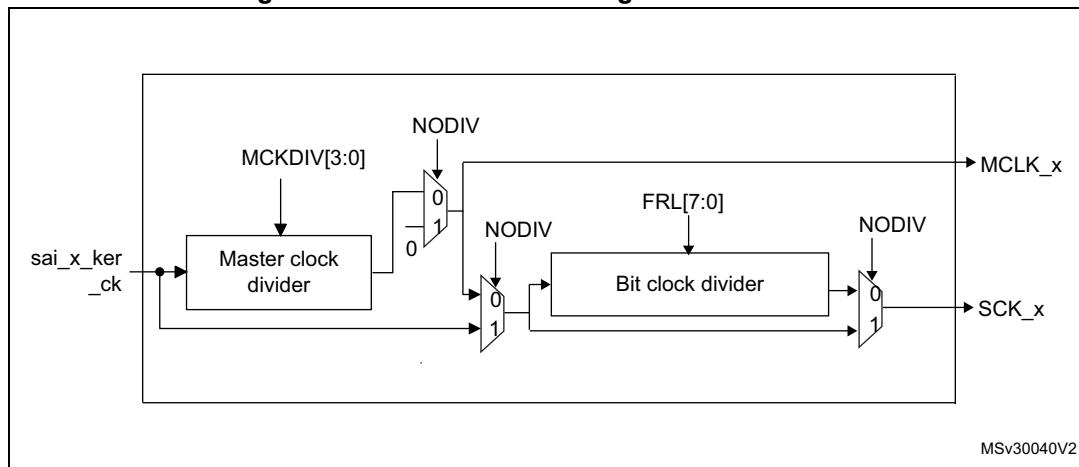
Each audio block has its own clock generator that makes these two blocks completely independent. There is no difference in terms of functionality between these two clock generators.

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 398* illustrates the architecture of the audio block clock generator.

**Figure 398. Audio block clock generator overview**



Note:

If NODIV is set to 1, the MCLK\_x signal is 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 Free protocol mode.

The division must be even 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 are encountered, as illustrated in *Table 215*.

**Table 215. 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 33.3.6: 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, the 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 33.3.11: SPDIF output](#) for details on clock generator programming in SPDIF mode.

### 33.3.9 Internal FIFOs

Each audio block in the SAI has its own FIFO. Depending on if the block is defined to be a transmitter or a receiver, the FIFO can be written or read, respectively. Thus, there is only one FIFO request linked to the FREQ bit in the SAI\_xSR register.

An interrupt is generated if the FREQIE bit is enabled in the SAI\_xIM register. This depends on:

- The FIFO threshold setting (FLVL bits in SAI\_xCR2).
- Communication direction (transmitter or receiver). Refer to [Interrupt generation in transmitter mode](#) and [Interrupt generation in reception mode](#).

#### Interrupt generation in transmitter mode

The interrupt generation depends on the FIFO configuration in transmitter mode:

- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO empty (FTH[2:0] set to 0b000), an interrupt is generated (FREQ bit set by hardware to 1 in the SAI\_xSR register) if no data are available in the SAI\_xDR register (FLVL[2:0] bits in SAI\_xSR are less than 0b001). This interrupt (FREQ bit in the SAI\_xSR register) is cleared by hardware when the FIFO is no longer empty (FLVL[2:0] bits in SAI\_xSR are different from 0b000), that is, one or more data are stored in the FIFO.
- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO quarter full (FTH[2:0] set to 0b001), an interrupt is generated (FREQ bit set by hardware to 1 in the SAI\_xSR register) if less than a quarter of the FIFO contains data (FLVL[2:0] bits in SAI\_xSR are less than 0b010). This interrupt (FREQ bit in the 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 than or equal to 0b010).
- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO half full (FTH[2:0] set to 0b010), an interrupt is generated (FREQ bit set by hardware to 1 in the SAI\_xSR register) if less than half of the FIFO contains data (FLVL[2:0] bits in SAI\_xSR are less than 0b011). This interrupt (FREQ bit in the 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 than or equal to 0b011).
- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO three quarter (FTH[2:0] set to 0b011), an interrupt is generated (FREQ bit is set by hardware to 1 in the SAI\_xSR register) if less than three quarters of the FIFO contains data (FLVL[2:0] bits in SAI\_xSR are less than 0b100). This interrupt (FREQ bit in the SAI\_xSR register) is cleared by hardware when at least three quarters of the FIFO contains data (FLVL[2:0] bits in SAI\_xSR are higher than or equal to 0b100).
- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO full (FTH[2:0] set to 0b100), an interrupt is generated (FREQ bit is set by hardware to 1 in the SAI\_xSR register) if the FIFO is not full (FLVL[2:0] bits in SAI\_xSR are less than 0b101). This interrupt (FREQ bit in the SAI\_xSR register) is cleared by hardware when the FIFO is full (FLVL[2:0] bits in SAI\_xSR are equal to 0b101).

#### Interrupt generation in reception mode

The interrupt generation depends on the FIFO configuration in reception mode:

- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO empty (FTH[2:0] set to 0b000), an interrupt is generated (FREQ bit is set by hardware to 1 in the SAI\_xSR register) if at least one data is available in the SAI\_xDR register (FLVL[2:0] bits

in SAI\_xSR are higher than or equal to 0b001). This interrupt (FREQ bit in the SAI\_xSR register) is cleared by hardware when the FIFO becomes empty (FLVL[2:0] bits in the SAI\_xSR are equal to 0b000), that is, no data are stored in FIFO.

- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO quarter full (FTH[2:0] set to 0b001), an interrupt is generated (FREQ bit is set by hardware to 1 in the SAI\_xSR register) if at least one quarter of the FIFO data locations are available (FLVL[2:0] bits in SAI\_xSR are higher than or equal to 0b010). This interrupt (FREQ bit in the 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 are less than 0b010).
- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO half full (FTH[2:0] set to 0b010), an interrupt is generated (FREQ bit is set by hardware to 1 in the SAI\_xSR register) if at least half of the FIFO data locations are available (FLVL[2:0] bits in SAI\_xSR are higher than or equal to 0b011). This interrupt (FREQ bit in the 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 are less than 0b011).
- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO three quarter full (FTH[2:0] set to 0b011), an interrupt is generated (FREQ bit is set by hardware to 1 in the SAI\_xSR register) if at least three quarters of the FIFO data locations are available (FLVL[2:0] bits in SAI\_xSR are higher than or equal to 0b100). This interrupt (FREQ bit in the SAI\_xSR register) is cleared by hardware when the FIFO has less than three quarters of the FIFO data locations available (FLVL[2:0] bits in SAI\_xSR is less than 0b100).
- When the FIFO threshold bits in the SAI\_xCR2 register are configured as FIFO full (FTH[2:0] set to 0b100), an interrupt is generated (FREQ bit is set by hardware to 1 in the SAI\_xSR register) if the FIFO is full (FLVL[2:0] bits in SAI\_xSR are equal to 0b101). This interrupt (FREQ bit in the SAI\_xSR register) is cleared by hardware when the FIFO is not full (FLVL[2:0] bits in SAI\_xSR are less than 0b101).

Like interrupt generation, the SAI can use the DMA if the DMAEN bit in the SAI\_xCR1 register is set. The FREQ bit assertion mechanism is the same as the interrupt 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 must be right-aligned when written in the SAI\_xDR register.

Data received are right-aligned in the SAI\_xDR register.

The FIFO pointers can be reinitialized when the SAI is disabled by configuring the FFLUSH bit in the SAI\_xCR2 register. If FFLUSH is set when the SAI is enabled, the data present in the FIFO are lost automatically.

### 33.3.10 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.

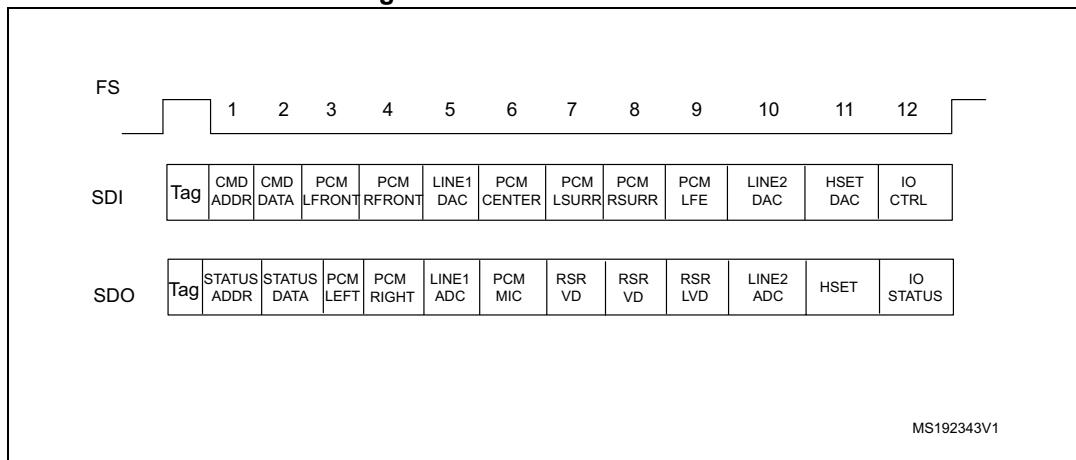
To select this protocol, set the PRTC[1:0] bits in the SAI\_xCR1 register to 10. When AC'97 mode is selected, only data sizes of 16 or 20 bits can be used, otherwise the SAI behavior is not guaranteed.

- The NBSLOT[3:0] and SLOTSZ[1:0] bits are consequently ignored.
- The number of slots is fixed at 13 slots. The first one is 16 bits wide and all the others are 20 bits wide (data slots).
- The 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 399* shows an AC'97 audio frame structure.

**Figure 399. AC'97 audio frame**



MS192343V1

**Note:** In the 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.

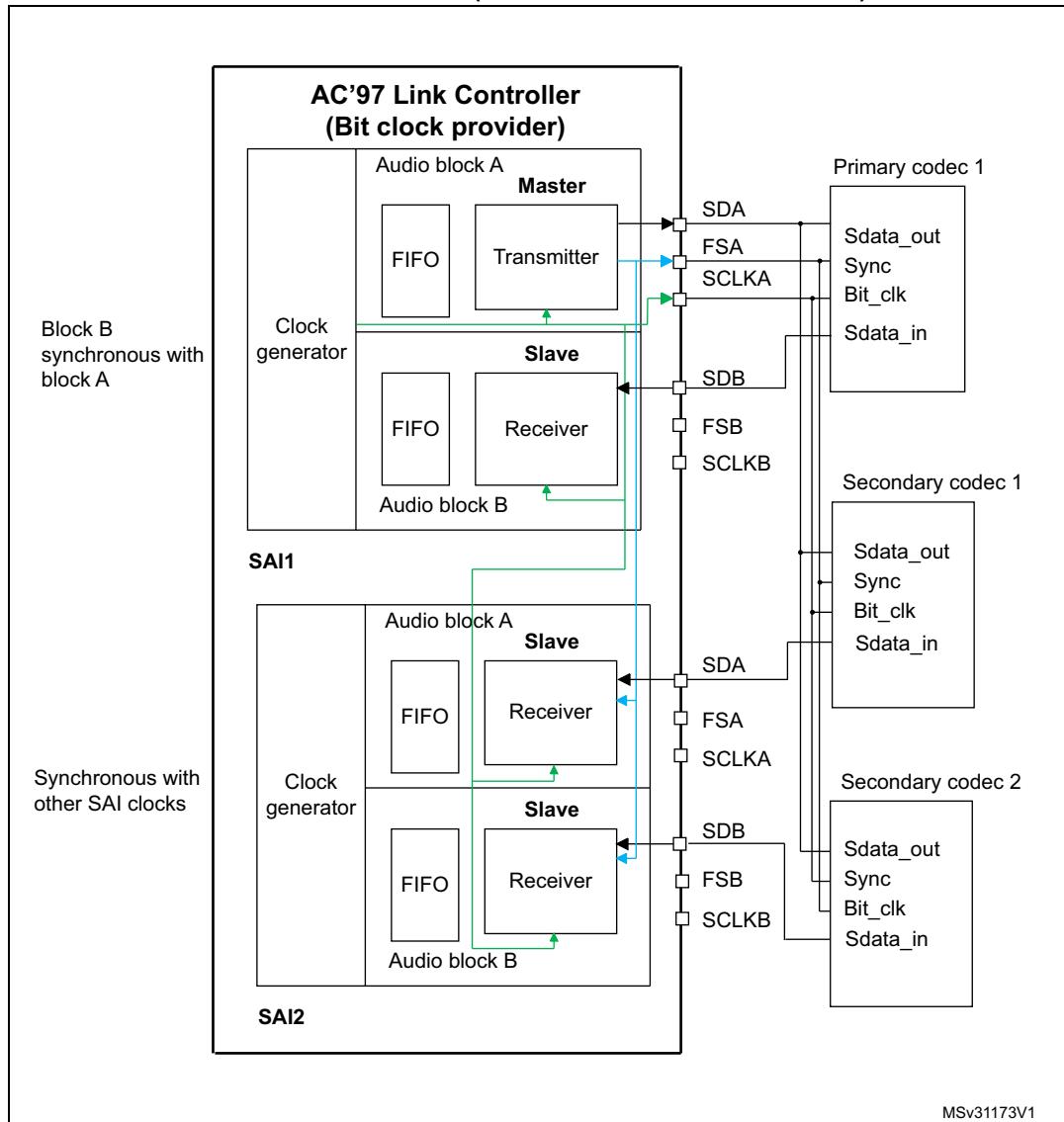
One SAI can be used to target an AC'97 point-to-point communication.

Using two SAIs (for devices featuring two embedded SAIs) enables the control of three external AC'97 decoders as illustrated in *Figure 400*.

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 400. Example of typical AC'97 configuration on devices featuring at least two embedded SAI (three external AC'97 decoders)**



MSv31173V1

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 slot 0 is decoded low. If bit CNRDYIE is enabled in the SAI\_xIM register, flag CNRDY is 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 must be set to 48 kHz. The formulas given in [Section 33.3.8: SAI clock generator](#) must be used with FRL = 255, to generate the proper frame rate ( $F_{FS\_x}$ ).

### 33.3.11 SPDIF output

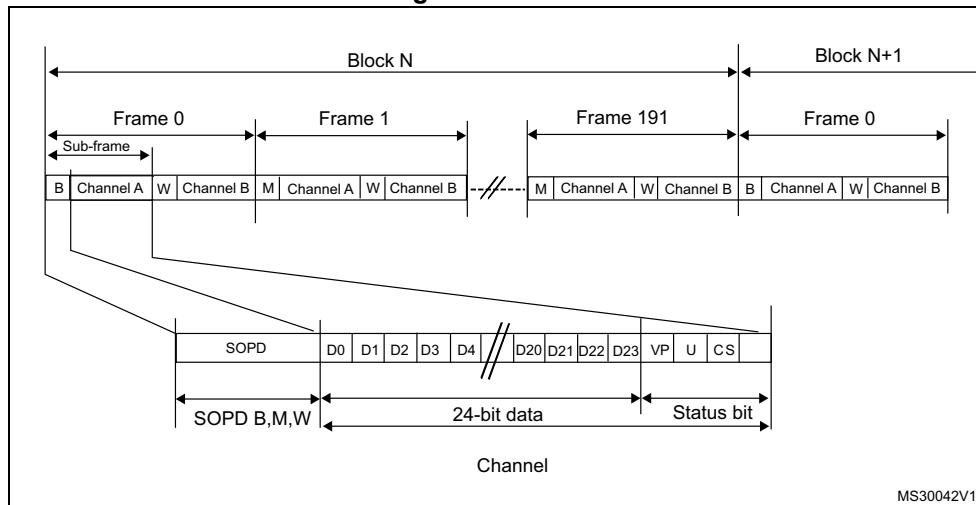
The SPDIF interface is available in transmitter mode only. It supports the audio IEC60958.

To select SPDIF mode, set the PRTC<sub>FG[1:0]</sub> bits to 01 in the SAI\_xCR1 register.

For SPDIF protocol:

- Only the SD data line is enabled.
- The FS, SCK, and MCLK I/Os pins are left free.
- The MODE[1] bit is forced to 0 to select the master mode 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 the 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 must 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 401](#).

**Figure 401. SPDIF format**



An SPDIF block contains 192 frames. Each frame is composed of two 32-bit subframes, generally one for the left channel and one for the right channel. Each subframe is composed of a SOPD pattern (4-bit) to specify if the subframe 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 216](#)). The next 28 bits of channel information are composed of 24 data bits + 4 status bits.

**Table 216. 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

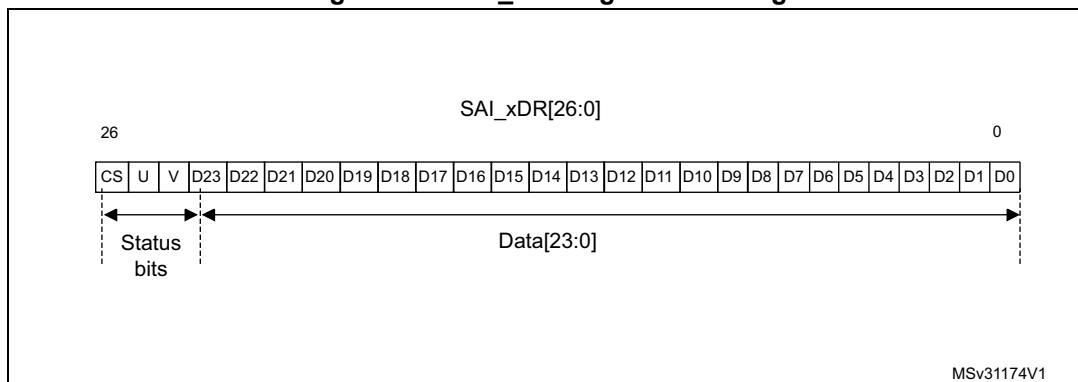
The data stored in SAI\_xDR has to be filled as follows:

- 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, the data must be mapped on SAI\_xDR[23:4].

If the data size is 16 bits, the data must be mapped on SAI\_xDR[23:8].

SAI\_xDR[23] always represents the MSB.

**Figure 402. SAI\_xDR register ordering**

Note:

*The transfer is always performed with LSB first.*

The SAI first sends the adequate preamble for each subframe in a block. The SAI\_xDR is then sent on the SD line (Manchester coded). The SAI ends the subframe by transferring the parity bit calculated as described in [Table 217](#).

**Table 217. 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 must be

executed to recover from an underrun error detected via the underrun interrupt or the underrun status bit:

1. Disable the DMA stream (via the DMA peripheral) if the DMA is used.
2. Disable the SAI and check that the peripheral is physically disabled by polling the SAIEN bit in the 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 the start of a new block (data for preamble B). If the DMA is used, the DMA source base address pointer must be updated accordingly.

5. Enable the DMA stream (DMA peripheral) again if the DMA is used to manage data transfers according to the new source base address.
6. Enable the SAI again by configuring the SAIEN bit in the SAI\_xCR1 register.

### Clock generator programming in SPDIF generator mode

For the SPDIF generator, the SAI provides a bit clock twice as fast as the symbol-rate. The table below shows examples of symbol rates with respect to the audio sampling rate.

**Table 218. Audio sampling frequency versus symbol rates**

Audio sampling frequencies ( $F_S$ )	Symbol-rate
44.1 kHz	2.8224 MHz
48 kHz	3.072 MHz
96 kHz	6.144 MHz
192 kHz	12.288 MHz

More generally, the relationship between the audio sampling frequency ( $F_S$ ) and the bit clock rate ( $F_{SCK\_x}$ ) is given by the formula:

$$F_S = \frac{F_{SCK\_x}}{128}$$

The bit clock rate is obtained as follows:

$$F_{SCK\_x} = F_{sai\_x\_ker\_ck}$$

*Note:* The above formulas are valid only if NODIV is set to 1 in the SAI\_ACR1 register.

### 33.3.12 Specific features

The SAI interface embeds specific features that 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 subblock is a transmitter or a receiver.

##### Audio subblock in transmission mode

In transmitter mode, the mute mode can be selected at any time. The mute mode is active for entire audio frames. The MUTE bit in the SAI\_xCR2 register enables the mute mode when it is configured 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 is still a mute frame.

If the number of slots set through the 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 the MUTEVAL bit in the SAI\_xCR2 register.

If the number of slots set in the NBSLOT[3:0] bits in the SAI\_xSLOTR register is greater than 2, the MUTEVAL bit in the SAI\_xCR2 register 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 subblock 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 the MUTEDETIE bit is set in SAI\_xCR2.

The mute frame counter is cleared when the audio subblock 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 the 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 is reduced by 2 since the data for slot 0 is duplicated into data slot 1.

To enable the mono mode:

1. Set the 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, like in transmitter mode. When it is set, only slot 0 data are stored in the FIFO. The data belonging to slot 1 are 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 processing 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 free protocol mode is selected), the application software can choose to process or not the data before sending it on the SD serial output line (compression) or to expand the data after the reception on the SD serial input line (expansion), as illustrated in [Figure 403](#). The two companding modes supported are the  $\mu$ -Law and the A-Law logs, 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).

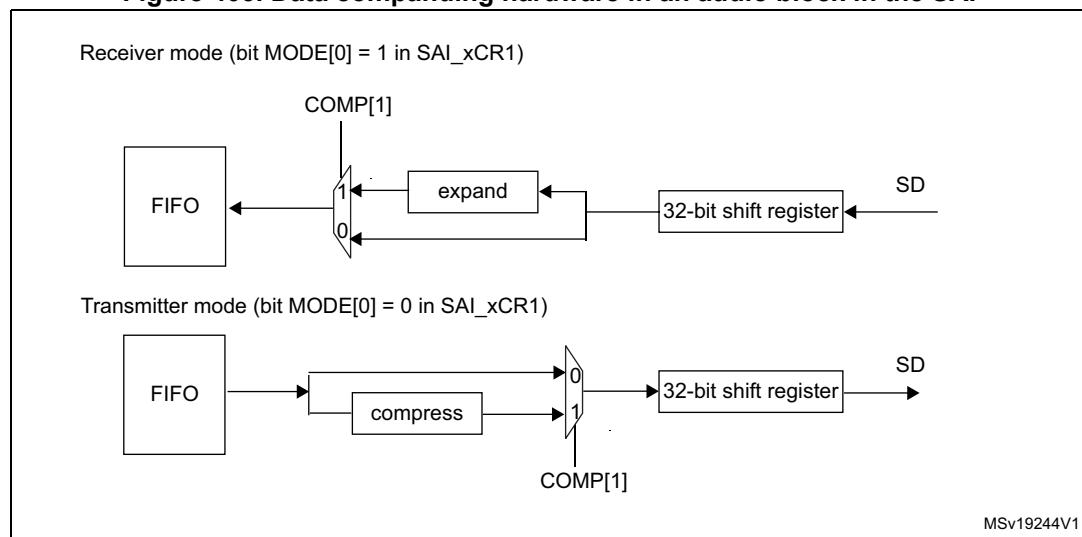
The European companding standard is A-Law and supports 13 bits of dynamic range (COMP[1:0] = 11 in the SAI\_xCR2 register).

Both  $\mu$ -Law and 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 bits wide. For this reason, the DS[2:0] bits in the SAI\_xCR1 register are forced to 010 when the SAI audio block is enabled (the SAIEN bit = 1 in the SAI\_xCR1 register) and when one of these two companding modes is selected through the COMP[1:0] bits.

If no companding processing is required, the COMP[1:0] bits must be kept clear.

**Figure 403. 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 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 is applied.
- If the SAI audio block is declared as a receiver, the expansion algorithm is 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 the TRIS bit).

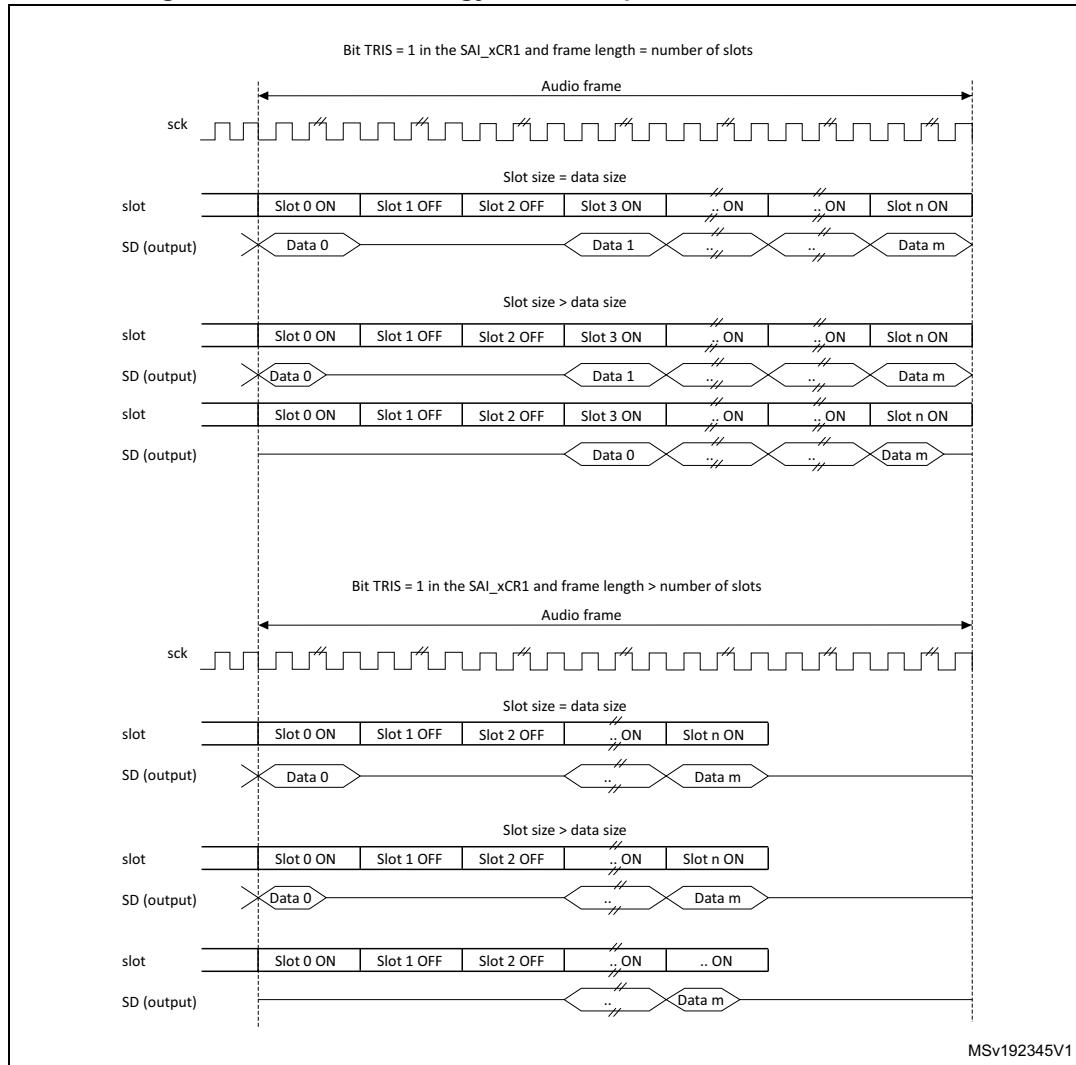
- Either the SAI forces 0 on the SD output line when an inactive slot is transmitted, or
- The line is released in high-Z state at the end of the last bit of data transferred, to release the line for other transmitters connected to this node.

It is important to note that the two transmitters cannot attempt to drive the same SD output pin simultaneously, which may 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 the bit SLOTSZ[1:0] = 10 in the SAI\_xSLOTR register. The SD output pin is then 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.

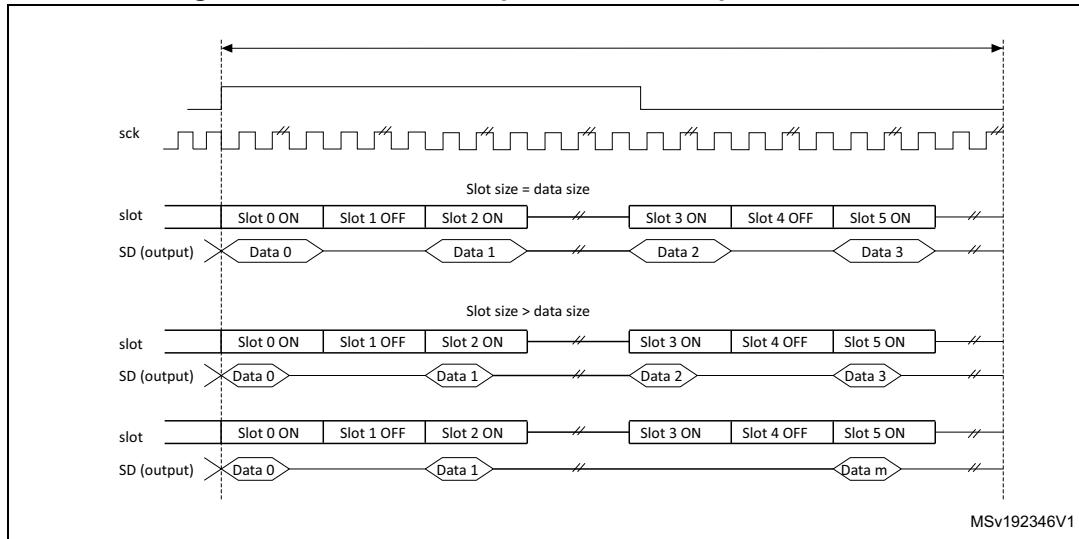
In addition, if the number of slots multiplied by the slot size is lower than the frame length, the SD output line is tri-stated when the padding to 0 is done to complete the audio frame.

*Figure 404* illustrates these behaviors.

Figure 404. 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 405](#) (where the 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 405. Tristate on output data line in a protocol like I<sub>2</sub>S**

If the TRIS bit in the SAI\_xCR2 register is cleared, all the high impedance states on the SD output line in [Figure 404](#) and [Figure 405](#) are replaced by a value of 0.

### 33.3.13 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.

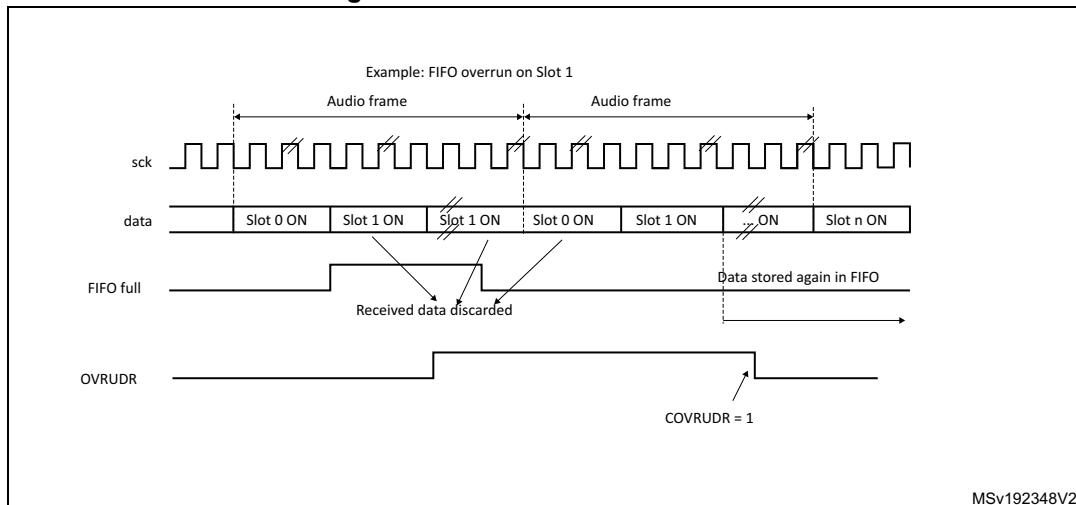
The overrun or underrun errors share the same bit since an audio block can be either receiver or transmitter and each audio block in a given SAI has its own SAI\_xSR register.

##### 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 OVRUDR flag in the SAI\_xSR register is set, and an interrupt is generated if the OVRUDRIE bit is set in the SAI\_xIM register. The slot number, from which the overrun occurs, is stored internally. No more data are 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 stores new data (from a new audio frame) from the slot number that was stored internally when the overrun condition was detected. This avoids data slot dealignment in the destination memory (refer to [Figure 406](#)).

The OVRUDR flag is cleared when the COVRUDR bit is set in the SAI\_xCLRFR register.

Figure 406. Overrun detection error



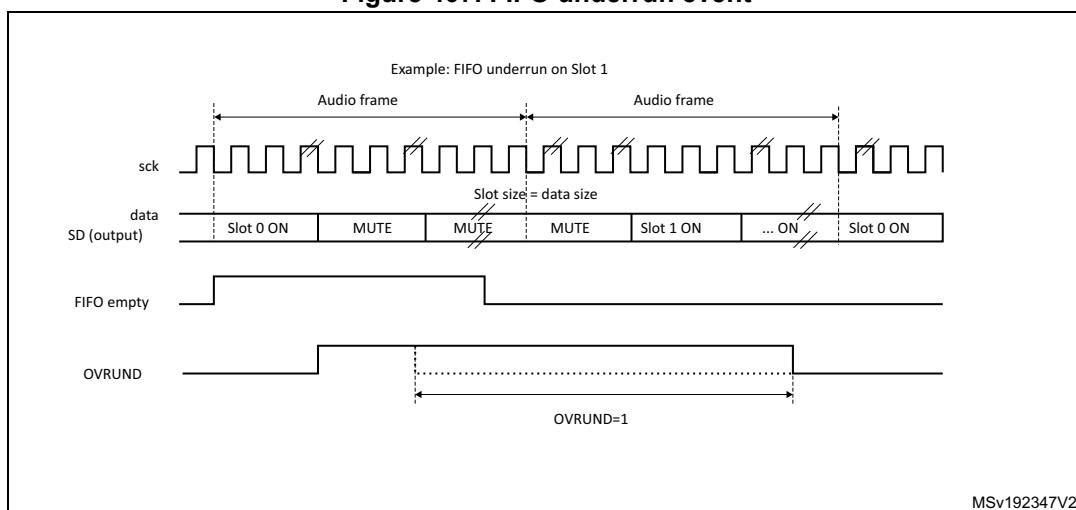
### Underrun

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 the 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 407](#)). 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 the COVRUDR bit in the SAI\_xCLRFR register.

The underrun event can occur when the audio subblock is configured as master or slave.

Figure 407. 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, and 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, the 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 the SAIEN bit in the SAI\_xCR1 register. To make sure that the SAI is disabled, read back the SAIEN bit and check it is set to 0.
2. Flush the FIFO via the FFLUS bit in the SAI\_xCR2 register.
3. Enable the SAI peripheral again (SAIEN bit set to 1).
4. The SAI block waits for the assertion on FS to restart the synchronization with master.

**Note:** *The AFSDET flag is not asserted in AC'97 mode since the SAI audio block acts as a link controller and generates the FS signal even when declared as slave. It has no meaning in SPDIF mode since the FS signal is not used.*

### Late frame synchronization detection

The LFSDET flag in the SAI\_xSR register can be set only when the SAI audio block operates as a slave. The frame length, the frame polarity, and the frame-offset configuration are known in register SAI\_xFRCR.

If the external master does not send the FS signal at the expected time, thus generating the signal too late, the LFSDET flag is set and an interrupt is generated if the LFSDETIE bit is set in the SAI\_xIM register.

The LFSDET flag is cleared when the CLFSDET bit is set in the SAI\_xCLRFR register.

The late frame synchronization detection flag is set when the corresponding error is detected. The SAI needs to be resynchronized with the master (see sequence described in [Anticipated frame synchronization detection \(AFSDET\)](#)).

In a noisy environment, glitches on the SCK clock may be wrongly detected by the audio block state machine and shift the SAI data at a wrong frame position. This event can be detected by the SAI and reported as a late frame synchronization detection error.

There is no corruption if the external master is not managing the audio data frame transfer in continuous mode, which must not be the case in most applications. In this case, the LFSDET flag is 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 (PRTC<sub>FG[1:0]</sub> = 10 in the SAI\_xCR1 register). If the 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 (slot 0) of the AC'97 audio frame. In this case, no data are 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 are captured when the codec is ready.

To clear the CNRDY flag, the 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 the 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.

The MODE, NODIV, and SAIEN bits belong to the SAI\_xCR1 register and FRL to the SAI\_xFRCR register.

If the WCKCFGIE bit is set, an interrupt is generated when the WCKCFG flag is set in the SAI\_xSR register. To clear this flag, set the CWCKCFG bit in the SAI\_xCLRFR register.

When the WCKCFG bit is set, the audio block is automatically disabled, thus performing a hardware clear of the SAIEN bit.

### 33.3.14 Disabling the SAI

The SAI audio block can be disabled at any moment by clearing the SAIEN bit in the SAI\_xCR1 register. All the already started frames are automatically completed before the SAI stops working. The SAIEN bit remains high until the SAI is completely switched off at the end of the current audio frame transfer.

If an audio block in the SAI operates synchronously with the other one, the one that is the master must be disabled first.

### 33.3.15 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 subblock supporting the basic DMA request/acknowledge protocol.

To configure the audio subblock for DMA transfer, set the 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 33.3.9: Internal FIFOs](#)). The DMA transfer direction is linked to the SAI audio subblock configuration:

- If the audio block operates as a transmitter, the audio block FIFO controller outputs a DMA request to load the FIFO with data written in the SAI\_xDR register.
- If the audio block operates as a receiver, the DMA request is related to read operations from the SAI\_xDR register.

Follow the sequence below to configure the SAI interface in DMA mode:

1. Configure the SAI and FIFO threshold levels to specify when the DMA request is launched.
2. Configure the SAI DMA channel.
3. Enable the DMA.
4. Enable the SAI interface.

### 33.4 SAI interrupts

The SAI supports 7 interrupt sources, as shown in [Table 219](#).

**Table 219. SAI interrupt sources**

Interrupt acronym	Interrupt source	Interrupt group	Audio block mode	Interrupt enable	Interrupt clear
SAI	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 33.3.9: 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
	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.

## 33.5 SAI registers

The peripheral registers have to be accessed by words (32 bits).

### 33.5.1 SAI global configuration register (SAI\_GCR)

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.	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] must be configured as “No synchronization output signals” when audio block is configured as SPDIF

01: Block A used for further synchronization for others SAI

10: Block B used for further synchronization for others SAI

11: Reserved. These bits must be set when both audio block (A and B) are disabled.

Bits 3:2 Reserved, must be kept at reset value.

Bits 1:0 **SYNCIN[1:0]**: Synchronization inputs

These bits are set and cleared by software.

Refer to [Table 214: External synchronization selection](#) for information on how to program this field.

These bits must be set when both audio blocks (A and B) are disabled.

They are meaningful if one of the two audio blocks is defined to operate in synchronous mode with an external SAI (SYNCEN[1:0] = 10 in SAI\_ACR1 or in SAI\_BCR1 registers).

### 33.5.2 SAI configuration register 1 (SAI\_ACR1)

Address offset: 0x04

Reset value: 0x0000 0040

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MCKDIV[3:0]				NODIV	Res.	DMAEN	SAIEN
								rw	rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res.	OUTD RIV	MONO	SYNCEN[1:0]	CKSTR	LSBFIRST	DS[2:0]				Res.	PRTCFG[1:0]	MODE[1:0]		
		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:20 MCKDIV[3:0]: Master clock divider

These bits are set and cleared by software. These bits are meaningless when the audio block operates in slave mode. They have to be configured when the audio block is disabled.

0000: Divides by 1 the master clock input.

Others: the master clock frequency is calculated accordingly to the following formula:

$$F_{SCK\_x} = \frac{F_{sai\_x\_ker\_ck}}{MCKDIV \times 2}$$

Bit 19 **NODIV**: No divider

This bit is set and cleared by software.

0: Master clock generator is enabled

1: No divider used in the clock generator (in this case Master Clock Divider bit has no effect)

Bit 18 Reserved, must be kept at reset value.

Bit 17 **DMAEN**: DMA enable

This bit is set and cleared by software.

0: DMA disabled

1: DMA enabled

*Note: Since the audio block defaults to operate as a transmitter after reset, the MODE[1:0] bits must be configured before setting DMAEN to avoid a DMA request in receiver mode.*

Bit 16 **SAIEN**: Audio block enable

This bit is set by software.

To switch off the audio block, the application software must program this bit to 0 and poll the bit till it reads back 0, meaning that the block is completely disabled. Before setting this bit to 1, check that it is set to 0, otherwise the enable command is not taken into account.

This bit enables to control the state of the SAI audio block. If it is disabled when an audio frame transfer is ongoing, the ongoing transfer completes and the cell is fully disabled at the end of this audio frame transfer.

0: SAI audio block disabled

1: SAI audio block enabled.

*Note: When the SAI block (A or B) is configured in master mode, the clock must be present on the SAI block input before setting SAIEN bit.*

Bits 15:14 Reserved, must be kept at reset value.

Bit 13 **OUTDRIV**: Output drive

This bit is set and cleared by software.

0: Audio block output driven when SAIEN is set

1: Audio block output driven immediately after the setting of this bit.

*Note: This bit has to be set before enabling the audio block and after the audio block configuration.*

Bit 12 **MONO**: Mono mode

This bit is set and cleared by software. It is meaningful only when the number of slots is equal to 2. When the mono mode is selected, slot 0 data are duplicated on slot 1 when the audio block operates as a transmitter. In reception mode, the slot1 is discarded and only the data received from slot 0 are stored. Refer to [Section : Mono/stereo mode](#) for more details.

0: Stereo mode

1: Mono mode.

Bits 11:10 **SYNCEN[1:0]**: Synchronization enable

These bits are set and cleared by software. They must be configured when the audio subblock is disabled.

00: audio subblock in asynchronous mode.

01: audio subblock is synchronous with the other internal audio subblock. In this case, the audio subblock must be configured in slave mode

10: audio subblock is synchronous with an external SAI embedded peripheral. In this case the audio subblock must be configured in Slave mode.

11: Reserved

*Note: The audio subblock must be configured as asynchronous when SPDIF mode is enabled.*

Bit 9 **CKSTR**: Clock strobing edge

This bit is set and cleared by software. It must be configured when the audio block is disabled. This bit has no meaning in SPDIF audio protocol.

0: Signals generated by the SAI change on SCK rising edge, while signals received by the SAI are sampled on the SCK falling edge.

1: Signals generated by the SAI change on SCK falling edge, while signals received by the SAI are sampled on the SCK rising edge.

Bit 8 **LSBFIRST**: Least significant bit first

This bit is set and cleared by software. It must be configured when the audio block is disabled. This bit has no meaning in AC'97 audio protocol since AC'97 data are always transferred with the MSB first. This bit has no meaning in SPDIF audio protocol since in SPDIF data are always transferred with LSB first.

0: Data are transferred with MSB first

1: Data are transferred with LSB first

Bits 7:5 **DS[2:0]**: Data size

These bits are set and cleared by software. These bits are ignored when the SPDIF protocols are selected (bit PRTCFCFG[1:0]), because the frame and the data size are fixed in such case. When the companding mode is selected through COMP[1:0] bits, DS[1:0] are ignored since the data size is fixed to 8 bits by the algorithm.

These bits must be configured when the audio block is disabled.

000: Reserved

001: Reserved

010: 8 bits

011: 10 bits

100: 16 bits

101: 20 bits

110: 24 bits

111: 32 bits

Bit 4 Reserved, must be kept at reset value.

Bits 3:2 **PRTCFCFG[1:0]**: Protocol configuration

These bits are set and cleared by software. These bits have to be configured when the audio block is disabled.

00: Free protocol. Free protocol enables 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).*

### 33.5.3 SAI configuration register 2 (SAI\_ACR2)

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
COMP[1:0]	CPL	MUTECNT[5:0]						MUTE VAL	MUTE	TRIS	F FLUSH	FTH[2:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	w	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:14 **COMP[1:0]**: Companding mode.

These bits are set and cleared by software. The  $\mu$ -Law and the A-Law log are a part of the CCITT G.711 recommendation, the type of complement that is 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 Free protocol mode 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 is set and an interrupt is 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 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 must 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 must 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 must 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 must 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/interrupt must be disabled

**Bits 2:0 FTH[2:0]:** FIFO threshold.

This bit is set and cleared by software.

000: FIFO empty

001: 1/4 FIFO

010: 1/2 FIFO

011: 3/4 FIFO

100: FIFO full

101: Reserved

110: Reserved

111: Reserved

### 33.5.4 SAI frame configuration register (SAI\_AFRCR)

Address offset: 0x0C

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 are 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 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 must 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. They must be configured when the audio block is disabled.

### 33.5.5 SAI slot register (SAI\_ASLOTR)

Address offset: 0x10

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 in the audio frame (including the number of inactive slots). The maximum number of slots is 16.

The number of slots must 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.

0000: Number of slots is 1.

0001: Number of slots is 2.

...

1111: Number of slots is 16.

**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 is undetermined.

Refer to [Output data line management on an inactive slot](#) for information on how to drive SD line.

These bits must be set when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

00: The slot size is equivalent to the data size (specified in DS[3:0] in the SAI\_xCR1 register).

01: 16-bit

10: 32-bit

11: Reserved

Bit 5 Reserved, must be kept at reset value.

**Bits 4:0 FBOFF[4:0]: First bit offset**

These bits are set and cleared by software.

The value set in this bitfield defines the position of the first data transfer bit in the slot. It represents an offset value. In transmission mode, the bits outside the data field are forced to 0. In reception mode, the extra received bits are discarded.

These bits must be set when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

**33.5.6 SAI interrupt mask register (SAI\_AIM)**

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.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LFSDETIE	AFSDETIE	CNRDYIE	FREQIE	WCKCFGIE	MUTEDETIE	OVRUDRIE								
									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 is 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 is 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 interrupt 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 interrupt 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 Free protocol 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.

### 33.5.7 SAI status register (SAI\_ASР)

Address offset: 0x18

Reset value: 0x0000 0008

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	FLVL[2:0]										
														r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LFSDET	AFSDET	CNRDY	FREQ	WCKCFG	MUTEDET	OVRUDR								
									r	r	r	r	r	r	r

Bits 31:19 Reserved, must be kept at reset value.

Bits 18:16 **FLVL[2:0]**: FIFO level threshold.

This bit is read only. The FIFO level threshold flag is managed only by hardware and its setting depends on SAI block configuration (transmitter or receiver mode).

000: FIFO empty (transmitter and receiver modes)

001: FIFO  $\leq \frac{1}{4}$  but not empty (transmitter mode), FIFO  $< \frac{1}{4}$  but not empty (receiver mode)

010:  $\frac{1}{4} < \text{FIFO} \leq \frac{1}{2}$  (transmitter mode),  $\frac{1}{4} \leq \text{FIFO} < \frac{1}{2}$  (receiver mode)

011:  $\frac{1}{2} < \text{FIFO} \leq \frac{3}{4}$  (transmitter mode),  $\frac{1}{2} \leq \text{FIFO} < \frac{3}{4}$  (receiver mode)

100:  $\frac{3}{4} < \text{FIFO}$  but not full (transmitter mode),  $\frac{3}{4} \leq \text{FIFO}$  but not full (receiver mode)

101: FIFO full (transmitter and receiver modes)

Others: Reserved

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 33.3.6: Frame synchronization](#) (configuration of FRL[7:0] bit in the SAI\_xFRCR register)

This bit is used only when the audio block operates in master mode (MODE[1] = 0) and NODIV = 0.

It can generate an interrupt if WCKCFGIE bit is set in SAI\_xIM register.

This flag is cleared when the software sets CWCKCFG bit in SAI\_xCLRFR register.

Bit 1 **MUTEDET**: Mute detection.

This bit is read only.

0: No MUTE detection on the SD input line

1: MUTE value detected on the SD input line (0 value) for a specified number of consecutive audio frame

This flag is set if consecutive 0 values are received in each slot of a given audio frame and for a consecutive number of audio frames (set in the MUTECNT bit in the SAI\_xCR2 register).

It can generate an interrupt if MUTEDETIE bit is set in SAI\_xIM register.

This flag is cleared when the software sets bit CMUTEDET in the SAI\_xCLRFR register.

Bit 0 **OVRUDR**: Overrun / underrun.

This bit is read only.

0: No overrun/underrun error.

1: Overrun/underrun error detection.

The overrun and underrun conditions can occur only when the audio block is configured as a receiver and a transmitter, respectively.

It can generate an interrupt if OVRUDRIE bit is set in SAI\_xIM register.

This flag is cleared when the software sets COVRUDR bit in SAI\_xCLRFR register.

### 33.5.8 SAI clear flag register (SAI\_ACLRFR)

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.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CLFSDET	CAFSDET	CCNRDY	Res.	CWCKCFG	CMUTEDET	COVRUDR								
									w	w	w		w	w	w

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **CLFSDET**: Clear late frame synchronization detection flag.

This bit is write only.

Programming this bit to 1 clears the LFSDET flag in the SAI\_xSR register.

This bit is not used in AC'97 or SPDIF mode

Reading this bit always returns the value 0.

Bit 5 **CAFSDET**: Clear anticipated frame synchronization detection flag.

This bit is write only.

Programming this bit to 1 clears the AFSDET flag in the SAI\_xSR register.

It is not used in AC'97 or SPDIF mode.

Reading this bit always returns the value 0.

Bit 4 **CCNRDY**: Clear Codec not ready flag.

This bit is write only.

Programming this bit to 1 clears the CNRDY flag in the SAI\_xSR register.

This bit is used only when the AC'97 audio protocol is selected in the SAI\_xCR1 register.

Reading this bit always returns the value 0.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **CWCKCFG**: Clear wrong clock configuration flag.

This bit is write only.

Programming this bit to 1 clears the WCKCFG flag in the SAI\_xSR register.

This bit is used only when the audio block is set as master (MODE[1] = 0) and NODIV = 0 in the SAI\_xCR1 register.

Reading this bit always returns the value 0.

Bit 1 **CMUTEDET**: Mute detection flag.

This bit is write only.

Programming this bit to 1 clears the MUTEDET flag in the SAI\_xSR register.

Reading this bit always returns the value 0.

Bit 0 **COVRUDR**: Clear overrun / underrun.

This bit is write only.

Programming this bit to 1 clears the OVRUDR flag in the SAI\_xSR register.

Reading this bit always returns the value 0.

### 33.5.9 SAI data register (SAI\_ADR)

Address offset: 0x20

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.

### 33.5.10 SAI configuration register 1 (SAI\_BCR1)

Address offset: 0x24

Reset value: 0x0000 0040

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MCKDIV[3:0]				NODIV	Res.	DMAEN	SAIEN
								rw	rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	OUTD RIV	MONO	SYNCEN[1:0]	CKSTR	LSBFIRST	DS[2:0]				Res.	PRTCFIG[1:0]	MODE[1:0]		
		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:20 MCKDIV[3:0]: Master clock divider

These bits are set and cleared by software. These bits are meaningless when the audio block operates in slave mode. They have to be configured when the audio block is disabled.

0000: Divides by 1 the master clock input.

Others: the master clock frequency is calculated accordingly to the following formula:

$$F_{SCK\_x} = \frac{F_{sai\_x\_ker\_ck}}{MCKDIV \times 2}$$

Bit 19 **NODIV**: No divider

This bit is set and cleared by software.

0: Master clock generator is enabled

1: No divider used in the clock generator (in this case Master Clock Divider bit has no effect)

Bit 18 Reserved, must be kept at reset value.

Bit 17 **DMAEN**: DMA enable

This bit is set and cleared by software.

0: DMA disabled

1: DMA enabled

*Note: Since the audio block defaults to operate as a transmitter after reset, the MODE[1:0] bits must be configured before setting DMAEN to avoid a DMA request in receiver mode.*

Bit 16 **SAIEN**: Audio block enable

This bit is set by software.

To switch off the audio block, the application software must program this bit to 0 and poll the bit till it reads back 0, meaning that the block is completely disabled. Before setting this bit to 1, check that it is set to 0, otherwise the enable command is not taken into account.

This bit enables to control the state of the SAI audio block. If it is disabled when an audio frame transfer is ongoing, the ongoing transfer completes and the cell is fully disabled at the end of this audio frame transfer.

0: SAI audio block disabled

1: SAI audio block enabled.

*Note: When the SAI block (A or B) is configured in master mode, the clock must be present on the SAI block input before setting SAIEN bit.*

Bits 15:14 Reserved, must be kept at reset value.

Bit 13 **OUTDRV**: Output drive

This bit is set and cleared by software.

0: Audio block output driven when SAIEN is set

1: Audio block output driven immediately after the setting of this bit.

*Note: This bit has to be set before enabling the audio block and after the audio block configuration.*

Bit 12 **MONO**: Mono mode

This bit is set and cleared by software. It is meaningful only when the number of slots is equal to 2. When the mono mode is selected, slot 0 data are duplicated on slot 1 when the audio block operates as a transmitter. In reception mode, the slot1 is discarded and only the data received from slot 0 are stored. Refer to [Section : Mono/stereo mode](#) for more details.

0: Stereo mode

1: Mono mode.

Bits 11:10 **SYNCEN[1:0]**: Synchronization enable

These bits are set and cleared by software. They must be configured when the audio subblock is disabled.

00: audio subblock in asynchronous mode.

01: audio subblock is synchronous with the other internal audio subblock. In this case, the audio subblock must be configured in slave mode

10: audio subblock is synchronous with an external SAI embedded peripheral. In this case the audio subblock must be configured in Slave mode.

11: Reserved

*Note: The audio subblock must be configured as asynchronous when SPDIF mode is enabled.*

Bit 9 **CKSTR**: Clock strobing edge

This bit is set and cleared by software. It must be configured when the audio block is disabled. This bit has no meaning in SPDIF audio protocol.

0: Signals generated by the SAI change on SCK rising edge, while signals received by the SAI are sampled on the SCK falling edge.

1: Signals generated by the SAI change on SCK falling edge, while signals received by the SAI are sampled on the SCK rising edge.

Bit 8 **LSBFIRST**: Least significant bit first

This bit is set and cleared by software. It must be configured when the audio block is disabled. This bit has no meaning in AC'97 audio protocol since AC'97 data are always transferred with the MSB first. This bit has no meaning in SPDIF audio protocol since in SPDIF data are always transferred with LSB first.

0: Data are transferred with MSB first

1: Data are transferred with LSB first

Bits 7:5 **DS[2:0]**: Data size

These bits are set and cleared by software. These bits are ignored when the SPDIF protocols are selected (bit PRTCFCFG[1:0]), because the frame and the data size are fixed in such case. When the companding mode is selected through COMP[1:0] bits, DS[1:0] are ignored since the data size is fixed to 8 bits by the algorithm.

These bits must be configured when the audio block is disabled.

000: Reserved

001: Reserved

010: 8 bits

011: 10 bits

100: 16 bits

101: 20 bits

110: 24 bits

111: 32 bits

Bit 4 Reserved, must be kept at reset value.

#### Bits 3:2 **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 enables 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.*

### 33.5.11 SAI configuration register 2 (SAI\_BCR2)

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
COMP[1:0]	CPL	MUTECNT[5:0]						MUTE VAL	MUTE	TRIS	F FLUSH	FTH[2:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	w	rw	rw	rw	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 is 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 Free protocol mode 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 is set and an interrupt is 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 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 must 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 must 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 must 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 must 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/interrupt must be disabled

Bits 2:0 **FTH[2:0]**: FIFO threshold.

This bit is set and cleared by software.

000: FIFO empty

001: ¼ FIFO

010: ½ FIFO

011: ¾ FIFO

100: FIFO full

101: Reserved

110: Reserved

111: Reserved

### 33.5.12 SAI frame configuration register (SAI\_BFRCR)

Address offset: 0x2C

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 is 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 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 must 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.

### 33.5.13 SAI slot register (SAI\_BSLOTR)

Address offset: 0x30

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 in the audio frame (including the number of inactive slots). The maximum number of slots is 16.

The number of slots must 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.

0000: Number of slots is 1.

0001: Number of slots is 2.

...

1111: Number of slots is 16.

**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 is undetermined.

Refer to [Output data line management on an inactive slot](#) for information on how to drive SD line.

These bits must be set when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

00: The slot size is equivalent to the data size (specified in DS[3:0] in the SAI\_xCR1 register).

01: 16-bit

10: 32-bit

11: Reserved

Bit 5 Reserved, must be kept at reset value.

**Bits 4:0 FBOFF[4:0]: First bit offset**

These bits are set and cleared by software.

The value set in this bitfield defines the position of the first data transfer bit in the slot. It represents an offset value. In transmission mode, the bits outside the data field are forced to 0. In reception mode, the extra received bits are discarded.

These bits must be set when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

**33.5.14 SAI interrupt mask register (SAI\_BIM)**

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.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LFSDETIE	AFSDETIE	CNRDYIE	FREQIE	WCKCFGIE	MUTEDETIE	OVRUDRIE								
									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 is 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 is 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 interrupt 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 interrupt 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 Free protocol 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.

### 33.5.15 SAI status register (SAI\_BSR)

Address offset: 0x38

Reset value: 0x0000 0008

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	FLVL[2:0]											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LFSDET	AFSDET	CNRDY	FREQ	WCKCFG	MUTEDET	OVRUDR								
									r	r	r	r	r	r	r

Bits 31:19 Reserved, must be kept at reset value.

Bits 18:16 **FLVL[2:0]**: FIFO level threshold.

This bit is read only. The FIFO level threshold flag is managed only by hardware and its setting depends on SAI block configuration (transmitter or receiver mode).

000: FIFO empty (transmitter and receiver modes)

001: FIFO  $\leq \frac{1}{4}$  but not empty (transmitter mode), FIFO  $< \frac{1}{4}$  but not empty (receiver mode)

010:  $\frac{1}{4} < \text{FIFO} \leq \frac{1}{2}$  (transmitter mode),  $\frac{1}{4} \leq \text{FIFO} < \frac{1}{2}$  (receiver mode)

011:  $\frac{1}{2} < \text{FIFO} \leq \frac{3}{4}$  (transmitter mode),  $\frac{1}{2} \leq \text{FIFO} < \frac{3}{4}$  (receiver mode)

100:  $\frac{3}{4} < \text{FIFO}$  but not full (transmitter mode),  $\frac{3}{4} \leq \text{FIFO}$  but not full (receiver mode)

101: FIFO full (transmitter and receiver modes)

Others: Reserved

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 33.3.6: Frame synchronization](#) (configuration of FRL[7:0] bit in the SAI\_xFRCR register)

This bit is used only when the audio block operates in master mode (MODE[1] = 0) and NODIV = 0.

It can generate an interrupt if WCKCFGIE bit is set in SAI\_xIM register.

This flag is cleared when the software sets CWCKCFG bit in SAI\_xCLRFR register.

Bit 1 **MUTEDET**: Mute detection.

This bit is read only.

0: No MUTE detection on the SD input line

1: MUTE value detected on the SD input line (0 value) for a specified number of consecutive audio frame

This flag is set if consecutive 0 values are received in each slot of a given audio frame and for a consecutive number of audio frames (set in the MUTECNT bit in the SAI\_xCR2 register).

It can generate an interrupt if MUTEDETIE bit is set in SAI\_xIM register.

This flag is cleared when the software sets bit CMUTEDET in the SAI\_xCLRFR register.

Bit 0 **OVRUDR**: Overrun / underrun.

This bit is read only.

0: No overrun/underrun error.

1: Overrun/underrun error detection.

The overrun and underrun conditions can occur only when the audio block is configured as a receiver and a transmitter, respectively.

It can generate an interrupt if OVRUDRIE bit is set in SAI\_xIM register.

This flag is cleared when the software sets COVRUDR bit in SAI\_xCLRFR register.

### 33.5.16 SAI clear flag register (SAI\_BCLRFR)

Address offset: 0x3C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CLFSDET	CAFSDET	CCNRDY	Res.	CWCKCFG	CMUTEDET	COVRUDR								
									w	w	w		w	w	w

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **CLFSDET**: Clear late frame synchronization detection flag.

This bit is write only.

Programming this bit to 1 clears the LFSDET flag in the SAI\_xSR register.

This bit is not used in AC'97 or SPDIF mode

Reading this bit always returns the value 0.

Bit 5 **CAFSDET**: Clear anticipated frame synchronization detection flag.

This bit is write only.

Programming this bit to 1 clears the AFSDET flag in the SAI\_xSR register.

It is not used in AC'97 or SPDIF mode.

Reading this bit always returns the value 0.

Bit 4 **CCNRDY**: Clear Codec not ready flag.

This bit is write only.

Programming this bit to 1 clears the CNRDY flag in the SAI\_xSR register.

This bit is used only when the AC'97 audio protocol is selected in the SAI\_xCR1 register.

Reading this bit always returns the value 0.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **CWCKCFG**: Clear wrong clock configuration flag.

This bit is write only.

Programming this bit to 1 clears the WCKCFG flag in the SAI\_xSR register.

This bit is used only when the audio block is set as master (MODE[1] = 0) and NODIV = 0 in the SAI\_xCR1 register.

Reading this bit always returns the value 0.

Bit 1 **CMUTEDET**: Mute detection flag.

This bit is write only.

Programming this bit to 1 clears the MUTEDET flag in the SAI\_xSR register.

Reading this bit always returns the value 0.

Bit 0 **COVRUDR**: Clear overrun / underrun.

This bit is write only.

Programming this bit to 1 clears the OVRUDR flag in the SAI\_xSR register.

Reading this bit always returns the value 0.

### 33.5.17 SAI data register (SAI\_BDR)

Address offset: 0x40

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.

### 33.5.18 SAI register map

**Table 220. SAI register map and reset values**

Refer to [Section 2.2: Memory organization](#) for the register boundary addresses.

## 34 SPDIF receiver interface (SPDIFRX)

### 34.1 SPDIFRX interface introduction

The SPDIFRX interface handles S/PDIF audio protocol.

### 34.2 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<sup>(a)</sup> supported
- Supports audio IEC-60958 and IEC-61937, consumer applications
- Parity bit management
- Communication using DMA for audio samples
- Communication using DMA for control and user channel information
- Interrupt capabilities

### 34.3 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 408* 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 APB 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

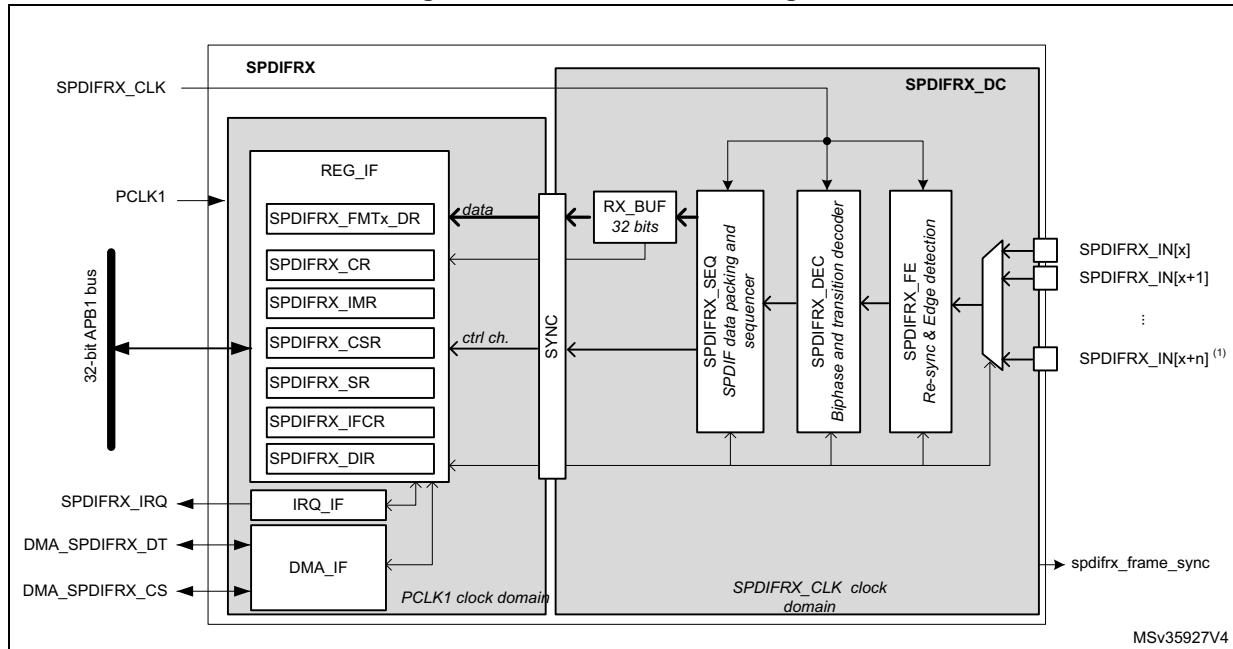
Interrupt services are also available either as an alternative function to the DMA, or for signaling error or key status of the peripheral.

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 is 50%, and the frequency equal to the frame rate.

This signal can be connected to timer events, in order to compute frequency drift.

a. Check the RCC capabilities in order to verify which sampling rates can be supported.

Figure 408. SPDIFRX block diagram



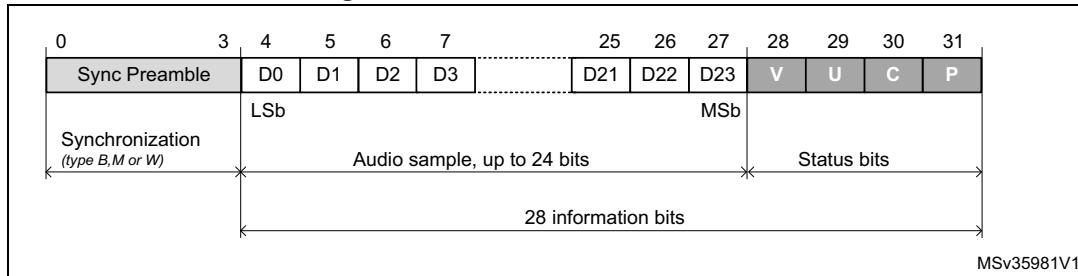
1. 'n' is fixed to 4, and 'x' is set to 1.

### 34.3.1 S/PDIF protocol (IEC-60958)

#### S/PDIF block

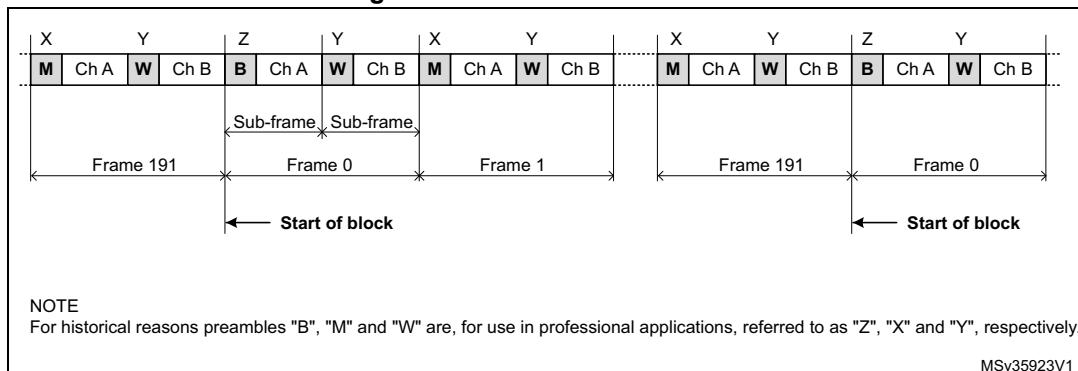
A S/PDIF frame is composed of two sub-frames (see [Figure 409](#)). Each sub-frame contains 32 bits (or time slots):

- 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 409. S/PDIF sub-frame format**

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".

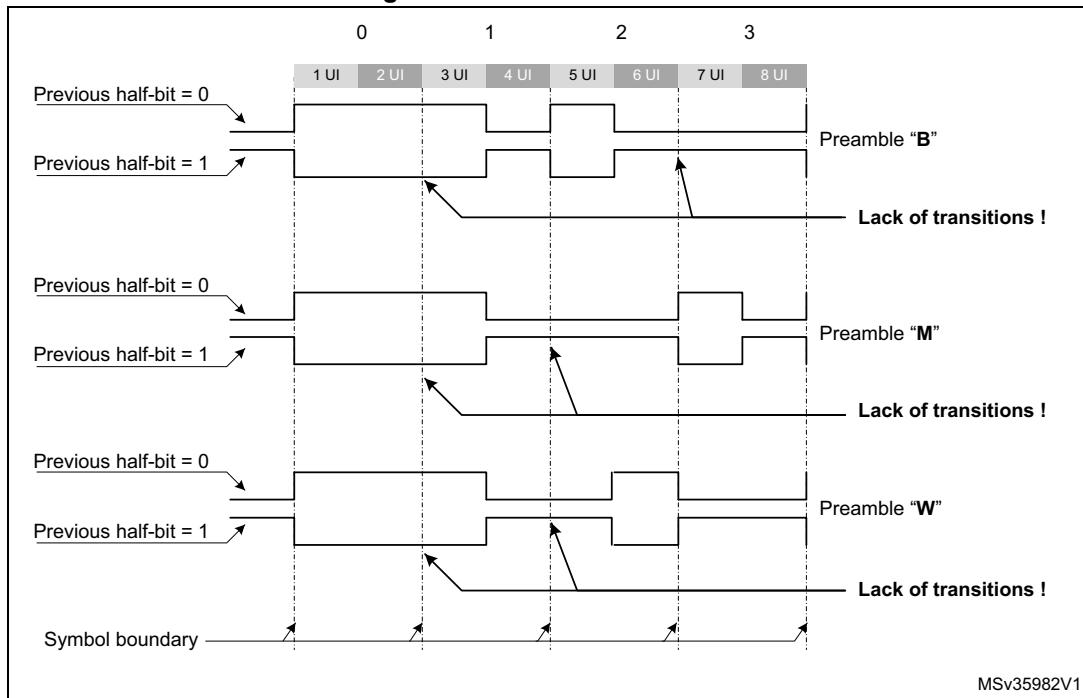
A S/PDIF block contains 192 pairs of sub-frames of 32 bits.

**Figure 410. 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 411](#).

Figure 411. S/PDIF Preambles



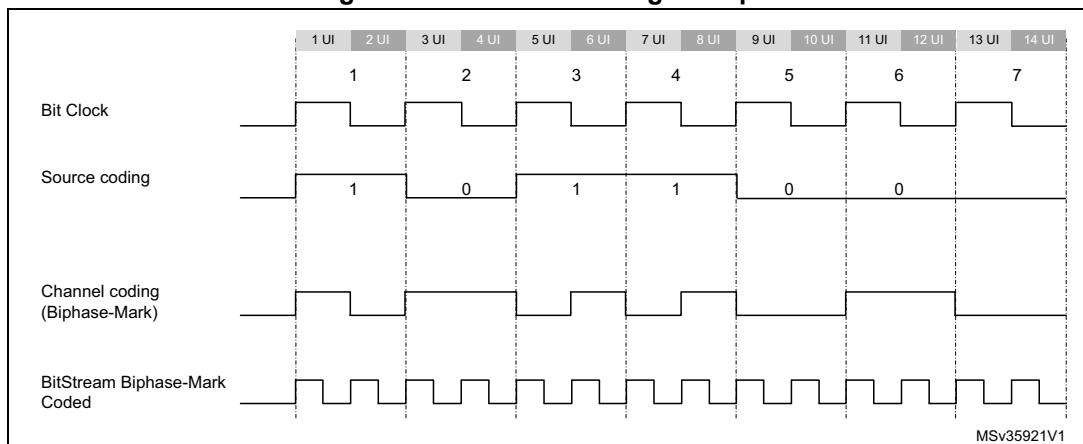
### Coding of information bits

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 412. Channel coding example



### 34.3.2 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:

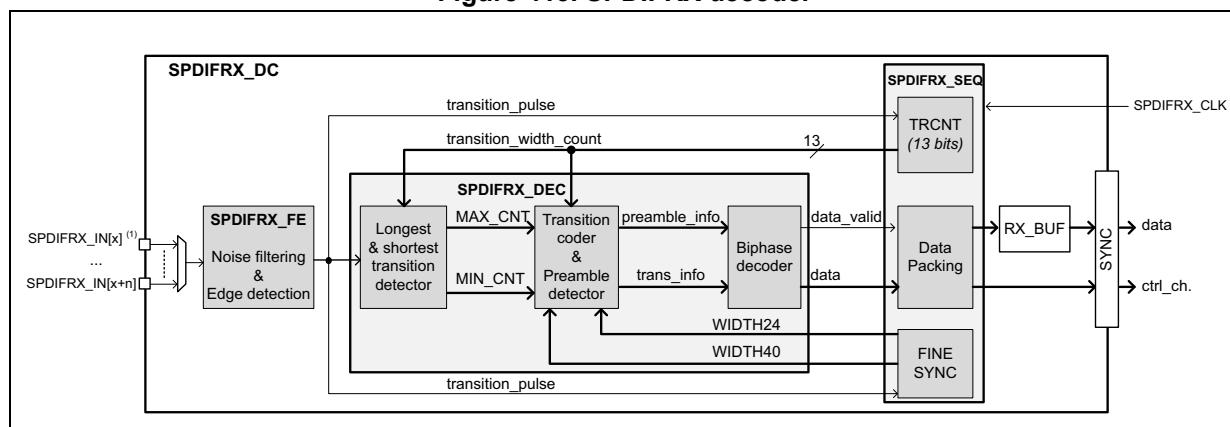
- The long time interval, having a duration of  $3 \times UI$ , noted TL. It appears only during preambles.
- The medium time interval, having a duration of  $2 \times UI$ , noted TM. It appears both in some preambles or into the information field.
- The short time interval, having a duration of  $1 \times UI$ , 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 413* gives a detailed view of the SPDIFRX decoder.

**Figure 413. SPDIFRX decoder**

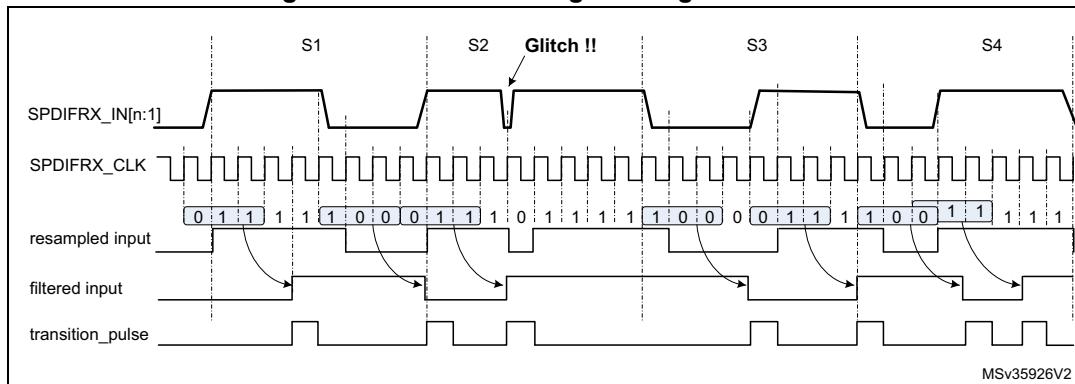


1. 'n' is fixed to 4, and 'x' is set to 1.

#### Noise filtering and 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 cancel spurs. This is performed by the stage detecting the edge transitions. The edge transitions are detected as follow:

- 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 414. 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 413](#)). 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

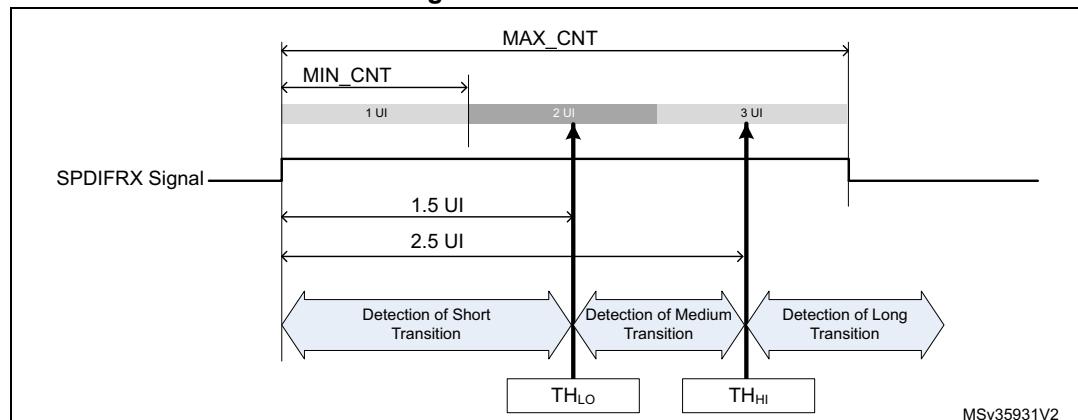
symbols (WIDTH24) for defining  $TH_{LO}$  and the length of 40 symbols (WIDTH40) for  $TH_{HI}$ .  $TH_{HI}$  and  $TH_{LO}$  are computed as follow:

- $TH_{LO} = (\text{WIDTH24}) / 32$
- $TH_{HI} = (\text{WIDTH40}) / 32$

This second synchronization phase is called the ‘fine synchronization’. Refer to [Figure 417](#) for additional information.

As shown in the figure hereafter,  $TH_{LO}$  is ideally equal to 1.5 UI, and to  $TH_{HI}$  2.5 UI.

**Figure 415. 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 221](#) 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 decode them.

**Table 221. 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 34.3.8: 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.

#### 34.3.3 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 is. 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.

#### 34.3.4 SPDIFRX synchronization

The synchronization phase starts when setting SPDIFRXEN to 01 or 11. [Figure 416](#) 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 SPDIFRX switches in COARSE SYNC mode only if activity is present on the selected SPDIFRX\_IN input, avoiding synchronization errors. See [Section 34.4: 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 can 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 is 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.

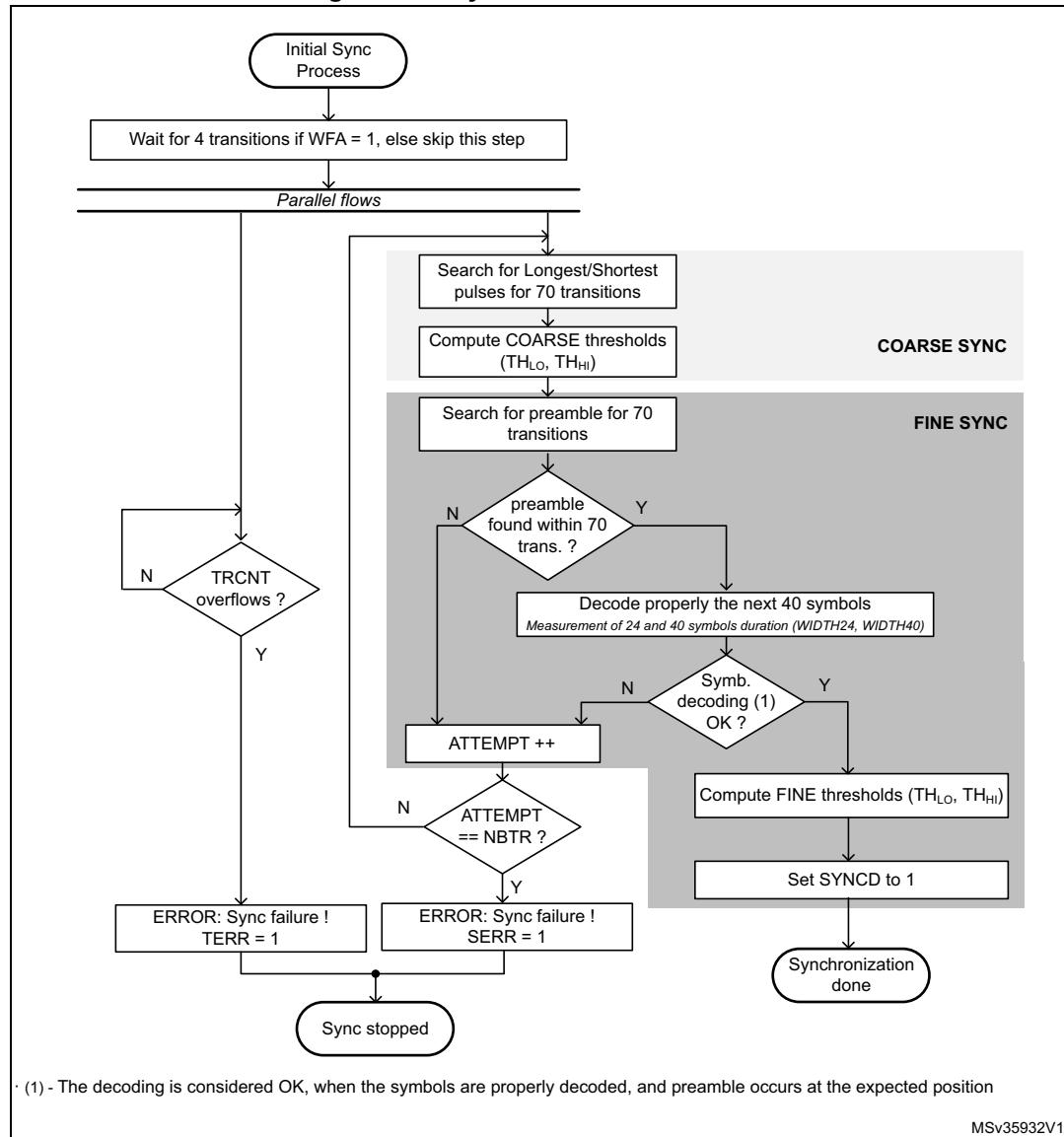
Two kinds of errors are detected:

- 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) starts when the next "B" preamble is detected (see [Figure 420](#)). 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 10 prior to start data reception. Note that DRFMT and RXSTEO cannot be modified when SPDIFRXEN = 11. Writes to these fields are ignored if SPDIFRXEN is

already 11, though these field can be changed with the same write instruction that causes SPDIFRXEN to become 11.  
Then the SPDIFRX waits for SPDIFRXEN = 11 and the “B” preamble before starting saving audio samples.

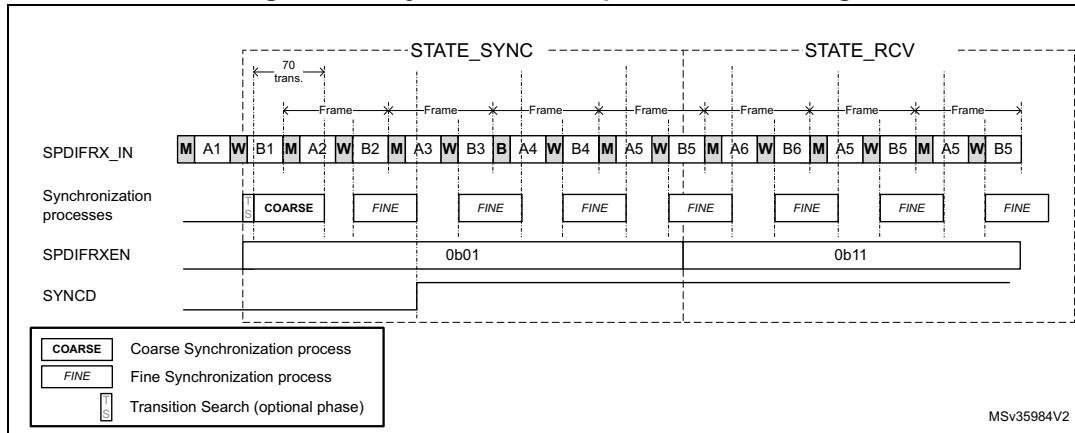
**Figure 416. Synchronization flowchart**



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 417](#) in order to continuously track S/PDIF synchronization.

Figure 417. Synchronization process scheduling



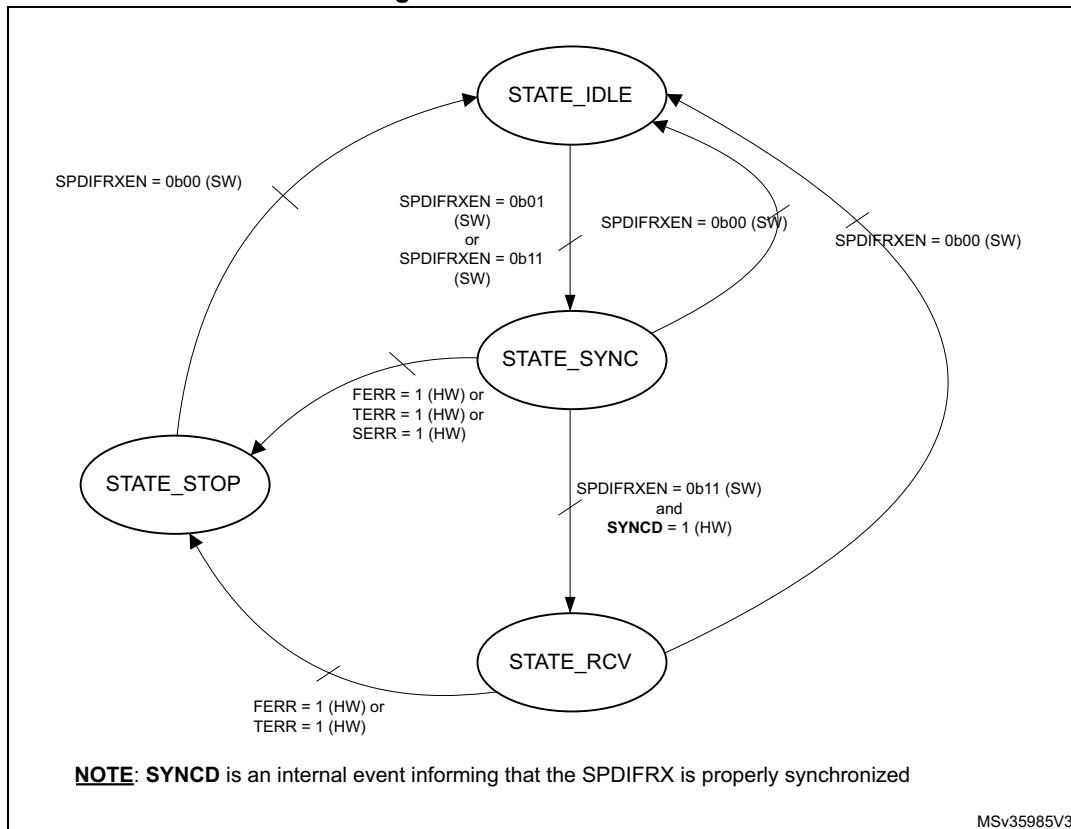
### 34.3.5 SPDIFRX handling

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 11, 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.

*Figure 418* 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 SPDIFRX control are followed by the mention "(HW)".

Figure 418. SPDIFRX States



When SPDIFRX is in STATE\_IDLE:

- The software can transition to STATE\_SYNC by setting SPDIFRXEN to 01 or 11

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 = 01 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 is properly completed.
- The SPDIFRX goes to STATE\_RCV if SPDIFRXEN = 11 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 is properly 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 SPDIFRX 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.

### 34.3.6 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\_FMTx\_DR register.

The valid data contained into the RX\_BUF are immediately transferred into SPDIFRX\_FMTx\_DR if SPDIFRX\_FMTx\_DR is empty.

The valid data contained into the RX\_BUF are transferred into SPDIFRX\_FMTx\_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 is completely received).
- The SPDIFRX\_FMTx\_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\_dma\_req/clr\_d) or interrupts for transferring the audio samples into the memory. The recommended option is DMA, refer to [Section 34.3.10: 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\_FMTx\_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 is 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).

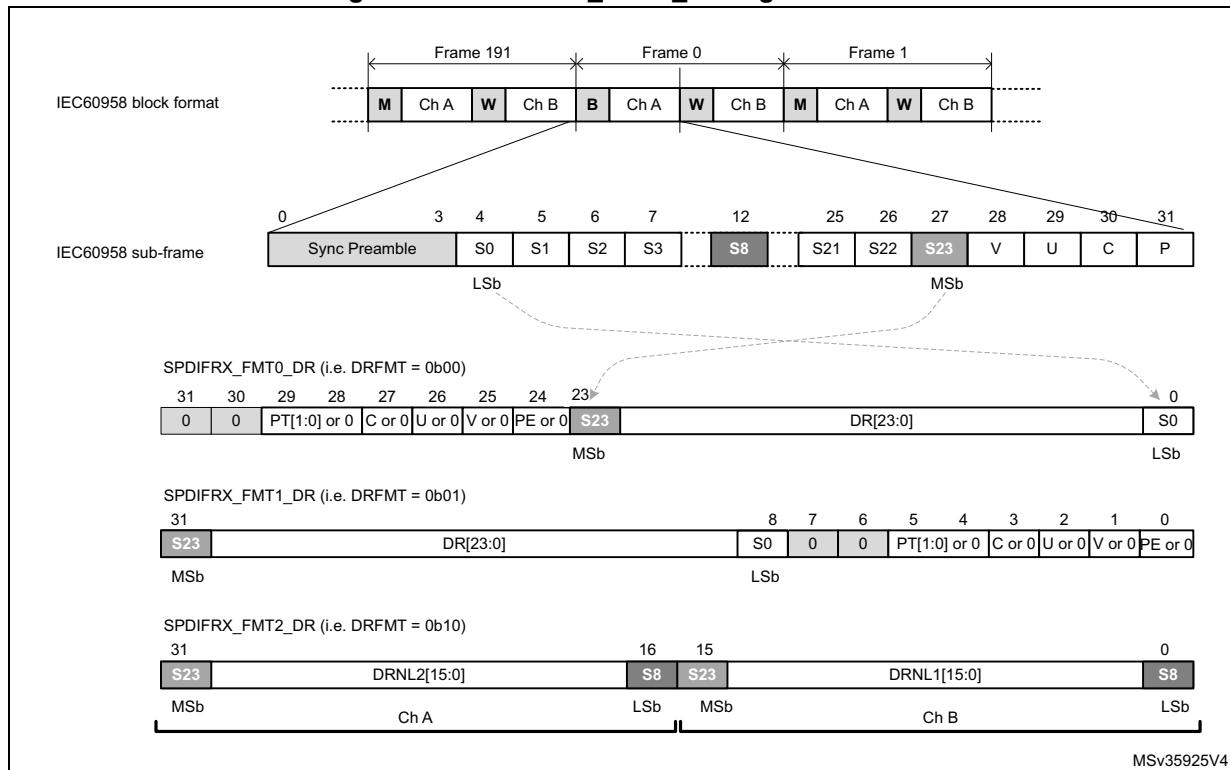
V, U and C are a direct copy of the value received from the S/PDIF interface.

The bit DRFMT allows the selection between 3 audio formats as shown in [Figure 419](#).

This document describes 3 data registers: SPDIFRX\_FMTx[2:0] ( $x = 2$  to  $0$ ), but in reality there is only one physical data register, having 3 possible formats:

- When DRFMT = 0, the format of the data register is the one described by SPDIFRX\_FMT0\_DR
- When DRFMT = 1, the format of the data register is the one described by SPDIFRX\_FMT1\_DR
- When DRFMT = 2, the format of the data register is the one described by SPDIFRX\_FMT2\_DR"

Figure 419. SPDIFRX\_FMTx\_DR register format



Setting DRFMT to 00 or 01, offers the possibility to have the data either right or left aligned into the SPDIFRX\_FMTx\_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= 10 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\_FMTx\_DR, dividing by two the amount of memory footprint. Note that when RXSTEO = 1, there is no misalignment risks (i.e. data from ChA are always stored into SPDIFRX\_FMTx\_DR[31:16]). If RXSTEO = 0, then there is a misalignment risk in case of overrun situation. In that case SPDIFRX\_FMTx\_DR[31:16] always contain the oldest value and SPDIFRX\_FMTx\_DR[15:0] the more recent value (see [Figure 421](#)).

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 34.3.7: 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 must be kept into the data register (SPDIFRX\_FMTx\_DR).

- When bit PMSK = 1, the parity error information is masked (set to 0), otherwise it is copied into SPDIFRX\_FMTx\_DR.
- When bit VMSK = 1, the validity information is masked (set to 0), otherwise it is copied into SPDIFRX\_FMTx\_DR.

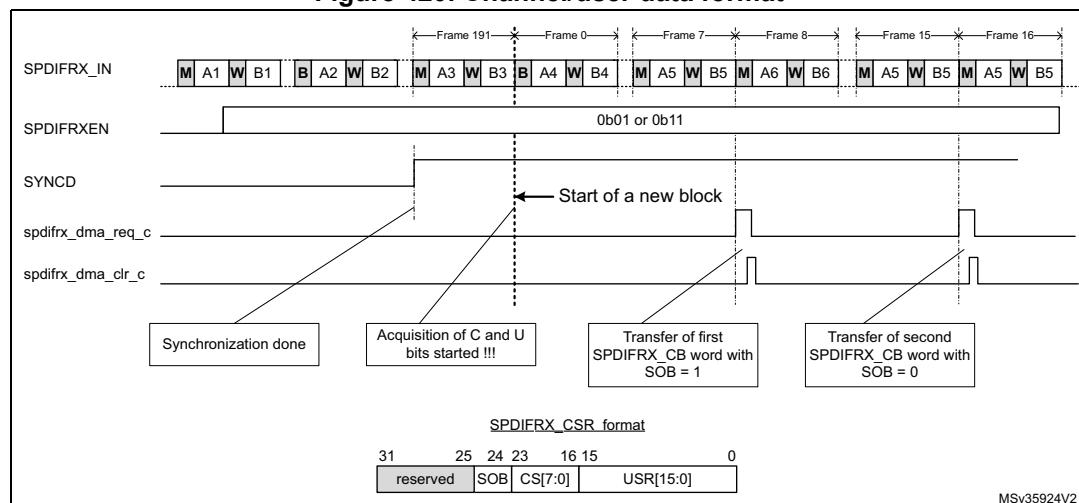
- When bit CUMSK = 1, the channel status, and used data information are masked (set to 0), otherwise they are copied into SPDIFRX\_FMTx\_DR.
- When bit PTMSK = 1, the preamble type is masked (set to 0), otherwise it is copied into SPDIFRX\_FMTx\_DR.

### 34.3.7 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 starts at the beginning of a IEC 60958 block. Two fields are available to control this path: CBDMAEN and SPDIFRXEN. When SPDIFRXEN is set to 01 or 0x11, the acquisition is started, after completion of the synchronization phase. When 8 channel status and 16 user data bits are 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 420](#)).

If CS[0] corresponds to the first bit of a new block, the bit SOB is set to 1. Refer to [Section 34.5.8: SPDIFRX 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 420. 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\_FMTx\_DR register does not affect this path.

### 34.3.8 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 01 or 11 in order to restart the SPDIFRX

Refer to [Figure 418](#) 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\_FMTx\_DR even if the interrupt is still pending.

The interrupt is acknowledged by clearing the PERR flag through PERRCF bit.

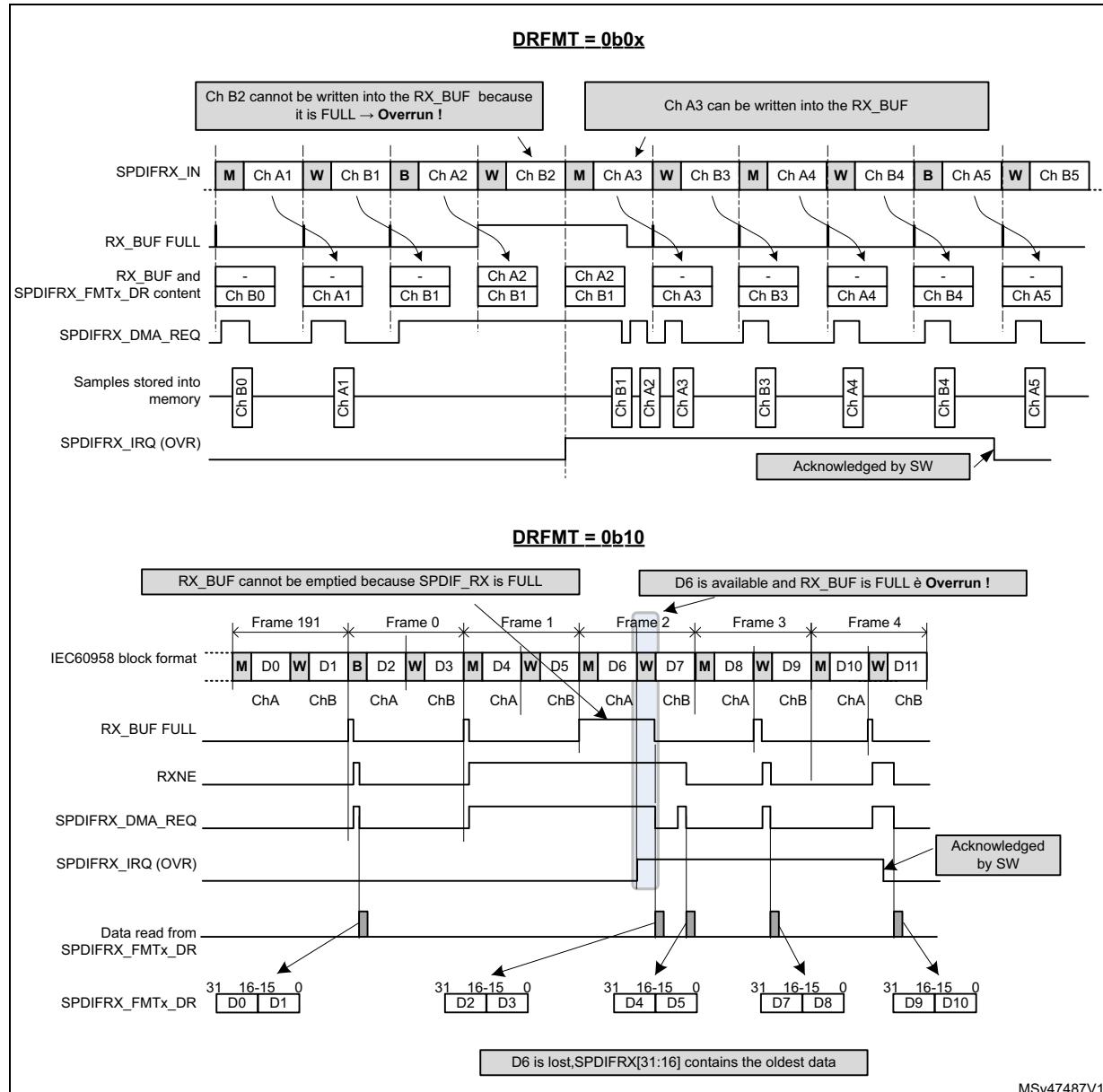
If the software wants to guarantee the coherency between the data read in the SPDIFRX\_FMTx\_DR register and the value of the bit PERR, the bit PMSK must be set to 0.

#### Overrun error

If both SPDIFRX\_FMTx\_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 SPDIFRX stores 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 421](#).

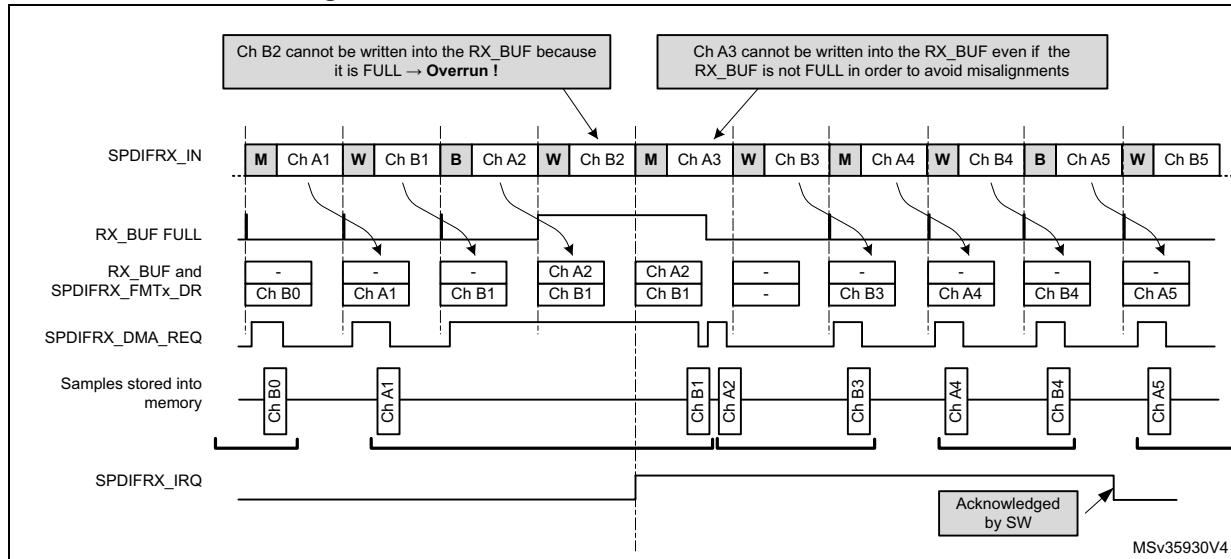
Figure 421. S/PDIF overrun error when RXSTEO = 0



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 422](#).

The OVR flag is cleared by software, by setting the OVRCF bit to 1.

Figure 422. S/PDIF overrun error when RXSTEO = 1



### 34.3.9 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 408](#)).

In order to decode properly the incoming S/PDIF stream the SPDIFRX\_DC must 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 must 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 222. Minimum SPDIFRX\_CLK frequency versus audio sampling rate<sup>(1)</sup>

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

1. Check the RCC capabilities in order to verify which sampling rates can be supported.

### 34.3.10 DMA interface

The SPDIFRX interface is able to perform communication using the DMA.

Note: *The user must 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\_FMTx\_DR is not empty, the SPDIFRX interface sends a transfer request to the DMA. The DMA reads the data received through the SPDIFRX\_FMTx\_DR register without CPU intervention.

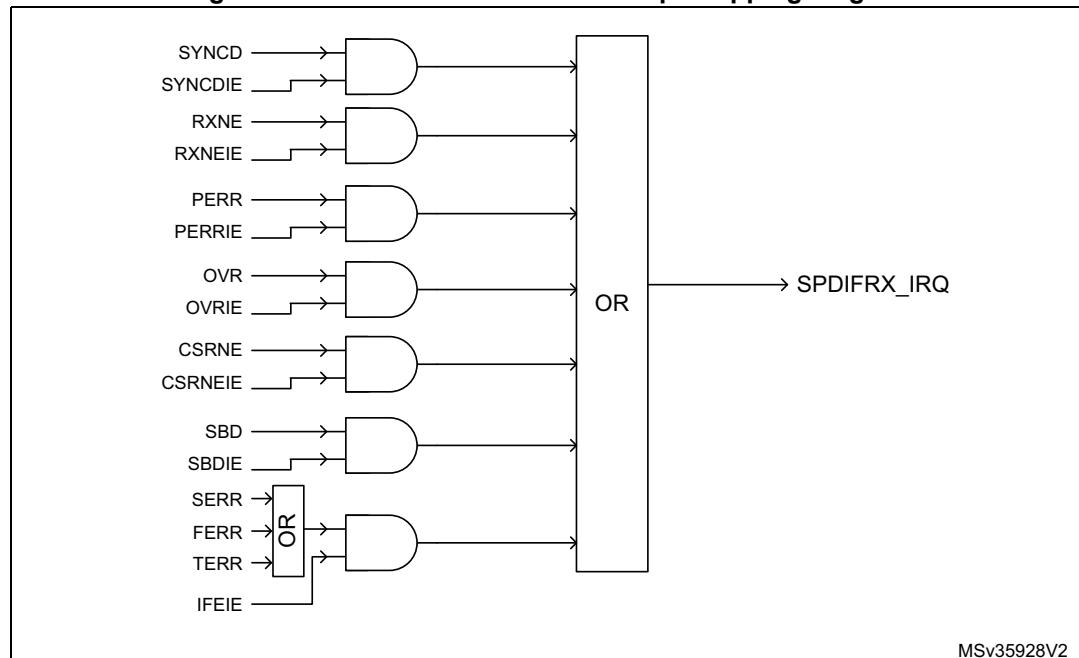
For the use of DMA for the control data refer to [Section 34.3.7: Dedicated control flow](#).

### 34.3.11 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 423. SPDIFRX interface interrupt mapping diagram**



MSv35928V2

### Clearing interrupt source

- RXNE is cleared when SPDIFRX\_FMTx\_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.*

### 34.3.12 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 223. Bit field property versus SPDIFRX state**

Registers	Field	SPDIFRXEN		
		00 (STATE_IDLE)	01 (STATE_SYNC)	11 (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 SPDIFRX is in STATE\_IDLE. In the others SPDIFRX 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 SPDIFRX already receives data.*

*Each of the mask bits (such as PMSK, VMSK) can be changed “on-the-fly” at any SPDIFRX state, but any change does not affect data which are already hold in SPDIFRX\_FMTx\_DR.*

## 34.4 Programming procedures

The following example illustrates a complete activation sequence of the SPDIFRX block. The data path and channel status and user information 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 focuses on detection. It is also possible to implement this function as follows:

- 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 as 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 01 or 11.

For those two implementations, the bit WFA is set to 0.

### 34.4.1 Initialization phase

- The initialization function looks 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\_FMTx\_DR register as source address for audio samples
  - Enable the generation of the SPDIFRX\_CLK. Refer to [Table 222](#) 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  $F_{SPDIFRX\_CLK}$  must 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 must 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 are read from sub-frame A)
  - DRFMT = 01 (data aligned to the left)
  - RXSTEO = 1 (expected stereo mode linear)
  - CBDMAEN = RXDMAEN = 1 (enable DMA channels)
  - SPDIFRXEN = 01 (switch SPDIFRX to STATE\_SYNC)
- The CPU can enter in WFI mode

Then the CPU receives interrupts coming either from DMA or SPDIFRX.

#### 34.4.2 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 SYNC is set to 1, then it means that the synchronization is 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 SYNC 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 re-tries are reached.
  - FERR indicates that the reading of information after synchronization fails (such as unexpected preamble, bad data decoding).
- If PERR is set to 1, it means that a parity error is 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 can 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.

#### 34.4.3 Handling of interrupts coming from DMA

If an interrupt comes from the DMA channel used of the channel status (SPDIFRX\_CSR):

If no error occurred (that is 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 10. Then the user can enable the data reception by setting SPDIFRXEN to 11.

The SOB bit, when set to 1 indicates the start of a new block. This information helps 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 always corresponds to the start of a new block.

If an interrupt comes from the DMA channel used of the audio samples (SPDIFRX\_FMTx\_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.

## 34.5 SPDIFRX interface registers

### 34.5.1 SPDIFRX control register (SPDIFRX\_CR)

Only 32-bit accesses are allowed in this register.

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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	INSEL[2:0]		
														rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	WFA	NBTR[1:0]		CHSEL	CBDMAEN	PTMSK	CUMSK	VMSK	PMSK	DRFMT[1:0]		RXSTEO	RXDMAEN	SPDIFRXEN[1:0]		
	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.

Bits 18:16 INSEL[2:0]: SPDIFRX input selection<sup>(1)</sup>

- 000: SPDIFRX\_IN1 selected
- 001: SPDIFRX\_IN2 selected
- 010: SPDIFRX\_IN3 selected
- 011: SPDIFRX\_IN4 selected
- other: reserved

Bit 15 Reserved, must be kept at reset value.

Bit 14 **WFA**: Wait for activity<sup>(1)</sup>

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<sup>(1)</sup>

00: No re-try is allowed (only one attempt)

01: 3 re-tries allowed

10: 15 re-tries allowed

11: 63 re-tries allowed

Bit 11 **CHSEL**: Channel selection<sup>(1)</sup>

This bit is set/reset by software.

1: The control flow takes the channel status from channel B.

0: The control flow takes the channel status from channel A.

Bit 10 **CBDMAEN**: Control buffer DMA enable for control flow<sup>(1)</sup>

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.

*Note: When this bit is set, the DMA request is made whenever the CSRNE flag is set.*

Bit 9 **PTMSK**: Mask of preamble type bits<sup>(1)</sup>

This bit is set/reset by software.

1: The preamble type bits are not copied into the SPDIFRX\_FMTx\_DR, zeros are written instead.

0: The preamble type bits are copied into the SPDIFRX\_FMTx\_DR.

Bit 8 **CUMSK**: Mask of channel status and user bits<sup>(1)</sup>

This bit is set/reset by software.

1: The channel status and user bits are not copied into the SPDIFRX\_FMTx\_DR, zeros are written instead.

0: The channel status and user bits are copied into the SPDIFRX\_FMTx\_DR.

Bit 7 **VMSK**: Mask of validity bit<sup>(1)</sup>

This bit is set/reset by software.

1: The validity bit is not copied into the SPDIFRX\_FMTx\_DR, a zero is written instead.

0: The validity bit is copied into the SPDIFRX\_FMTx\_DR.

Bit 6 **PMSK**: Mask parity error bit<sup>(1)</sup>

This bit is set/reset by software.

1: The parity error bit is not copied into the SPDIFRX\_FMTx\_DR, a zero is written instead.

0: The parity error bit is copied into the SPDIFRX\_FMTx\_DR.

Bits 5:4 **DRFMT[1:0]**: RX data format<sup>(1)</sup>

This bit is set/reset by software.

11: reserved

10: Data sample are packed by setting two 16-bit sample into a 32-bit word.

01: Data samples are aligned in the left (MSB)

00: Data samples are aligned in the right (LSB).

Bit 3 **RXSTEO**: Stereo mode<sup>(1)</sup>

This bit is set/reset by software.

1: The peripheral is in stereo mode.

0: The peripheral is in mono mode.

*Note: This bit is used in case of overrun situation in order to handle misalignment.*

Bit 2 **RXDMAEN**: Receiver DMA enable for data flow<sup>(1)</sup>

This bit is set/reset by software.

1: DMA mode is enabled for reception.

0: DMA mode is disabled for reception.

*Note: 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<sup>(1)</sup>

This field is modified by software.

It must be used to change the peripheral phase among the three possible states: STATE\_IDLE, STATE\_SYNC and STATE\_RCV.

00: Disable SPDIFRX (STATE\_IDLE).

01: Enable SPDIFRX synchronization only.

10: Reserved

11: Enable SPDIF receiver.

*Note: It is not possible to transition from STATE\_RCV to STATE\_SYNC, the user must first go the STATE\_IDLE.*

*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.*

- Refer to [Section 34.3.12: Register protection](#) for additional information on fields properties.

### 34.5.2 SPDIFRX interrupt mask register (SPDIFRX\_IMR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	IFE IE	SYNCD IE	SBLK IE	OVR IE	PERR IE	CSRNE IE	RXNE IE								
									rw	rw	rw	rw	rw	rw	rw

Bits 31:7 Reserved, must be kept at reset value.

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.

### 34.5.3 SPDIFRX status register (SPDIFRX\_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.	WIDTH5[14:0]														
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, must be kept at reset value.

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, must be kept at reset value.

**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: such as preamble not at the expected place, short transition not grouped by pairs.

This is set by the hardware only if the synchronization is 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\_IFCR register.

An interrupt is generated if SYNCDE = 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\_IFCR register.

An interrupt is generated if SBLKIE = 1 in the SPDIFRX\_IMR register.

0: No "B" preamble is detected.

1: "B" preamble is detected.

**Bit 3 OVR:** Overrun error

This bit is set by hardware when a received data is ready to be transferred in the SPDIFRX\_FMTx\_DR register while RXNE = 1 and both SPDIFRX\_FMTx\_DR and RX\_BUF are full.

This flag is cleared by writing a 1 to its corresponding bit on SPDIFRX\_IFCR 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\_FMTx\_DR register content is not lost but the last data received are.*

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\_IFCR register.

An interrupt is generated if PIE = 1 in the SPDIFRX\_IMR register.

0: No parity error

1: Parity error

Bit 1 **CSRNE**: Control buffer register 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\_FMTx\_DR register.

This flag is cleared by reading the SPDIFRX\_FMTx\_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.

**34.5.4 SPDIFRX interrupt flag clear register (SPDIFRX\_IFCR)**

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.	SYNCD CF	SBD CF	OVR CF	PERR CF	Res.	Res.									
										w	w	w	w		

Bits 31:6 Reserved, must be kept at reset value.

Bit 5 **SYNCDCF**: clears the synchronization done flag

Writing 1 in this bit clears the flag SYNCD in the SPDIFRX\_SR register.

Reading this bit always returns the value 0.

Bit 4 **SBCDF**: 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, must be kept at reset value.

### 34.5.5 SPDIFRX data input register (SPDIFRX\_FMT0\_DR)

Address offset: 0x10

Reset value: 0x0000 0000

This register can take three different formats according to DRFMT. Here is the format when DRFMT = 00:

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, must be kept at reset value.

Bits 29:28 **PT[1:0]:** 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[23:0]:** data value

Contains the 24 received data bits, aligned on D[23]

### 34.5.6 SPDIFRX data input register (SPDIFRX\_FMT1\_DR)

Address offset: 0x10

Reset value: 0x0000 0000

This register can take three different formats according to DRFMT. Here is the format when DRFMT = 01:

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[23:0]**: data value

Contains the 24 received data bits, aligned on D[23]

Bits 7:6 Reserved, must be kept at reset value.

Bits 5:4 **PT[1:0]**: 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

### 34.5.7 SPDIFRX data input register (SPDIFRX\_FMT2\_DR)

Address offset: 0x10

Reset value: 0x0000 0000

This register can take 3 different formats according to DRFMT.

The data format proposed when DRFMT = 10, 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[15:0]**: data value

This field contains the channel A

Bits 15:0 **DRNL1[15:0]**: data value

This field contains the channel B

### 34.5.8 SPDIFRX channel status register (SPDIFRX\_CSR)

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.	SOB	CS[7:0]							
							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
USR[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.

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.

### 34.5.9 SPDIFRX debug information register (SPDIFRX\_DIR)

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.	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
THI[12:0]															
			r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:29 Reserved, must be kept at reset value.

Bits 28:16 **TLO[12:0]**: 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 SYNCD goes high, and then every frame.

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:0 **THI[12:0]**: 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:

$$\text{Sampling Rate} = [2 \times THI \times T_{SPDIFRX\_CLK} +/ - T_{SPDIFRX\_CLK}] \times 2/5$$

Note that THI is updated by the hardware when SYNCND goes high, and then every frame.

### 34.5.10 SPDIFRX interface register map

Table 224. SPDIFRX interface register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5			
0x00	SPDIFRX_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	INSEL[2:0]	NBTR[1:0]	Res.	Res.	WFA	Res.	Res.	CHSEL	CBDMAEN	PTMSK	CUMSK	VMSK	PMSK	SyncDIE			
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x04	SPDIFRX_IMR	Res.	Res.	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	SPDIFRX_SR	Res.	WIDTH5[14:0]														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			
0x0C	SPDIFRX_IFCR	Res.	Res.	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	SPDIFRX_FMT0_DR	Res.	Res.	PT[1:0]	C	U	V	P	E	DR[23:0]															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		
0x10	SPDIFRX_FMT1_DR	DR[23:0]															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		
0x10	SPDIFRX_FMT2_DR	DRNL2[15:0]															DRNL1[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		
0x14	SPDIFRX_CSR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SOB	CS[7:0]							USR[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	
0x18	SPDIFRX_DIR	Res.	Res.	Res.	Res.	0	0	0	0	0	TLO[12:0]							THI[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		

Refer to [Section 2.2](#) for the register boundary addresses.

## 35 SD/SDIO/MMC card host interface (SDMMC)

### 35.1 SDMMC main features

The SD/SDIO MMC card host interface (SDMMC) provides an interface between the APB2 peripheral bus and MultiMediaCards (MMCs), SD memory cards and SDIO cards.

The MultiMediaCard system specifications are available through the MultiMediaCard Association website, published by the MMCA technical committee.

SD memory card and SD I/O card system specifications are available through the SD card Association website.

The SDMMC features include the following:

- Full compliance with *MultiMediaCard System Specification Version 4.2*. Card support for three different databus modes: 1-bit (default), 4-bit and 8-bit
- Full compatibility with previous versions of MultiMediaCards (forward compatibility)
- Full compliance with *SD Memory Card Specifications Version 2.0*
- Full compliance with *SD I/O Card Specification Version 2.0*: card support for two different databus modes: 1-bit (default) and 4-bit
- Data transfer up to 50 MHz for the 8 bit mode
- Data and command output enable signals to control external bidirectional drivers.

- Note: 1 *The SDMMC does not have an SPI-compatible communication mode.*
- 2 *The SD memory card protocol is a superset of the MultiMediaCard protocol as defined in the MultiMediaCard system specification V2.11. Several commands required for SD memory devices are not supported by either SD I/O-only cards or the I/O portion of combo cards. Some of these commands have no use in SD I/O devices, such as erase commands, and thus are not supported in the SDIO protocol. In addition, several commands are different between SD memory cards and SD I/O cards and thus are not supported in the SDIO protocol. For details refer to SD I/O card Specification Version 1.0.*

The MultiMediaCard/SD bus connects cards to the controller.

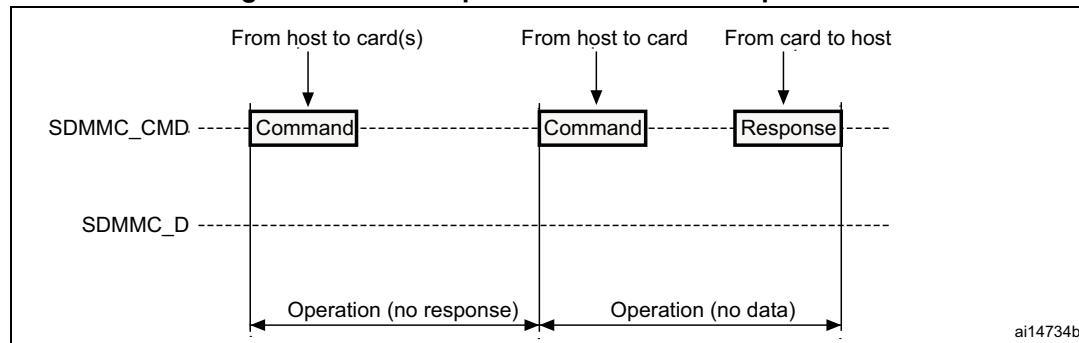
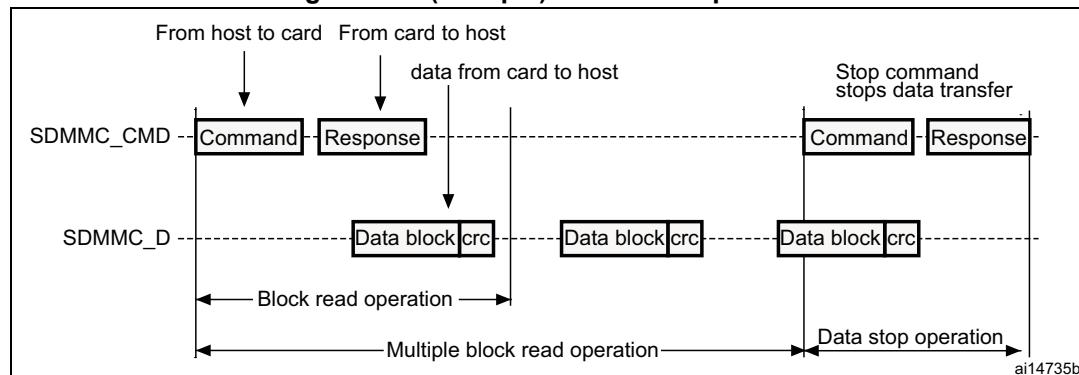
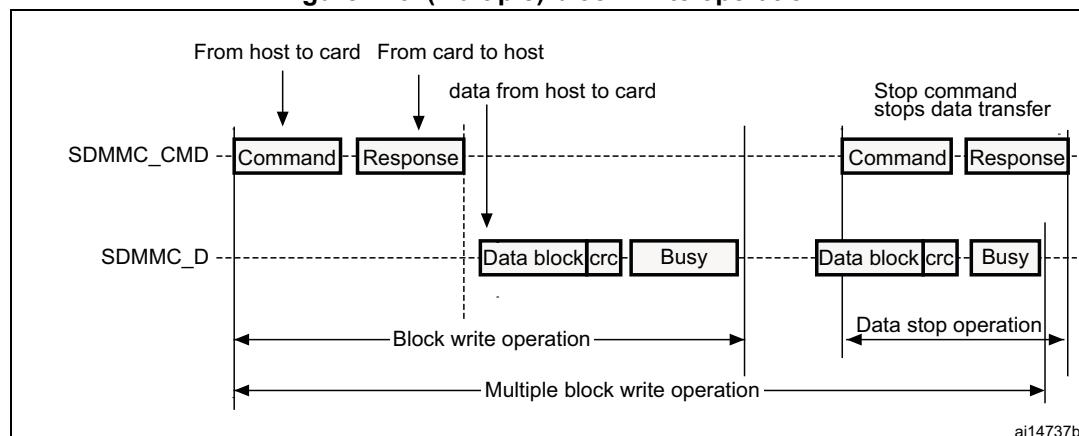
The current version of the SDMMC supports only one SD/SDIO/MMC4.2 card at any one time and a stack of MMC4.1 or previous.

### 35.2 SDMMC bus topology

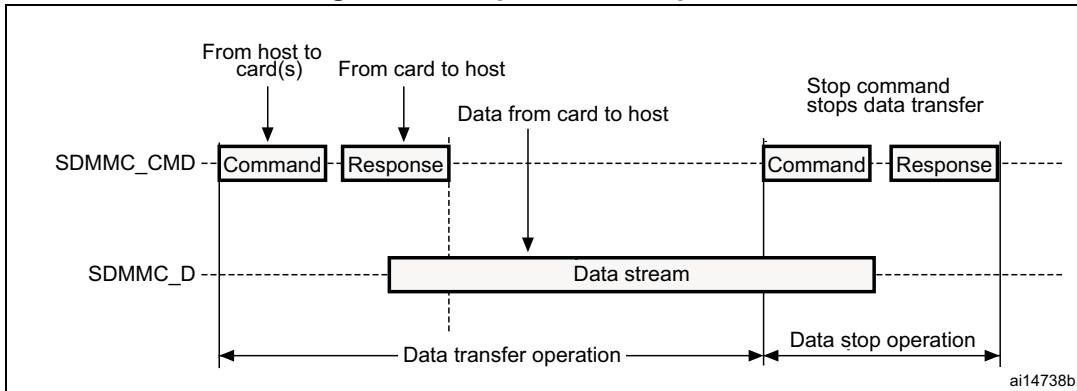
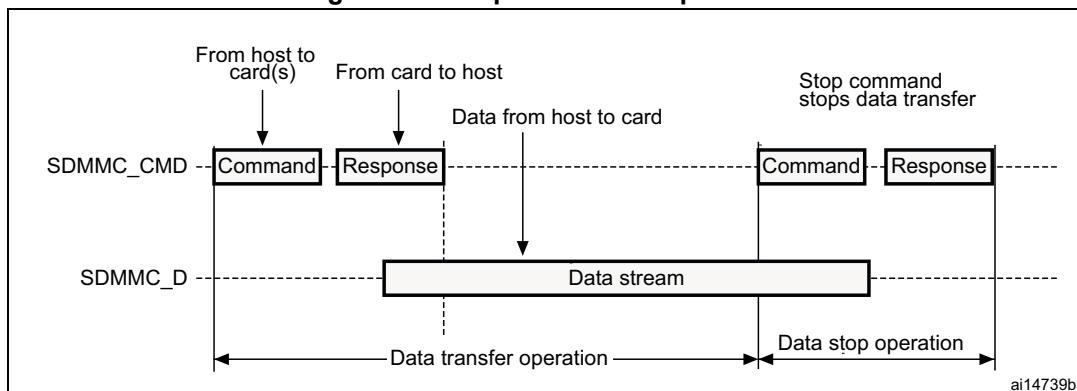
Communication over the bus is based on command and data transfers.

The basic transaction on the MultiMediaCard/SD/SD I/O bus is the command/response transaction. These types of bus transaction transfer their information directly within the command or response structure. In addition, some operations have a data token.

Data transfers to/from SD/SDIO memory cards are done in data blocks. Data transfers to/from MMC are done data blocks or streams.

**Figure 424. “No response” and “no data” operations****Figure 425. (Multiple) block read operation****Figure 426. (Multiple) block write operation**

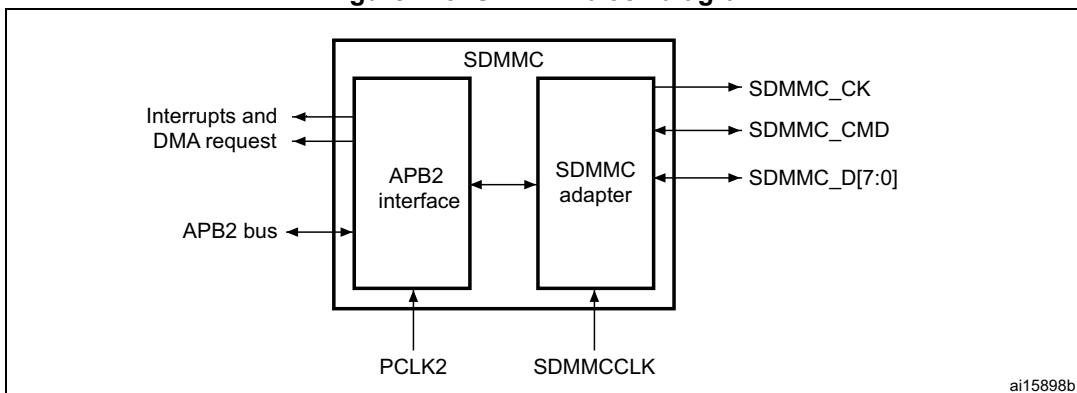
**Note:** The SDMMC will not send any data as long as the Busy signal is asserted (SDMMC\_D0 pulled low).

**Figure 427. Sequential read operation****Figure 428. Sequential write operation**

### 35.3 SDMMC functional description

The SDMMC consists of two parts:

- The SDMMC adapter block provides all functions specific to the MMC/SD/SD I/O card such as the clock generation unit, command and data transfer.
- The APB2 interface accesses the SDMMC adapter registers, and generates interrupt and DMA request signals.

**Figure 429. SDMMC block diagram**

By default SDMMC\_D0 is used for data transfer. After initialization, the host can change the databus width.

If a MultiMediaCard is connected to the bus, SDMMC\_D0, SDMMC\_D[3:0] or SDMMC\_D[7:0] can be used for data transfer. MMC V3.31 or previous, supports only 1 bit of data so only SDMMC\_D0 can be used.

If an SD or SD I/O card is connected to the bus, data transfer can be configured by the host to use SDMMC\_D0 or SDMMC\_D[3:0]. All data lines are operating in push-pull mode.

**SDMMC\_CMD** has two operational modes:

- Open-drain for initialization (only for MMCV3.31 or previous)
- Push-pull for command transfer (SD/SD I/O card MMC4.2 use push-pull drivers also for initialization)

**SDMMC\_CK** is the clock to the card: one bit is transferred on both command and data lines with each clock cycle.

The SDMMC uses two clock signals:

- SDMMC adapter clock SDMMCCLK = 50 MHz)
- APB2 bus clock (PCLK2)

PCLK2 and SDMMC\_CK clock frequencies must respect the following condition:

$$\text{Frequenc(PCLK2)} > ((3 \times \text{Width}) / 32) \times \text{Frequency(SDMMC_CK)}$$

The signals shown in [Table 225](#) are used on the MultiMediaCard/SD/SD I/O card bus.

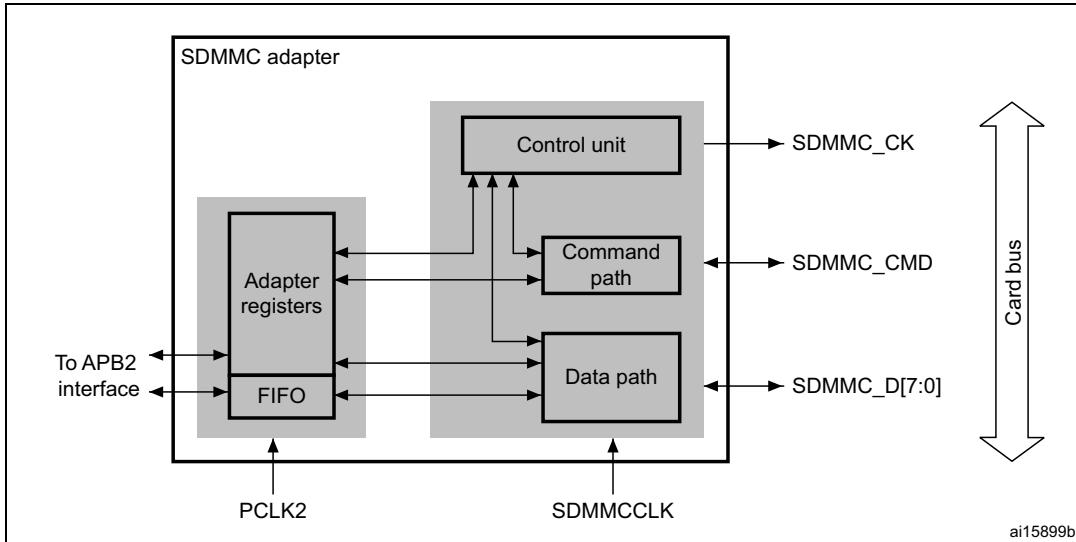
**Table 225. SDMMC I/O definitions**

Pin	Direction	Description
SDMMC_CK	Output	MultiMediaCard/SD/SDIO card clock. This pin is the clock from host to card.
SDMMC_CMD	Bidirectional	MultiMediaCard/SD/SDIO card command. This pin is the bidirectional command/response signal.
SDMMC_D[7:0]	Bidirectional	MultiMediaCard/SD/SDIO card data. These pins are the bidirectional databus.

### 35.3.1 SDMMC adapter

*Figure 430* shows a simplified block diagram of an SDMMC adapter.

**Figure 430. SDMMC adapter**



The SDMMC adapter is a multimedia/secure digital memory card bus master that provides an interface to a multimedia card stack or to a secure digital memory card. It consists of five subunits:

- Adapter register block
- Control unit
- Command path
- Data path
- Data FIFO

*Note:* The adapter registers and FIFO use the APB2 bus clock domain (PCLK2). The control unit, command path and data path use the SDMMC adapter clock domain (SDMMCCLK).

#### Adapter register block

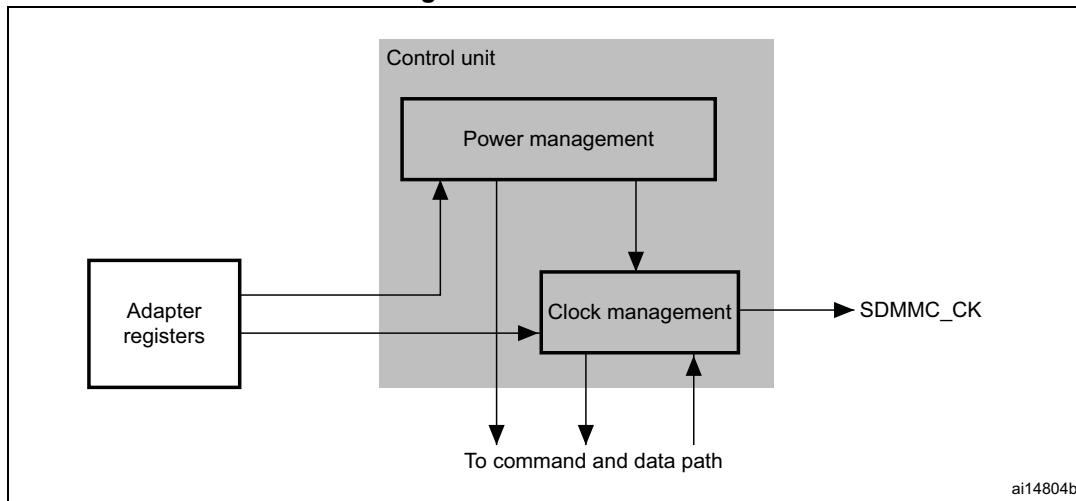
The adapter register block contains all system registers. This block also generates the signals that clear the static flags in the multimedia card. The clear signals are generated when 1 is written into the corresponding bit location in the SDMMC Clear register.

#### Control unit

The control unit contains the power management functions and the clock divider for the memory card clock.

There are three power phases:

- power-off
- power-up
- power-on

**Figure 431. Control unit**

The control unit is illustrated in [Figure 431](#). It consists of a power management subunit and a clock management subunit.

The power management subunit disables the card bus output signals during the power-off and power-up phases.

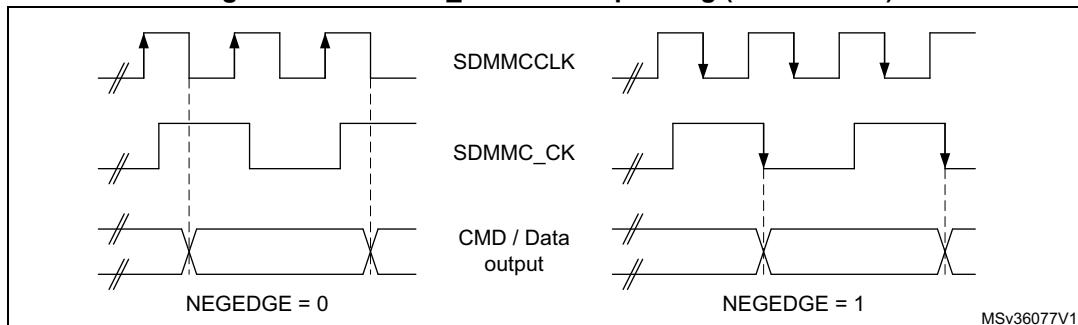
The clock management subunit generates and controls the SDMMC\_CK signal. The SDMMC\_CK output can use either the clock divide or the clock bypass mode. The clock output is inactive:

- after reset
- during the power-off or power-up phases
- if the power saving mode is enabled and the card bus is in the Idle state (eight clock periods after both the command and data path subunits enter the Idle phase)

The clock management subunit controls SDMMC\_CK dephasing. When not in bypass mode the SDMMC command and data output are generated on the SDMMCCCLK falling edge succeeding the rising edge of SDMMC\_CK. (SDMMC\_CK rising edge occurs on SDMMCCCLK rising edge) when SDMMC\_CLKCR[13] bit is reset (NEGEDGE = 0). When SDMMC\_CLKCR[13] bit is set (NEGEDGE = 1) SDMMC command and data changed on the SDMMC\_CK falling edge.

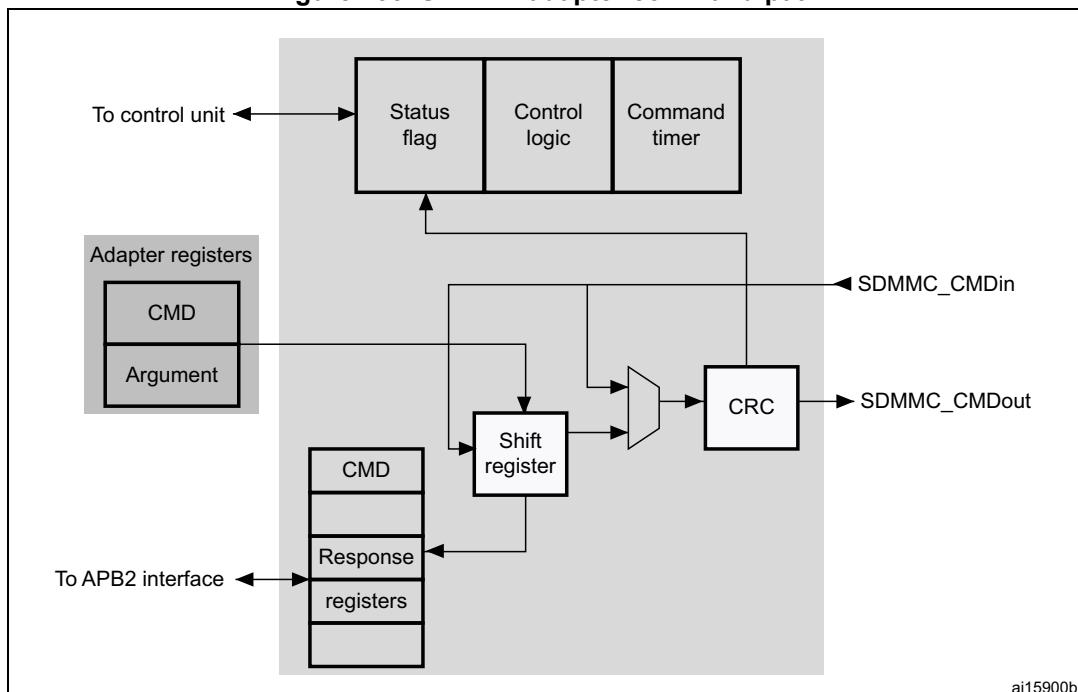
When SDMMC\_CLKCR[10] is set (BYPASS = 1), SDMMC\_CK rising edge occurs on SDMMCCCLK rising edge. The data and the command change on SDMMCCCLK falling edge whatever NEGEDGE value.

The data and command responses are latched using SDMMC\_CK rising edge.

**Figure 432. SDMMC\_CK clock dephasing (BYPASS = 0)**

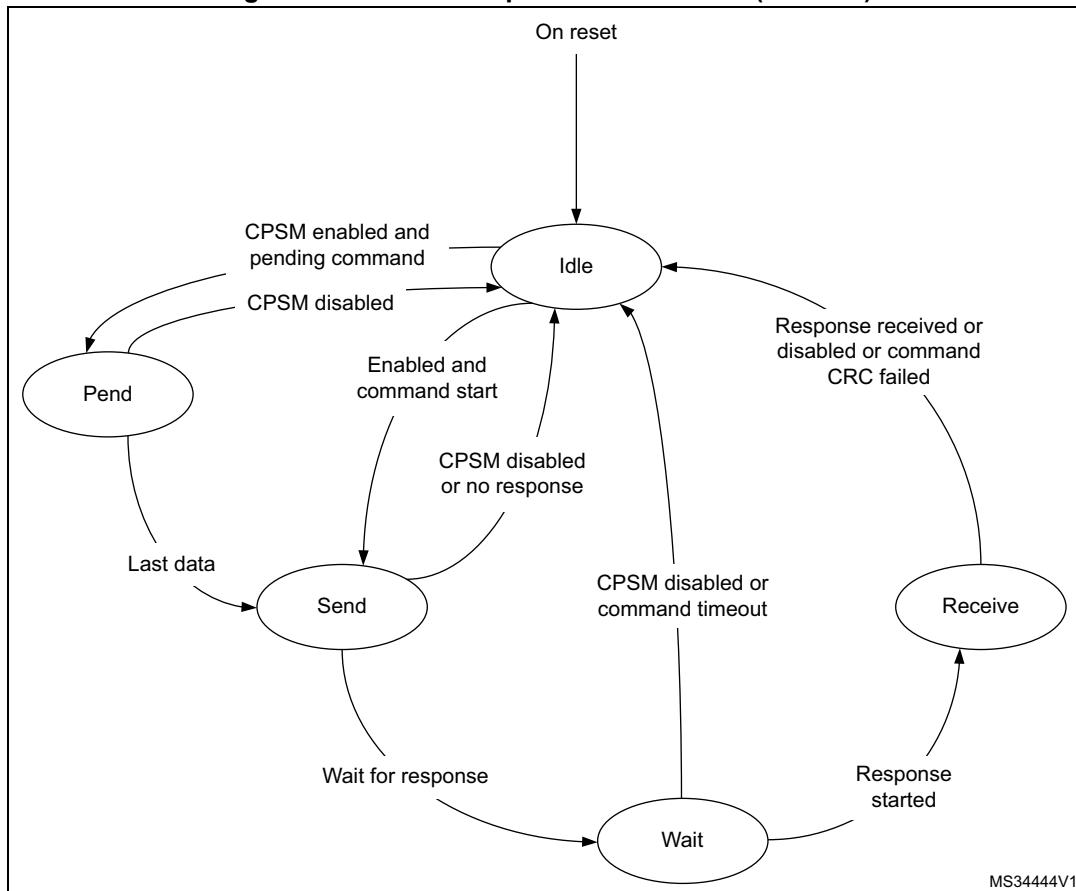
### Command path

The command path unit sends commands to and receives responses from the cards.

**Figure 433. SDMMC 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 434 on page 1226](#)). When the response is received, the received CRC code and the internally generated code are compared, and the appropriate status flags are set.

Figure 434. Command path state machine (SDMMC)



When the Wait state is entered, the command timer starts running. If the timeout is reached before the CPSM moves to the Receive state, the timeout flag is set and the Idle state is entered.

Note:

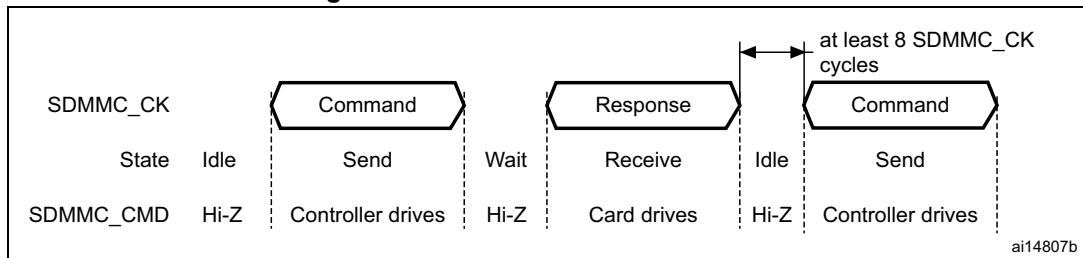
*The command timeout has a fixed value of 64 SDMMC\_CK clock periods.*

If the interrupt bit is set in the command register, the timer is disabled and the CPSM waits for an interrupt request from one of the cards. If a pending bit is set in the command register, the CPSM enters the Pend state, and waits for a CmdPend signal from the data path subunit. When CmdPend is detected, the CPSM moves to the Send state. This enables the data counter to trigger the stop command transmission.

Note:

*The CPSM remains in the Idle state for at least eight SDMMC\_CK periods to meet the  $N_{CC}$  and  $N_{RC}$  timing constraints.  $N_{CC}$  is the minimum delay between two host commands, and  $N_{RC}$  is the minimum delay between the host command and the card response.*

Figure 435. SDMMC 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 226](#).

The command path operates in a half-duplex mode, so that commands and responses can either be sent or received. If the CPSM is not in the Send state, the SDMMC\_CMD output is in the Hi-Z state, as shown in [Figure 435 on page 1227](#). Data on SDMMC\_CMD are synchronous with the rising edge of SDMMC\_CK. [Table 226](#) shows the command format.

Table 226. Command format

Bit position	Width	Value	Description
47	1	0	Start bit
46	1	1	Transmission bit
[45:40]	6	-	Command index
[39:8]	32	-	Argument
[7:1]	7	-	CRC7
0	1	1	End bit

- Response: a response is a token that is sent from an addressed card (or synchronously from all connected cards for MMC V3.31 or previous), to the host as an answer to a previously received command. Responses are transferred serially on the CMD line.

The SDMMC supports two response types. Both use CRC error checking:

- 48 bit short response
- 136 bit long response

Note: *If the response does not contain a CRC (CMD1 response), the device driver must ignore the CRC failed status.*

**Table 227. 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 228. 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 35.8.4 on page 1264](#)). The command path implements the status flags shown in [Table 229](#):

**Table 229. 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:

$$\text{CRC}[6:0] = \text{Remainder } [(M(x) * x^7) / G(x)]$$

$$G(x) = x^7 + x^3 + 1$$

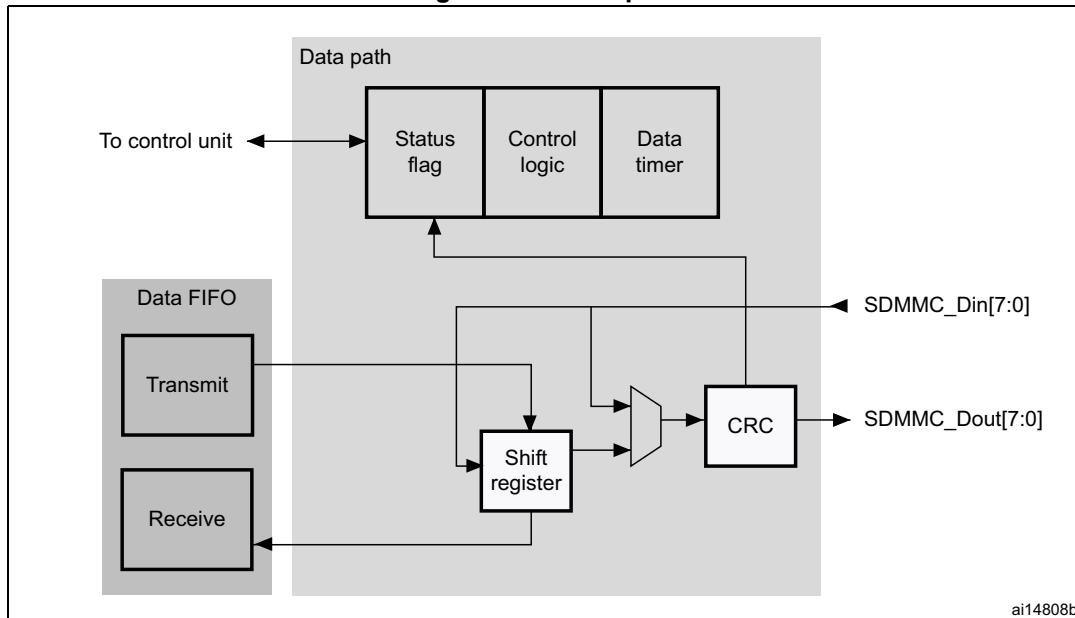
$$M(x) = (\text{start bit}) * x^{39} + \dots + (\text{last bit before CRC}) * x^0, \text{ or}$$

$$M(x) = (\text{start bit}) * x^{119} + \dots + (\text{last bit before CRC}) * x^0$$

## Data path

The data path subunit transfers data to and from cards. [Figure 436](#) shows a block diagram of the data path.

**Figure 436. Data path**



The card databus width can be programmed using the clock control register. If the 4-bit wide bus mode is enabled, data is transferred at four bits per clock cycle over all four data signals (SDMMC\_D[3:0]). If the 8-bit wide bus mode is enabled, data is transferred at eight bits per clock cycle over all eight data signals (SDMMC\_D[7:0]). If the wide bus mode is not enabled, only one bit per clock cycle is transferred over SDMMC\_D0.

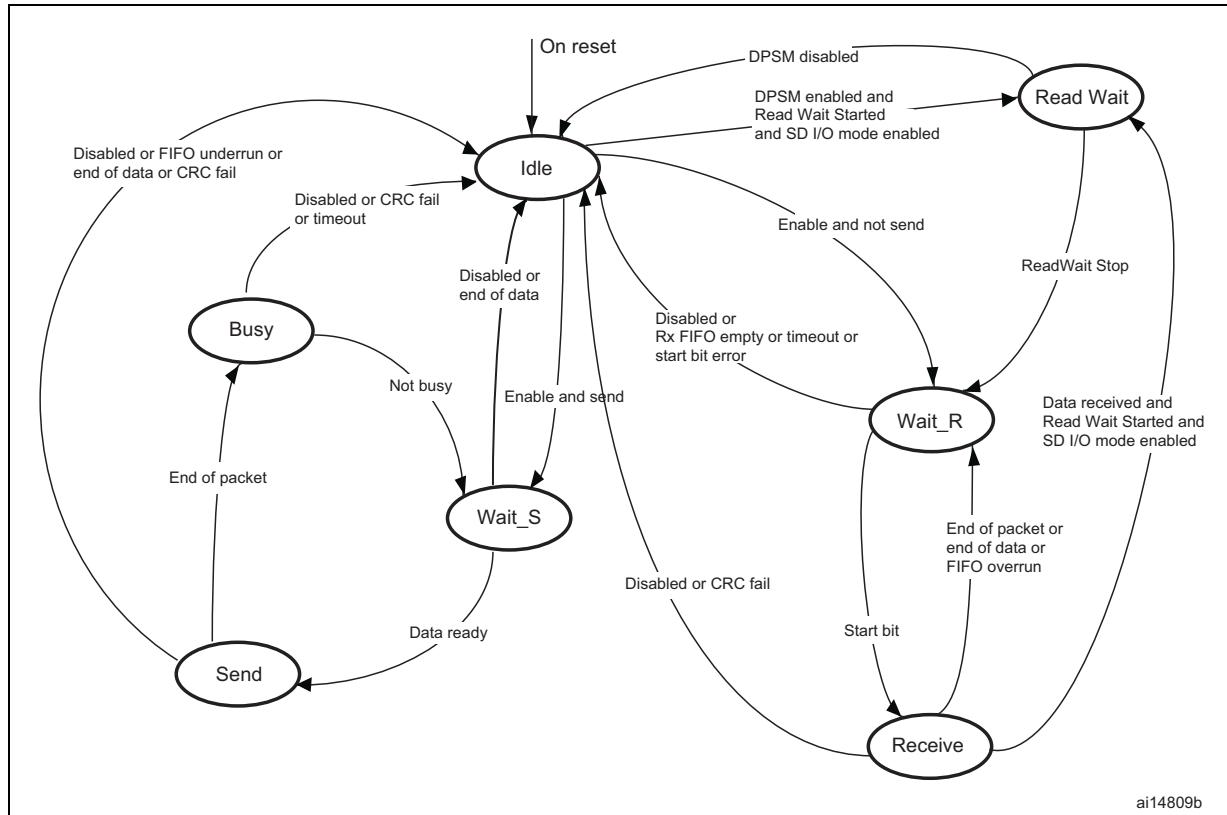
Depending on the transfer direction (send or receive), the data path state machine (DPSM) moves to the Wait\_S or Wait\_R state when it is enabled:

- Send: the DPSM moves to the Wait\_S state. If there is data in the transmit FIFO, the DPSM moves to the Send state, and the data path subunit starts sending data to a card.
- Receive: the DPSM moves to the Wait\_R state and waits for a start bit. When it receives a start bit, the DPSM moves to the Receive state, and the data path subunit starts receiving data from a card.

### Data path state machine (DPSM)

The DPSM operates at SDMMC\_CK frequency. Data on the card bus signals is synchronous to the rising edge of SDMMC\_CK. The DPSM has six states, as shown in [Figure 437: Data path state machine \(DPSM\)](#).

Figure 437. Data path state machine (DPSM)



- **Idle:** the data path is inactive, and the SDMMC\_D[7:0] outputs are in Hi-Z. When the data control register is written and the enable bit is set, the DPSM loads the data counter with a new value and, depending on the data direction bit, moves to either the Wait\_S or the Wait\_R state.
- **Wait\_R:** if the data counter equals zero, the DPSM moves to the Idle state when the receive FIFO is empty. If the data counter is not zero, the DPSM waits for a start bit on SDMMC\_D. The DPSM moves to the Receive state if it receives a start bit before a timeout, and loads the data block counter. If it reaches a timeout before it detects a start bit, it moves to the Idle state and sets the timeout status flag.
- **Receive:** serial data received from a card is packed in bytes and written to the data FIFO. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:
  - In block mode, when the data block counter reaches zero, the DPSM waits until it receives the CRC code. If the received code matches the internally generated CRC code, the DPSM moves to the Wait\_R state. If not, the CRC fail status flag is set and the DPSM moves to the Idle state.
  - In stream mode, the DPSM receives data while the data counter is not zero. When the counter is zero, the remaining data in the shift register is written to the data FIFO, and the DPSM moves to the Wait\_R state.
- If a FIFO overrun error occurs, the DPSM sets the FIFO error flag and moves to the Idle state:
- **Wait\_S:** the DPSM moves to the Idle state if the data counter is zero. If not, it waits until the data FIFO empty flag is deasserted, and moves to the Send state.

**Note:** The DPSM remains in the Wait\_S state for at least two clock periods to meet the  $N_{WR}$  timing requirements, where  $N_{WR}$  is the number of clock cycles between the reception of the card response and the start of the data transfer from the host.

- Send: the DPSM starts sending data to a card. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:
  - In block mode, when the data block counter reaches zero, the DPSM sends an internally generated CRC code and end bit, and moves to the Busy state.
  - In stream mode, the DPSM sends data to a card while the enable bit is high and the data counter is not zero. It then moves to the Idle state.
 If a FIFO underrun error occurs, the DPSM sets the FIFO error flag and moves to the Idle state.
- Busy: the DPSM waits for the CRC status flag:
  - If it does not receive a positive CRC status, it moves to the Idle state and sets the CRC fail status flag.
  - If it receives a positive CRC status, it moves to the Wait\_S state if SDMMC\_D0 is not low (the card is not busy).
 If a timeout occurs while the DPSM is in the Busy state, it sets the data timeout flag and moves to the Idle state.
- The data timer is enabled when the DPSM is in the Wait\_R or Busy state, and generates the data timeout error:
  - When transmitting data, the timeout occurs if the DPSM stays in the Busy state for longer than the programmed timeout period
  - When receiving data, the timeout occurs if the end of the data is not true, and if the DPSM stays in the Wait\_R state for longer than the programmed timeout period.
- **Data:** data can be transferred from the card to the host or vice versa. Data is transferred via the data lines. They are stored in a FIFO of 32 words, each word is 32 bits wide.

Table 230. 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 231. DPSM flags**

Flag	Description
DBCKEND	Set to high when data block send/receive CRC check is passed. In SDIO multibyte transfer mode this flag is set at the end of the transfer (a multibyte transfer is considered as a single block transfer by the host).
DATAEND	Set to high when SDMMC_DCOUNT register decrements and reaches 0. DATAEND indicates the end of a transfer on SDMMC data line.
DTIMEOUT	Set to high when data timeout period is reached. When data timer reaches zero while DPSM is in Wait_R or Busy state, timeout is set. DTIMEOUT can be set after DATAEND if DPSM remains in busy state for longer than the programmed period.
DCRCFAIL	Set to high when data block send/receive CRC check fails.

## Data FIFO

The data FIFO (first-in-first-out) subunit is a data buffer with a transmit and receive unit.

The FIFO contains a 32-bit wide, 32-word deep data buffer, and transmit and receive logic. Because the data FIFO operates in the APB2 clock domain (PCLK2), all signals from the subunits in the SDMMC clock domain (SDMMCCCLK) are resynchronized.

Depending on the TXACT and RXACT flags, the FIFO can be disabled, transmit enabled, or receive enabled. TXACT and RXACT are driven by the data path subunit and are mutually exclusive:

- The transmit FIFO refers to the transmit logic and data buffer when TXACT is asserted
- The receive FIFO refers to the receive logic and data buffer when RXACT is asserted
- Transmit FIFO:

Data can be written to the transmit FIFO through the APB2 interface when the SDMMC is enabled for transmission.

The transmit FIFO is accessible via 32 sequential addresses. The transmit FIFO contains a data output register that holds the data word pointed to by the read pointer. When the data path subunit has loaded its shift register, it increments the read pointer and drives new data out.

If the transmit FIFO is disabled, all status flags are deasserted. The data path subunit asserts TXACT when it transmits data.

**Table 232. Transmit FIFO status flags**

Flag	Description
TXFIFOF	Set to high when all 32 transmit FIFO words contain valid data.
TXFIFOE	Set to high when the transmit FIFO does not contain valid data.
TXFIFOHE	Set to high when 8 or more transmit FIFO words are empty. This flag can be used as a DMA request.
TXDAVL	Set to high when the transmit FIFO contains valid data. This flag is the inverse of the TXFIFOE flag.
TXUNDERR	Set to high when an underrun error occurs. This flag is cleared by writing to the SDMMC Clear register. <i>Note:</i> In case of TXUNDERR, and DMA is used to fill SDMMC FIFO, user software should disable DMA stream, and then write DMAEN bit in SDMMC_DCTRL with '0' (to disable DMA request generation).

- Receive FIFO

When the data path subunit receives a word of data, it drives the data on the write databus. The write pointer is incremented after the write operation completes. On the read side, the contents of the FIFO word pointed to by the current value of the read pointer is driven onto the read databus. If the receive FIFO is disabled, all status flags are deasserted, and the read and write pointers are reset. The data path subunit asserts RXACT when it receives data. [Table 233](#) lists the receive FIFO status flags. The receive FIFO is accessible via 32 sequential addresses.

**Table 233. Receive FIFO status flags**

Flag	Description
RXFIFOF	Set to high when all 32 receive FIFO words contain valid data
RXFIFOE	Set to high when the receive FIFO does not contain valid data.
RXFIFOHF	Set to high when 8 or more receive FIFO words contain valid data. This flag can be used as a DMA request.
RXDAVL	Set to high when the receive FIFO is not empty. This flag is the inverse of the RXFIFOE flag.
RXOVERR	Set to high when an overrun error occurs. This flag is cleared by writing to the SDMMC Clear register. <i>Note:</i> In case of RXOVERR, and DMA is used to read SDMMC FIFO, user software should disable DMA stream, and then write DMAEN bit in SDMMC_DCTRL with '0' (to disable DMA request generation).

### 35.3.2 SDMMC APB2 interface

The APB2 interface generates the interrupt and DMA requests, and accesses the SDMMC adapter registers and the data FIFO. It consists of a data path, register decoder, and interrupt/DMA logic.

#### SDMMC interrupts

The interrupt logic generates an interrupt request signal that is asserted when at least one of the selected status flags is high. A mask register is provided to allow selection of the conditions that will generate an interrupt. A status flag generates the interrupt request if a corresponding mask flag is set.

#### SDMMC/DMA interface

SDMMC APB interface controls all subunit to perform transfers between the host and card

#### Example of read procedure using DMA

Send CMD17 (READ\_BLOCK) as follows:

- a) Program the SDMMC data length register (SDMMC data timer register should be already programmed before the card identification process)
- b) Program DMA channel (refer to [DMA configuration for SDMMC controller](#))
- c) Program the SDMMC data control register: DTEN with '1' (SDMMC card host enabled to send data); DTDIR with '1' (from card to controller); DTMODE with '0' (block data transfer); DMAEN with '1' (DMA enabled); DBLOCKSIZE with 0x9 (512 bytes). Other fields are don't care.
- d) Program the SDMMC argument register with the address location of the card from where data is to be transferred
- e) Program the SDMMC command register: CmdIndex with 17(READ\_BLOCK); WaitResp with '1' (SDMMC card host waits for a response); CPSMEN with '1' (SDMMC card host enabled to send a command). Other fields are at their reset value.
- f) Wait for SDMMC\_STA[6] = CMDREND interrupt, (CMDREND is set if there is no error on command path).
- g) Wait for SDMMC\_STA[10] = DBCKEND, (DBCKEND is set in case of no errors until the CRC check is passed)
- h) Wait until the FIFO is empty, when FIFO is empty the SDMMC\_STA[5] = RXOVERRR value has to be checked to guarantee that read succeeded

Note:

*When FIFO overrun error occurs with last 1-4 bytes, it may happen that RXOVERRR flag is set 2 APB clock cycles after DATAEND flag is set. To guarantee success of read operation RXOVERRR must be checked after FIFO is empty.*

### Example of write procedure using DMA

Send CMD24 (WRITE\_BLOCK) as follows:

- a) Program the SDMMC data length register (SDMMC data timer register should be already programmed before the card identification process)
- b) Program DMA channel (refer to [DMA configuration for SDMMC controller](#))
- c) Program the SDMMC argument register with the address location of the card from where data is to be transferred
- d) Program the SDMMC command register: CmdIndex with 24(WRITE\_BLOCK); WaitResp with '1' (SDMMC card host waits for a response); CPSMEN with '1' (SDMMC card host enabled to send a command). Other fields are at their reset value.
- e) Wait for SDMMC\_STA[6] = CMDREND interrupt, then Program the SDMMC data control register: DTEN with '1' (SDMMC card host enabled to send data); DTDIR with '0' (from controller to card); DTMODE with '0' (block data transfer); DMAEN with '1' (DMA enabled); DBLOCKSIZE with 0x9 (512 bytes). Other fields are don't care.
- f) Wait for SDMMC\_STA[10] = DBCKEND, (DBCKEND is set in case of no errors)

### DMA configuration for SDMMC controller

- a) Enable DMA2 controller and clear any pending interrupts.
- b) Program the DMA2\_Stream3 (or DMA2\_Stream6) Channel4 source address register with the memory location base address and DMA2\_Stream3 (or DMA2\_Stream6) Channel4 destination address register with the SDMMC\_FIFO register address.
- c) Program DMA2\_Stream3 (or DMA2\_Stream6) Channel4 control register (memory increment, not peripheral increment, peripheral and source width is word size).
- d) Program DMA2\_Stream3 (or DMA2\_Stream6) Channel4 to select the peripheral as flow controller (set PFCTRL bit in DMA\_S3CR (or DMA\_S6CR) configuration register).
- e) Configure the incremental burst transfer to 4 beats (at least from peripheral side) in DMA2\_Stream3 (or DMA2\_Stream6) Channel4.
- f) Enable DMA2\_Stream3 (or DMA2\_Stream6) Channel4

Note:

*SDMMC host allows only to use the DMA in peripheral flow controller mode. DMA stream used to serve SDMMC must be configured in peripheral flow controller mode*

*SDMMC generates only DMA burst requests to DMA controller. DMA must be configured in incremental burst mode on peripheral side.*

## 35.4 Card functional description

### 35.4.1 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.

### 35.4.2 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).

### 35.4.3 Operating voltage range validation

All cards can communicate with the SDMMC card host using any operating voltage within the specification range. The supported minimum and maximum  $V_{DD}$  values are defined in the operation conditions register (OCR) on the card.

Cards that store the card identification number (CID) and card specific data (CSD) in the payload memory are able to communicate this information only under data-transfer  $V_{DD}$  conditions. When the SDMMC card host module and the card have incompatible  $V_{DD}$  ranges, the card is not able to complete the identification cycle and cannot send CSD data. For this purpose, the special commands, SEND\_OP\_COND (CMD1), SD\_APP\_OP\_COND (ACMD41 for SD Memory), and IO\_SEND\_OP\_COND (CMD5 for SD I/O), are designed to provide a mechanism to identify and reject cards that do not match the  $V_{DD}$  range desired by the SDMMC card host. The SDMMC card host sends the required  $V_{DD}$  voltage window as the operand of these commands. Cards that cannot perform data transfer in the specified range disconnect from the bus and go to the inactive state.

By using these commands without including the voltage range as the operand, the SDMMC card host can query each card and determine the common voltage range before placing out-of-range cards in the inactive state. This query is used when the SDMMC card host is able to select a common voltage range or when the user requires notification that cards are not usable.

### 35.4.4 Card identification process

The card identification process differs for MultiMediaCards and SD cards. For MultiMediaCard cards, the identification process starts at clock rate  $F_{od}$ . The SDMMC\_CMD line output drivers are open-drain and allow parallel card operation during this process. The registration process is accomplished as follows:

1. The bus is activated.
2. The SDMMC card host broadcasts SEND\_OP\_COND (CMD1) to receive operation conditions.
3. The response is the wired AND operation of the operation condition registers from all cards.
4. Incompatible cards are placed in the inactive state.
5. The SDMMC card host broadcasts ALL\_SEND\_CID (CMD2) to all active cards.
6. The active cards simultaneously send their CID numbers serially. Cards with outgoing CID bits that do not match the bits on the command line stop transmitting and must wait for the next identification cycle. One card successfully transmits a full CID to the SDMMC card host and enters the Identification state.
7. The SDMMC card host issues SET\_RELATIVE\_ADDR (CMD3) to that card. This new address is called the relative card address (RCA); it is shorter than the CID and

addresses the card. The assigned card changes to the Standby state, it does not react to further identification cycles, and its output switches from open-drain to push-pull.

8. The SDMMC card host repeats steps 5 through 7 until it receives a timeout condition.

For the SD card, the identification process starts at clock rate  $F_{od}$ , and the SDMMC\_CMD line output drivers are push-pull drivers instead of open-drain. The registration process is accomplished as follows:

1. The bus is activated.
2. The SDMMC card host broadcasts SD\_APP\_OP\_COND (ACMD41).
3. The cards respond with the contents of their operation condition registers.
4. The incompatible cards are placed in the inactive state.
5. The SDMMC card host broadcasts ALL\_SEND\_CID (CMD2) to all active cards.
6. The cards send back their unique card identification numbers (CIDs) and enter the Identification state.
7. The SDMMC card host issues SET\_RELATIVE\_ADDR (CMD3) to an active card with an address. This new address is called the relative card address (RCA); it is shorter than the CID and addresses the card. The assigned card changes to the Standby state. The SDMMC card host can reissue this command to change the RCA. The RCA of the card is the last assigned value.
8. The SDMMC card host repeats steps 5 through 7 with all active cards.

For the SD I/O card, the registration process is accomplished as follows:

1. The bus is activated.
2. The SDMMC card host sends IO\_SEND\_OP\_COND (CMD5).
3. The cards respond with the contents of their operation condition registers.
4. The incompatible cards are set to the inactive state.
5. The SDMMC card host issues SET\_RELATIVE\_ADDR (CMD3) to an active card with an address. This new address is called the relative card address (RCA); it is shorter than the CID and addresses the card. The assigned card changes to the Standby state. The SDMMC card host can reissue this command to change the RCA. The RCA of the card is the last assigned value.

### 35.4.5 Block write

During block write (CMD24 - 27) one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. A card supporting block write is always able to accept a block of data defined by WRITE\_BL\_LEN. If the CRC fails, the card indicates the failure on the SDMMC\_D line and the transferred data are discarded and not written, and all further transmitted blocks (in multiple block write mode) are ignored.

If the host uses partial blocks whose accumulated length is not block aligned and, block misalignment is not allowed (CSD parameter WRITE\_BLK\_MISALIGN is not set), the card will detect the block misalignment error before the beginning of the first misaligned block. (ADDRESS\_ERROR error bit is set in the status register). The write operation will also be aborted if the host tries to write over a write-protected area. In this case, however, the card will set the WP\_VIOLATION bit.

Programming of the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in ROM, then this unchangeable part must match the corresponding part of the receive buffer. If this match fails, then the card reports an error and does not change any register contents.

Some cards may require long and unpredictable times to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing and holds the SDMMC\_D line low if its write buffer is full and unable to accept new data from a new WRITE\_BLOCK command. The host may poll the status of the card with a SEND\_STATUS command (CMD13) at any time, and the card will respond with its status. The READY\_FOR\_DATA status bit indicates whether the card can accept new data or whether the write process is still in progress. The host may deselect the card by issuing CMD7 (to select a different card), which will place the card in the Disconnect state and release the SDMMC\_D line(s) without interrupting the write operation. When reselecting the card, it will reactivate busy indication by pulling SDMMC\_D to low if programming is still in progress and the write buffer is unavailable.

#### 35.4.6 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).

#### 35.4.7 Stream access, stream write and stream read (MultiMediaCard only)

In stream mode, data is transferred in bytes and no CRC is appended at the end of each block.

##### Stream write (MultiMediaCard only)

WRITE\_DAT\_UNTIL\_STOP (CMD20) starts the data transfer from the SDMMC card host to the card, beginning at the specified address and continuing until the SDMMC card host issues a stop command. When partial blocks are allowed (CSD parameter WRITE\_BL\_PARTIAL is set), the data stream can start and stop at any address within the card address space, otherwise it can only start and stop at block boundaries. Because the amount of data to be transferred is not determined in advance, a CRC cannot be used. When the end of the memory range is reached while sending data and no stop command is sent by the SDMMC card host, any additional transferred data are discarded.

The maximum clock frequency for a stream write operation is given by the following equation fields of the card-specific data register:

$$\text{Maximumspeed} = \text{MIN}(\text{TRANSPEED}, \frac{(8 \times 2^{\text{writeblen}})(-\text{NSAC})}{\text{TAAC} \times \text{R2WFACTOR}})$$

- Maximumspeed = maximum write frequency
- TRANSPEED = maximum data transfer rate
- writeblen = maximum write data block length
- NSAC = data read access time 2 in CLK cycles
- TAAC = data read access time 1
- R2WFACTOR = write speed factor

If the host attempts to use a higher frequency, the card may not be able to process the data and stop programming, set the OVERRUN error bit in the status register, and while ignoring all further data transfer, wait (in the receive data state) for a stop command. The write operation is also aborted if the host tries to write over a write-protected area. In this case, however, the card sets the WP\_VIOLATION bit.

### **Stream read (MultiMediaCard only)**

READ\_DAT\_UNTIL\_STOP (CMD11) controls a stream-oriented data transfer.

This command instructs the card to send its data, starting at a specified address, until the SDMMC card host sends STOP\_TRANSMISSION (CMD12). The stop command has an execution delay due to the serial command transmission and the data transfer stops after the end bit of the stop command. When the end of the memory range is reached while sending data and no stop command is sent by the SDMMC card host, any subsequent data sent are considered undefined.

The maximum clock frequency for a stream read operation is given by the following equation and uses fields of the card specific data register.

$$\text{Maximumspeed} = \text{MIN}(\text{TRANSPEED}, \frac{(8 \times 2^{\text{readblen}})(-\text{NSAC})}{\text{TAAC} \times \text{R2WFACTOR}})$$

- Maximumspeed = maximum read frequency
- TRANSPEED = maximum data transfer rate
- readblen = maximum read data block length
- writeblen = maximum write data block length
- NSAC = data read access time 2 in CLK cycles
- TAAC = data read access time 1
- R2WFACTOR = write speed factor

If the host attempts to use a higher frequency, the card is not able to sustain data transfer. If this happens, the card sets the UNDERRUN error bit in the status register, aborts the transmission and waits in the data state for a stop command.

### 35.4.8 Erase: group erase and sector erase

The erasable unit of the MultiMediaCard is the erase group. The erase group is measured in write blocks, which are the basic writable units of the card. The size of the erase group is a card-specific parameter and defined in the CSD.

The host can erase a contiguous range of Erase Groups. Starting the erase process is a three-step sequence.

First the host defines the start address of the range using the ERASE\_GROUP\_START (CMD35) command, next it defines the last address of the range using the ERASE\_GROUP\_END (CMD36) command and, finally, it starts the erase process by issuing the ERASE (CMD38) command. The address field in the erase commands is an Erase Group address in byte units. The card ignores all LSBs below the Erase Group size, effectively rounding the address down to the Erase Group boundary.

If an erase command is received out of sequence, the card sets the ERASE\_SEQ\_ERROR bit in the status register and resets the whole sequence.

If an out-of-sequence (neither of the erase commands, except SEND\_STATUS) command received, the card sets the ERASE\_RESET status bit in the status register, resets the erase sequence and executes the last command.

If the erase range includes write protected blocks, they are left intact and only nonprotected blocks are erased. The WP\_ERASE\_SKIP status bit in the status register is set.

The card indicates that an erase is in progress by holding SDMMC\_D low. The actual erase time may be quite long, and the host may issue CMD7 to deselect the card.

### 35.4.9 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).

### 35.4.10 Protection management

Three write protection methods for the cards are supported in the SDMMC card host module:

1. internal card write protection (card responsibility)
2. mechanical write protection switch (SDMMC card host module responsibility only)
3. password-protected card lock operation

#### Internal card write protection

Card data can be protected against write and erase. By setting the permanent or temporary write-protect bits in the CSD, the entire card can be permanently write-protected by the manufacturer or content provider. For cards that support write protection of groups of sectors by setting the WP\_GRP\_ENABLE bit in the CSD, portions of the data can be protected, and the write protection can be changed by the application. The write protection is in units of WP\_GRP\_SIZE sectors as specified in the CSD. The SET\_WRITE\_PROT and CLR\_WRITE\_PROT commands control the protection of the addressed group. The SEND\_WRITE\_PROT command is similar to a single block read command. The card sends

a data block containing 32 write protection bits (representing 32 write protect groups starting at the specified address) followed by 16 CRC bits. The address field in the write protect commands is a group address in byte units.

The card ignores all LSBs below the group size.

### Mechanical write protect switch

A mechanical sliding tab on the side of the card allows the user to set or clear the write protection on a card. When the sliding tab is positioned with the window open, the card is write-protected, and when the window is closed, the card contents can be changed. A matched switch on the socket side indicates to the SDMMC card host module that the card is write-protected. The SDMMC card host module is responsible for protecting the card. The position of the write protect switch is unknown to the internal circuitry of the card.

### Password protect

The password protection feature enables the SDMMC card host module to lock and unlock a card with a password. The password is stored in the 128-bit PWD register and its size is set in the 8-bit PWD\_LEN register. These registers are nonvolatile so that a power cycle does not erase them. Locked cards respond to and execute certain commands. This means that the SDMMC card host module is allowed to reset, initialize, select, and query for status, however it is not allowed to access data on the card. When the password is set (as indicated by a nonzero value of PWD\_LEN), the card is locked automatically after power-up. As with the CSD and CID register write commands, the lock/unlock commands are available in the transfer state only. In this state, the command does not include an address argument and the card must be selected before using it. The card lock/unlock commands have the structure and bus transaction types of a regular single-block write command. The transferred data block includes all of the required information for the command (the password setting mode, the PWD itself, and card lock/unlock). The command data block size is defined by the SDMMC card host module before it sends the card lock/unlock command, and has the structure shown in [Table 247](#).

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 247](#)), the 8-bit PWD\_LEN, and the number of bytes of the current password.
3. Send LOCK/UNLOCK (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (LOCK\_UNLOCK = 1), the length (PWD\_LEN), and the password (PWD) itself.
4. When the password is matched, the card is locked and the CARD\_IS\_LOCKED status bit is set in the card status register. When the password sent does not correspond (in size and/or content) to the expected password, the LOCK\_UNLOCK\_FAILED error bit is set in the card status register, and the lock fails.

It is possible to set the password and to lock the card in the same sequence. In this case, the SDMMC card host module performs all the required steps for setting the password (see [Setting the password on page 1241](#)), 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 247](#)), 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 247](#)) 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.

### 35.4.11 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 234* defines the different entries of the status. The type and clear condition fields in the table are abbreviated as follows:

Type:

- E: error bit
- S: status bit
- R: detected and set for the actual command response
- X: detected and set during command execution. The SDMMC card host must poll the card by issuing the status command to read these bits.

Clear condition:

- A: according to the card current state
- B: always related to the previous command. Reception of a valid command clears it (with a delay of one command)
- C: clear by read

**Table 234. 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 234. 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 234. Card status (continued)**

<b>Bits</b>	<b>Identifier</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>	<b>Clear condition</b>
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					

### 35.4.12 SD status register

The SD status contains status bits that are related to the SD memory card proprietary features and may be used for future application-specific usage. The size of the SD Status is one data block of 512 bits. The contents of this register are transmitted to the SDMMC card host if ACMD13 is sent (CMD55 followed with CMD13). ACMD13 can be sent to a card in transfer state only (card is selected).

[Table 235](#) defines the different entries of the SD status register. The type and clear condition fields in the table are abbreviated as follows:

Type:

- E: error bit
- S: status bit
- R: detected and set for the actual command response
- X: detected and set during command execution. The SDMMC card Host must poll the card by issuing the status command to read these bits

Clear condition:

- A: according to the card current state
- B: always related to the previous command. Reception of a valid command clears it (with a delay of one command)
- C: clear by read

**Table 235. 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_PROTECT_ED_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 235. 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 236. 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 237. 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 238. 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 239](#). The card can be set to any AU size between RU size and maximum AU size.

**Table 239. 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 240. 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 241. 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 242. Erase offset field**

ERASE_OFFSET	Value definition
0h	0 [sec]
1h	1 [sec]

**Table 242. Erase offset field (continued)**

ERASE_OFFSET	Value definition
2h	2 [sec]
3h	3 [sec]

### 35.4.13 SD I/O mode

#### SD I/O interrupts

To allow the SD I/O card to interrupt the MultiMediaCard/SD module, an interrupt function is available on a pin on the SD interface. Pin 8, used as SDMMC\_D1 when operating in the 4-bit SD mode, signals the cards interrupt to the MultiMediaCard/SD module. The use of the interrupt is optional for each card or function within a card. The SD I/O interrupt is level-sensitive, which means that the interrupt line must be held active (low) until it is either recognized and acted upon by the MultiMediaCard/SD module or deasserted due to the end of the interrupt period. After the MultiMediaCard/SD module has serviced the interrupt, the interrupt status bit is cleared via an I/O write to the appropriate bit in the SD I/O card's internal registers. The interrupt output of all SD I/O cards is active low and the application must provide pull-up resistors externally on all data lines (SDMMC\_D[3:0]). The MultiMediaCard/SD module samples the level of pin 8 (SDMMC\_D/IRQ) into the interrupt detector only during the interrupt period. At all other times, the MultiMediaCard/SD module ignores this value.

The interrupt period is applicable for both memory and I/O operations. The definition of the interrupt period for operations with single blocks is different from the definition for multiple-block data transfers.

#### SD I/O suspend and resume

Within a multifunction SD I/O or a card with both I/O and memory functions, there are multiple devices (I/O and memory) that share access to the MMC/SD bus. To share access to the MMC/SD module among multiple devices, SD I/O and combo cards optionally implement the concept of suspend/resume. When a card supports suspend/resume, the MMC/SD module can temporarily halt a data transfer operation to one function or memory (suspend) to free the bus for a higher-priority transfer to a different function or memory. After this higher-priority transfer is complete, the original transfer is resumed (restarted) where it left off. Support of suspend/resume is optional on a per-card basis. To perform the suspend/resume operation on the MMC/SD bus, the MMC/SD module performs the following steps:

1. Determines the function currently using the SDMMC\_D [3:0] line(s)
2. Requests the lower-priority or slower transaction to suspend
3. Waits for the transaction suspension to complete
4. Begins the higher-priority transaction
5. Waits for the completion of the higher priority transaction
6. Restores the suspended transaction

#### SD I/O ReadWait

The optional ReadWait (RW) operation is defined only for the SD 1-bit and 4-bit modes. The ReadWait operation allows the MMC/SD module to signal a card that it is reading multiple

registers (IO\_RW\_EXTENDED, CMD53) to temporarily stall the data transfer while allowing the MMC/SD module to send commands to any function within the SD I/O device. To determine when a card supports the ReadWait protocol, the MMC/SD module must test capability bits in the internal card registers. The timing for ReadWait is based on the interrupt period.

### 35.4.14 Commands and responses

#### Application-specific and general commands

The SDMMC card host module system is designed to provide a standard interface for a variety of applications types. In this environment, there is a need for specific customer/application features. To implement these features, two types of generic commands are defined in the standard: application-specific commands (ACMD) and general commands (GEN\_CMD).

When the card receives the APP\_CMD (CMD55) command, the card expects the next command to be an application-specific command. ACMDs have the same structure as regular MultiMediaCard commands and can have the same CMD number. The card recognizes it as ACMD because it appears after APP\_CMD (CMD55). When the command immediately following the APP\_CMD (CMD55) is not a defined application-specific command, the standard command is used. For example, when the card has a definition for SD\_STATUS (ACMD13), and receives CMD13 immediately following APP\_CMD (CMD55), this is interpreted as SD\_STATUS (ACMD13). However, when the card receives CMD7 immediately following APP\_CMD (CMD55) and the card does not have a definition for ACMD7, this is interpreted as the standard (SELECT/DESELECT\_CARD) CMD7.

To use one of the manufacturer-specific ACMDs the SD card Host must perform the following steps:

1. Send APP\_CMD (CMD55)  
The card responds to the MultiMediaCard/SD module, indicating that the APP\_CMD bit is set and an ACMD is now expected.
2. Send the required ACMD  
The card responds to the MultiMediaCard/SD module, indicating that the APP\_CMD bit is set and that the accepted command is interpreted as an ACMD. When a nonACMD is sent, it is handled by the card as a normal MultiMediaCard command and the APP\_CMD bit in the card status register stays clear.

When an invalid command is sent (neither ACMD nor CMD) it is handled as a standard MultiMediaCard illegal command error.

The bus transaction for a GEN\_CMD is the same as the single-block read or write commands (WRITE\_BLOCK, CMD24 or READ\_SINGLE\_BLOCK,CMD17). In this case, the argument denotes the direction of the data transfer rather than the address, and the data block has vendor-specific format and meaning.

The card must be selected (in transfer state) before sending GEN\_CMD (CMD56). The data block size is defined by SET\_BLOCKLEN (CMD16). The response to GEN\_CMD (CMD56) is in R1b format.

## Command types

Both application-specific and general commands are divided into the four following types:

- **broadcast command (BC)**: sent to all cards; no responses returned.
- **broadcast command with response (BCR)**: sent to all cards; responses received from all cards simultaneously.
- **addressed (point-to-point) command (AC)**: sent to the card that is selected; does not include a data transfer on the SDMMC\_D line(s).
- **addressed (point-to-point) data transfer command (ADTC)**: sent to the card that is selected; includes a data transfer on the SDMMC\_D line(s).

## Command formats

See [Table 226 on page 1227](#) for command formats.

## Commands for the MultiMediaCard/SD module

**Table 243. 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	<b>SET_BLOCK_COUNT</b>	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	<b>WRITE_BLOCK</b>	Writes a block of the size selected by the SET_BLOCKLEN command.
CMD25	adtc	[31:0] data address	R1	<b>WRITE_MULTIPLE_BLOCK</b>	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	<b>PROGRAM_CID</b>	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	<b>PROGRAM_CSD</b>	Programming of the programmable bits of the CSD.

**Table 244. Block-oriented write protection commands**

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD28	ac	[31:0] data address	R1b	<b>SET_WRITE_PROT</b>	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	<b>CLR_WRITE_PROT</b>	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	<b>SEND_WRITE_PROT</b>	If the card provides write protection features, this command asks the card to send the status of the write protection bits.
CMD31	Reserved				

**Table 245. 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	<b>ERASE_GROUP_START</b>	Sets the address of the first erase group within a range to be selected for erase.
CMD36	ac	[31:0] data address	R1	<b>ERASE_GROUP_END</b>	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	<b>ERASE</b>	Erases all previously selected write blocks.

**Table 246. 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	<b>FAST_IO</b>	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 246. 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 247. 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 248. 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.				

## 35.5 Response formats

All responses are sent via the SDMMC command line SDMMC\_CMD. The response transmission always starts with the left bit of the bit string corresponding to the response code word. The code length depends on the response type.

A response always starts with a start bit (always 0), followed by the bit indicating the direction of transmission (card = 0). A value denoted by x in the tables below indicates a variable entry. All responses, except for the R3 response type, are protected by a CRC. Every command code word is terminated by the end bit (always 1).

There are five types of responses. Their formats are defined as follows:

### 35.5.1 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 249. 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

### 35.5.2 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.

### 35.5.3 R2 (CID, CSD register)

Code length = 136 bits. The contents of the CID register are sent as a response to the CMD2 and CMD10 commands. The contents of the CSD register are sent as a response to CMD9. Only the bits [127...1] of the CID and CSD are transferred, the reserved bit [0] of these registers is replaced by the end bit of the response. The card indicates that an erase is in progress by holding SDMMC\_D0 low. The actual erase time may be quite long, and the host may issue CMD7 to deselect the card.

**Table 250. 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

### 35.5.4 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 251. 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

### 35.5.5 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 252. 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

### 35.5.6 R4b

For SD I/O only: an SDIO card receiving the CMD5 will respond with a unique SDIO response R4. The format is:

**Table 253. R4b response**

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	X	Reserved

**Table 253. 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.

### 35.5.7 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 254. 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

### 35.5.8 R6

Only for SD I/O. The normal response to CMD3 by a memory device. It is shown in [Table 255](#).

**Table 255. R6 response**

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	'101000'	CMD40
[39:8] Argument field	[31:16]	16	RCA [31:16] of winning card or of the host
	[15:0]	16	Not defined. May be used for IRQ data
[7:1]	7	X	CRC7
0	1	1	End bit

The card [23:8] status bits are changed when CMD3 is sent to an I/O-only card. In this case, the 16 bits of response are the SD I/O-only values:

- Bit [15] COM\_CRC\_ERROR
- Bit [14] ILLEGAL\_COMMAND
- Bit [13] ERROR
- Bits [12:0] Reserved

## 35.6 SDIO I/O card-specific operations

The following features are SD I/O-specific operations:

- SDIO read wait operation by SDMMC\_D2 signalling
- SDIO read wait operation by stopping the clock
- SDIO suspend/resume operation (write and read suspend)
- SDIO interrupts

The SDMMC supports these operations only if the SDMMC\_DCTRL[11] bit is set, except for read suspend that does not need specific hardware implementation.

### 35.6.1 SDIO I/O read wait operation by SDMMC\_D2 signalling

It is possible to start the readwait interval before the first block is received: when the data path is enabled (SDMMC\_DCTRL[0] bit set), the SDIO-specific operation is enabled (SDMMC\_DCTRL[11] bit set), read wait starts (SDMMC\_DCTRL[10] =0 and SDMMC\_DCTRL[8] =1) and data direction is from card to SDMMC (SDMMC\_DCTRL[1] = 1), the DPSM directly moves from Idle to Readwait. In Readwait the DPSM drives SDMMC\_D2 to 0 after 2 SDMMC\_CK clock cycles. In this state, when you set the RWSTOP bit (SDMMC\_DCTRL[9]), the DPSM remains in Wait for two more SDMMC\_CK clock cycles to drive SDMMC\_D2 to 1 for one clock cycle (in accordance with SDIO specification). The DPSM then starts waiting again until it receives data from the card. The DPSM will not start a readwait interval while receiving a block even if read wait start is set: the readwait interval will start after the CRC is received. The RWSTOP bit has to be cleared to start a new read wait operation. During the readwait interval, the SDMMC can detect SDIO interrupts on SDMMC\_D1.

### 35.6.2 SDIO read wait operation by stopping SDMMC\_CK

If the SDIO card does not support the previous read wait method, the SDMMC can perform a read wait by stopping SDMMC\_CK (SDMMC\_DCTRL is set just like in the method presented in [Section 35.6.1](#), but SDMMC\_DCTRL[10] =1): DPSM stops the clock two SDMMC\_CK cycles after the end bit of the current received block and starts the clock again after the read wait start bit is set.

As SDMMC\_CK is stopped, any command can be issued to the card. During a read/wait interval, the SDMMC can detect SDIO interrupts on SDMMC\_D1.

### 35.6.3 SDIO suspend/resume operation

While sending data to the card, the SDMMC can suspend the write operation. the SDMMC\_CMD[11] bit is set and indicates to the CPSM that the current command is a suspend command. The CPSM analyzes the response and when the ACK is received from the card (suspend accepted), it acknowledges the DPSM that goes Idle after receiving the CRC token of the current block.

The hardware does not save the number of the remaining block to be sent to complete the suspended operation (resume).

The write operation can be suspended by software, just by disabling the DPSM (SDMMC\_DCTRL[0] =0) when the ACK of the suspend command is received from the card. The DPSM enters then the Idle state.

To suspend a read: the DPSM waits in the Wait\_r state as the function to be suspended sends a complete packet just before stopping the data transaction. The application continues reading RxFIFO until the FIFO is empty, and the DPSM goes Idle automatically.

### 35.6.4 SDIO interrupts

SDIO interrupts are detected on the SDMMC\_D1 line once the SDMMC\_DCTRL[11] bit is set.

When SDIO interrupt is detected, SDMMC\_STA[22] (SDIOIT) bit is set. This static bit can be cleared with clear bit SDMMC\_ICR[22] (SDIOITC). An interrupt can be generated when SDIOIT status bit is set. Separated interrupt enable SDMMC\_MASK[22] bit (SDIOITE) is available to enable and disable interrupt request.

When SD card interrupt occurs (SDMMC\_STA[22] bit set), host software follows below steps to handle it.

1. Disable SDIOIT interrupt signaling by clearing SDIOITE bit (SDMMC\_MASK[22] = '0'),
2. Serve card interrupt request, and clear the source of interrupt on the SD card,
3. Clear SDIOIT bit by writing '1' to SDIOITC bit (SDMMC\_ICR[22] = '1'),
4. Enable SDIOIT interrupt signaling by writing '1' to SDIOITE bit (SDMMC\_MASK[22] = '1').

*Steps 2 to 4 can be executed out of the SDIO interrupt service routine.*

## 35.7 HW flow control

The HW flow control functionality is used to avoid FIFO underrun (TX mode) and overrun (RX mode) errors.

The behavior is to stop SDMMC\_CK and freeze SDMMC state machines. The data transfer is stalled while the FIFO is unable to transmit or receive data. Only state machines clocked by SDMMCCLK are frozen, the APB2 interface is still alive. The FIFO can thus be filled or emptied even if flow control is activated.

To enable HW flow control, the SDMMC\_CLKCR[14] register bit must be set to 1. After reset Flow Control is disabled.

## 35.8 SDMMC registers

The device communicates to the system via 32-bit-wide control registers accessible via APB2.

### 35.8.1 SDMMC power control register (SDMMC\_POWER)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PWRCTRL														
															rw   rw

Bits 31:2 Reserved, must be kept at reset value.

[1:0] **PWRCTRL:** Power supply control bits.

These bits are used to define the current functional state of the card clock:

00: Power-off: the clock to card is stopped.

01: Reserved

10: Reserved power-up

11: Power-on: the card is clocked.

**Note:** At least seven PCLK2 clock periods are needed between two write accesses to this register.

**Note:** After a data write, data cannot be written to this register for three SDMMCCLK clock periods plus two PCLK2 clock periods.

### 35.8.2 SDMMC clock control register (SDMMC\_CLKCR)

Address offset: 0x04

Reset value: 0x0000 0000

The SDMMC\_CLKCR register controls the SDMMC\_CK output clock.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	HWFC_EN	NEGEDGE	WIDBUS	BYPASS	PWRSAV	CLKEN	CLKDIV								
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, must be kept at reset value.

Bit 14 **HWFC\_EN:** HW Flow Control enable

0b: HW Flow Control is disabled

1b: HW Flow Control is enabled

When HW Flow Control is enabled, the meaning of the TXFIFOE and RXFIFOF interrupt signals, see SDMMC Status register definition in [Section 35.8.11](#).

Bit 13 **NEGEDGE:** SDMMC\_CK dephasing selection bit

0b: Command and Data changed on the SDMMCCLK falling edge succeeding the rising edge of SDMMC\_CK. (SDMMC\_CK rising edge occurs on SDMMCCLK rising edge).

1b: Command and Data changed on the SDMMC\_CK falling edge.

When BYPASS is active, the data and the command change on SDMMCCLK falling edge whatever NEGEDGE value.

Bits 12:11 **WIDBUS:** Wide bus mode enable bit

00: Default bus mode: SDMMC\_D0 used

01: 4-wide bus mode: SDMMC\_D[3:0] used

10: 8-wide bus mode: SDMMC\_D[7:0] used

Bit 10 **BYPASS:** Clock divider bypass enable bit

0: Disable bypass: SDMMCCLK is divided according to the CLKDIV value before driving the SDMMC\_CK output signal.

1: Enable bypass: SDMMCCLK directly drives the SDMMC\_CK output signal.

Bit 9 **PWRSAV:** Power saving configuration bit

For power saving, the SDMMC\_CK clock output can be disabled when the bus is idle by setting PWRSAV:

0: SDMMC\_CK clock is always enabled

1: SDMMC\_CK is only enabled when the bus is active

Bit 8 **CLKEN:** Clock enable bit

0: SDMMC\_CK is disabled

1: SDMMC\_CK is enabled

Bits 7:0 **CLKDIV:** Clock divide factor

This field defines the divide factor between the input clock (SDMMCCLK) and the output clock (SDMMC\_CK): SDMMC\_CK frequency = SDMMCCLK / [CLKDIV + 2].

Note: Only even values of CLKDIV are supported (in order to have a duty cycle of 50%).

- Note:
- 1 While the SD/SDIO card or MultiMediaCard is in identification mode, the SDMMC\_CK frequency must be less than 400 kHz.
  - 2 The clock frequency can be changed to the maximum card bus frequency when relative card addresses are assigned to all cards.
  - 3 After a data write, data cannot be written to this register for three SDMMCCLK clock periods plus two PCLK2 clock periods. SDMMC\_CK can also be stopped during the read wait

*interval for SD I/O cards: in this case the SDMMC\_CLKCR register does not control SDMMC\_CK.*

### 35.8.3 SDMMC argument register (SDMMC\_ARG)

Address offset: 0x08

Reset value: 0x0000 0000

The SDMMC\_ARG register contains a 32-bit command argument, which is sent to a card as part of a command message.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CMDARG[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMDARG[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **CMDARG:** Command argument

Command argument sent to a card as part of a command message. If a command contains an argument, it must be loaded into this register before writing a command to the command register.

### 35.8.4 SDMMC command register (SDMMC\_CMD)

Address offset: 0x0C

Reset value: 0x0000 0000

The SDMMC\_CMD register contains the command index and command type bits. The command index is sent to a card as part of a command message. The command type bits control the command path state machine (CPSM).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	SDIO Suspend	CPSM EN	WAIT PEND	WAIT INT	WAITRESP		CMDINDEX					
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bit 11 **SDIOSuspend:** SD I/O suspend command

If this bit is set, the command to be sent is a suspend command (to be used only with SDIO card).

Bit 10 **CPSMEN:** Command path state machine (CPSM) Enable bit

If this bit is set, the CPSM is enabled.

Bit 9 **WAITPEND:** CPSM Waits for ends of data transfer (CmdPend internal signal).

If this bit is set, the CPSM waits for the end of data transfer before it starts sending a command. This feature is available only with Stream data transfer mode SDMMC\_DCTRL[2] = 1.

Bit 8 **WAITINT**: CPSM waits for interrupt request

If this bit is set, the CPSM disables command timeout and waits for an interrupt request.

Bits 7:6 **WAITRESP**: Wait for response bits

They are used to configure whether the CPSM is to wait for a response, and if yes, which kind of response.

00: No response, expect CMDSENT flag

01: Short response, expect CMDREND or CCRCFAIL flag

10: No response, expect CMDSENT flag

11: Long response, expect CMDREND or CCRCFAIL flag

Bits 5:0 **CMDINDEX**: Command index

The command index is sent to the card as part of a command message.

- Note:**
- 1 *After a data write, data cannot be written to this register for three SDMMCCLK clock periods plus two PCLK2 clock periods.*
  - 2 *MultiMediaCards can send two kinds of response: short responses, 48 bits long, or long responses, 136 bits long. SD card and SD I/O card can send only short responses, the argument can vary according to the type of response: the software will distinguish the type of response according to the sent command.*

### 35.8.5 SDMMC command response register (SDMMC\_RESPCMD)

Address offset: 0x10

Reset value: 0x0000 0000

The SDMMC\_RESPCMD register contains the command index field of the last command response received. If the command response transmission does not contain the command index field (long or OCR response), the RESPCMD field is unknown, although it must contain 111111b (the value of the reserved field from the response).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
										r	r	r	r	r	r
RESPCMD															

Bits 31:6 Reserved, must be kept at reset value.

Bits 5:0 **RESPCMD**: Response command index

Read-only bit field. Contains the command index of the last command response received.

### 35.8.6 SDMMC response 1..4 register (SDMMC\_RESPx)

Address offset: (0x10 + (4 × x)); x = 1..4

Reset value: 0x0000 0000

The SDMMC\_RESP1/2/3/4 registers contain the status of a card, which is part of the received response.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CARDSTATUSx[31:16]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CARDSTATUSx[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **CARDSTATUSx**: see [Table 256](#).

The Card Status size is 32 or 127 bits, depending on the response type.

**Table 256. Response type and SDMMC\_RESPx registers**

Register	Short response	Long response
SDMMC_RESP1	Card Status[31:0]	Card Status [127:96]
SDMMC_RESP2	Unused	Card Status [95:64]
SDMMC_RESP3	Unused	Card Status [63:32]
SDMMC_RESP4	Unused	Card Status [31:1]0b

The most significant bit of the card status is received first. The SDMMC\_RESP4 register LSB is always 0b.

### 35.8.7 SDMMC data timer register (SDMMC\_DTIMER)

Address offset: 0x24

Reset value: 0x0000 0000

The SDMMC\_DTIMER register contains the data timeout period, in card bus clock periods.

A counter loads the value from the SDMMC\_DTIMER register, and starts decrementing when the data path state machine (DPSM) enters the Wait\_R or Busy state. If the timer reaches 0 while the DPSM is in either of these states, the timeout status flag is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATETIME[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATETIME[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DATETIME**: Data timeout period

Data timeout period expressed in card bus clock periods.

**Note:** A data transfer must be written to the data timer register and the data length register before being written to the data control register.

### 35.8.8 SDMMC data length register (SDMMC\_DLEN)

Address offset: 0x28

Reset value: 0x0000 0000

The SDMMC\_DLEN register contains the number of data bytes to be transferred. The value is loaded into the data counter when data transfer starts.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16					
Res.	Res.	Res.	Res.	Res.	Res.	Res.	DATALENGTH[24:16]													
							rw	rw	rw	rw	rw	rw	rw	rw	rw					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
DATALENGTH[15:0]																				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw					

Bits 31:25 Reserved, must be kept at reset value.

Bits 24:0 **DATALENGTH:** Data length value

Number of data bytes to be transferred.

**Note:** For a block data transfer, the value in the data length register must be a multiple of the block size (see SDMMC\_DCTRL). Before being written to the data control register a timeout must be written to the data timer register and the data length register.

In case of IO\_RW\_EXTENDED (CMD53):

- If the Stream or SDIO multibyte data transfer is selected the value in the data length register must be between 1 and 512.

- If the Block data transfer is selected the value in the data length register must be between 1\*Data block size and 512\*Data block size.

### 35.8.9 SDMMC data control register (SDMMC\_DCTRL)

Address offset: 0x2C

Reset value: 0x0000 0000

The SDMMC\_DCTRL register control the data path state machine (DPSM).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	SDIO EN	RW MOD	RW STOP	RW START	DBLOCKSIZE				DMA EN	DT MODE	DTDIF	DTEN
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bit 11 **SDIOEN:** SD I/O enable functions

If this bit is set, the DPSM performs an SD I/O-card-specific operation.

Bit 10 **RWMOD:** Read wait mode

- 0: Read Wait control stopping SDMMC\_D2
- 1: Read Wait control using SDMMC\_CK

Bit 9 **RWSTOP:** Read wait stop

- 0: Read wait in progress if RWSTART bit is set
- 1: Enable for read wait stop if RWSTART bit is set

Bit 8 **RWSTART:** Read wait start

If this bit is set, read wait operation starts.

Bits 7:4 **DBLOCKSIZE:** Data block size

Define the data block length when the block data transfer mode is selected:

- 0000: (0 decimal) lock length =  $2^0$  = 1 byte
- 0001: (1 decimal) lock length =  $2^1$  = 2 bytes
- 0010: (2 decimal) lock length =  $2^2$  = 4 bytes
- 0011: (3 decimal) lock length =  $2^3$  = 8 bytes
- 0100: (4 decimal) lock length =  $2^4$  = 16 bytes
- 0101: (5 decimal) lock length =  $2^5$  = 32 bytes
- 0110: (6 decimal) lock length =  $2^6$  = 64 bytes
- 0111: (7 decimal) lock length =  $2^7$  = 128 bytes
- 1000: (8 decimal) lock length =  $2^8$  = 256 bytes
- 1001: (9 decimal) lock length =  $2^9$  = 512 bytes
- 1010: (10 decimal) lock length =  $2^{10}$  = 1024 bytes
- 1011: (11 decimal) lock length =  $2^{11}$  = 2048 bytes
- 1100: (12 decimal) lock length =  $2^{12}$  = 4096 bytes
- 1101: (13 decimal) lock length =  $2^{13}$  = 8192 bytes
- 1110: (14 decimal) lock length =  $2^{14}$  = 16384 bytes
- 1111: (15 decimal) reserved

Bit 3 **DMAEN:** DMA enable bit

- 0: DMA disabled.
- 1: DMA enabled.

Bit 2 **DTMODE:** Data transfer mode selection 1: Stream or SDIO multibyte data transfer.

- 0: Block data transfer
- 1: Stream or SDIO multibyte data transfer

Bit 1 **DTDIR:** Data transfer direction selection

- 0: From controller to card.
- 1: From card to controller.

[0] **DTEN:** Data transfer enabled bit

Data transfer starts if 1b is written to the DTEN bit. Depending on the direction bit, DTDIR, the DPSM moves to the Wait\_S, Wait\_R state or Readwait if RW Start is set immediately at the beginning of the transfer. It is not necessary to clear the enable bit after the end of a data transfer but the SDMMC\_DCTRL must be updated to enable a new data transfer

**Note:** After a data write, data cannot be written to this register for three SDMMCCCLK (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.

### 35.8.10 SDMMC data counter register (SDMMC\_DCOUNT)

Address offset: 0x30

Reset value: 0x0000 0000

The SDMMC\_DCOUNT register loads the value from the data length register (see SDMMC\_DLEN) when the DPSM moves from the Idle state to the Wait\_R or Wait\_S state. As data is transferred, the counter decrements the value until it reaches 0. The DPSM then moves to the Idle state and the data status end flag, DATAEND, is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16						
Res.	Res.	Res.	Res.	Res.	Res.	Res.	DATACOUNT[24:16]														
							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.

### 35.8.11 SDMMC status register (SDMMC\_STA)

Address offset: 0x34

Reset value: 0x0000 0000

The SDMMC\_STA register is a read-only register. It contains two types of flag:

- Static flags (bits [23:22,10:0]): these bits remain asserted until they are cleared by writing to the SDMMC Interrupt Clear register (see SDMMC\_ICR)
- Dynamic flags (bits [21:11]): these bits change state depending on the state of the underlying logic (for example, FIFO full and empty flags are asserted and deasserted as data while written to the FIFO)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIOIT	RXD AVL	TXD AVL	RX FIFOE	TX FIFOE	RX FIFOF	TX FIFOF
									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 SDMMC FIFO (DMAEN bit is set in SDMMC\_DCTRL register), user software should disable DMA stream, and then write with '0' (to disable DMA request generation).*
- Bit 4 **TXUNDERR:** Transmit FIFO underrun error  
*Note: If DMA is used to fill SDMMC FIFO (DMAEN bit is set in SDMMC\_DCTRL register), user software should disable DMA stream, and then write DMAEN with '0' (to disable DMA request generation).*
- Bit 3 **DTIMEOUT:** Data timeout
- Bit 2 **CTIMEOUT:** Command response timeout  
The Command TimeOut period has a fixed value of 64 SDMMC\_CK clock periods.
- Bit 1 **DCRCFAIL:** Data block sent/received (CRC check failed)
- Bit 0 **CCRCFAIL:** Command response received (CRC check failed)

### 35.8.12 SDMMC interrupt clear register (SDMMC\_ICR)

Address offset: 0x38

Reset value: 0x0000 0000

The SDMMC\_ICR register is a write-only register. Writing a bit with 1b clears the corresponding bit in the SDMMC\_STA Status register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIO ITC	Res.	Res.	Res.	Res.	Res.	Res.
									rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DBCK ENDC	Res.	DATA ENDC	CMD SENTC	CMD REND C	RX OVERR C	TX UNDERR C	DTIME OUTC	CTIME OUTC	DCRC FAILC	CCRC FAILC
					rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bit 22 **SDIOITC:** SDIOIT flag clear bit

Set by software to clear the SDIOIT flag.

- 0: SDIOIT not cleared
- 1: SDIOIT cleared

Bits 21:11 Reserved, must be kept at reset value.

Bit 10 **DBCKENDC:** DBCKEND flag clear bit

Set by software to clear the DBCKEND flag.

- 0: DBCKEND not cleared
- 1: DBCKEND cleared

Bit 9 Reserved, must be kept at reset value.

Bit 8 **DATAENDC:** DATAEND flag clear bit

Set by software to clear the DATAEND flag.

- 0: DATAEND not cleared
- 1: DATAEND cleared

Bit 7 **CMDSENTC:** CMDSENT flag clear bit

Set by software to clear the CMDSENT flag.

- 0: CMDSENT not cleared
- 1: CMDSENT cleared

Bit 6 **CMDRENDC:** CMDREND flag clear bit

Set by software to clear the CMDREND flag.

- 0: CMDREND not cleared
- 1: CMDREND cleared

Bit 5 **RXOVERRC:** RXOVERR flag clear bit

Set by software to clear the RXOVERR flag.

- 0: RXOVERR not cleared
- 1: RXOVERR cleared

Bit 4 **TXUNDERRC:** TXUNDERR flag clear bit

Set by software to clear TXUNDERR flag.

- 0: TXUNDERR not cleared
- 1: TXUNDERR cleared

Bit 3 **DTIMEOUTC:** DTIMEOUT flag clear bit

Set by software to clear the DTIMEOUT flag.

- 0: DTIMEOUT not cleared
- 1: DTIMEOUT cleared

Bit 2 **CTIMEOUTC:** CTIMEOUT flag clear bit

Set by software to clear the CTIMEOUT flag.

0: CTIMEOUT not cleared

1: CTIMEOUT cleared

Bit 1 **DCRCFAILC:** DCRCFAIL flag clear bit

Set by software to clear the DCRCFAIL flag.

0: DCRCFAIL not cleared

1: DCRCFAIL cleared

Bit 0 **CCRCFAILC:** CCRCFAIL flag clear bit

Set by software to clear the CCRCFAIL flag.

0: CCRCFAIL not cleared

1: CCRCFAIL cleared

**35.8.13 SDMMC mask register (SDMMC\_MASK)**

Address offset: 0x3C

Reset value: 0x0000 0000

The interrupt mask register determines which status flags generate an interrupt request by setting the corresponding bit to 1b.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIO ITIE	RXD AVLIE	TXD AVLIE	RX FIFO EIE	TX FIFO EIE	RX FIFO EIE	TX FIFO EIE
									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

### 35.8.14 SDMMC FIFO counter register (SDMMC\_FIFOCNT)

Address offset: 0x48

Reset value: 0x0000 0000

The SDMMC\_FIFOCNT register contains the remaining number of words to be written to or read from the FIFO. The FIFO counter loads the value from the data length register (see SDMMC\_DLEN) when the data transfer enable bit, DTEN, is set in the data control register (SDMMC\_DCTRL register) and the DPSM is at the Idle state. If the data length is not word-aligned (multiple of 4), the remaining 1 to 3 bytes are regarded as a word.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16						
Res	Res	Res	Res	Res	Res	Res	Res	FIFOCOUNT[23:16]													
								r	r	r	r	r	r	r	r						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
FIFOCOUNT[15:0]																					
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r						

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:0 **FIFOCOUNT**: Remaining number of words to be written to or read from the FIFO.

### 35.8.15 SDMMC data FIFO register (SDMMC\_FIFO)

Address offset: 0x80

Reset value: 0x0000 0000

The receive and transmit FIFOs can be read or written as 32-bit wide registers. The FIFOs contain 32 entries on 32 sequential addresses. This allows the CPU to use its load and store multiple operands to read from/write to the FIFO.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FIFOData[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIFOData[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

bits 31:0 **FIFOData**: Receive and transmit FIFO data

The FIFO data occupies 32 entries of 32-bit words, from address:  
SDMMC base + 0x080 to SDMMC base + 0xFC.

### 35.8.16 SDMMC register map

The following table summarizes the SDMMC registers.

**Table 257. SDMMC register map**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<b>SDMMC_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.	Res.	
		Reset value																															
0x04	<b>SDMMC_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.	Res.	
		Reset value																															
0x08	<b>SDMMC_ARG</b>	CMDARG																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0C	<b>SDMMC_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.	Res.	
		Reset value																															
0x10	<b>SDMMC_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.	Res.	
		Reset value																															
0x14	<b>SDMMC_RESP1</b>	CARDSTATUS1																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x18	<b>SDMMC_RESP2</b>	CARDSTATUS2																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x1C	<b>SDMMC_RESP3</b>	CARDSTATUS3																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x20	<b>SDMMC_RESP4</b>	CARDSTATUS4																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x24	<b>SDMMC_DTIME</b>	DATATIME																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x28	<b>SDMMC_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.	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
0x2C	<b>SDMMC_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.	Res.	
		Reset value																															

Table 257. SDMMC register map (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x30	<b>SDMMC_DCOUNT</b>	Res.																																
	Reset value																																	
0x34	<b>SDMMC_STA</b>	Res.																																
	Reset value																																	
0x38	<b>SDMMC_ICR</b>	Res.																																
	Reset value																																	
0x3C	<b>SDMMC_MASK</b>	Res.																																
	Reset value																																	
0x48	<b>SDMMC_FIFOCNT</b>	Res.																																
	Reset value																																	
0x80	<b>SDMMC_FIFO</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	0	

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 36 Controller area network (bxCAN)

### 36.1 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.

### 36.2 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 for dual CAN
- Identifier list feature
- Configurable FIFO overrun
- Time stamp on SOF reception

#### Time-triggered communication option

- Disable automatic retransmission mode
- 16-bit free running timer
- Time stamp sent in last two data bytes

#### Management

- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

#### Dual CAN peripheral configuration

- CAN1: Master bxCAN for managing the communication between a Slave bxCAN and the 512-byte SRAM
- CAN2: Slave bxCAN, with no direct access to the SRAM.
- The two bxCAN cells share the 512-byte SRAM (see [Figure 439](#))

### 36.3 bxCAN general description

In today CAN applications, the number of nodes in a network is increasing and 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.

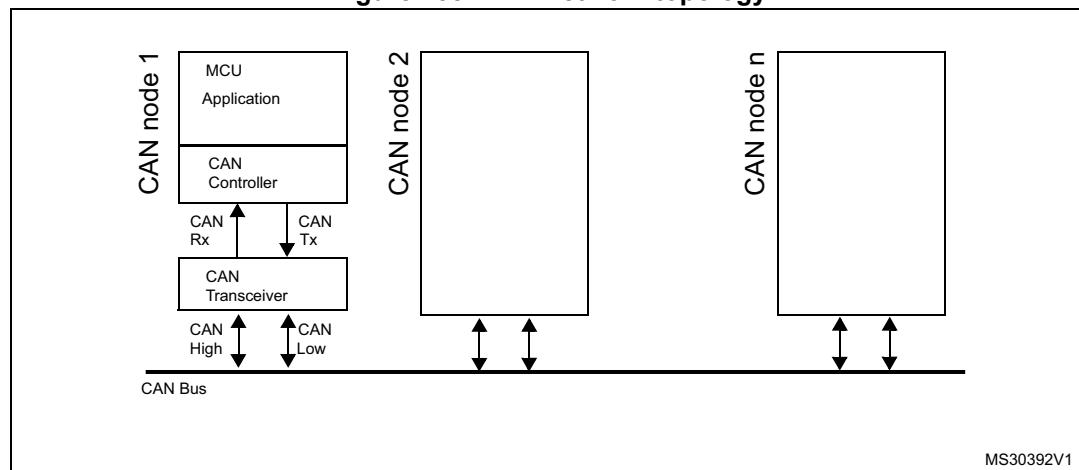
An enhanced filtering mechanism is required to handle each type of message.

Furthermore, application tasks require more CPU time, therefore real-time constraints caused by message reception must 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 438. CAN network topology**



#### 36.3.1 CAN 2.0B active core

The bxCAN module handles the transmission and the reception of CAN messages autonomously. Standard identifiers (11-bit) and extended identifiers (29-bit) are fully supported by hardware.

#### 36.3.2 Control, status, and configuration registers

The application uses these registers to:

- Configure CAN parameters, for example baud rate
- Request transmissions
- Handle receptions
- Manage interrupts
- Get diagnostic information

#### 36.3.3 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.

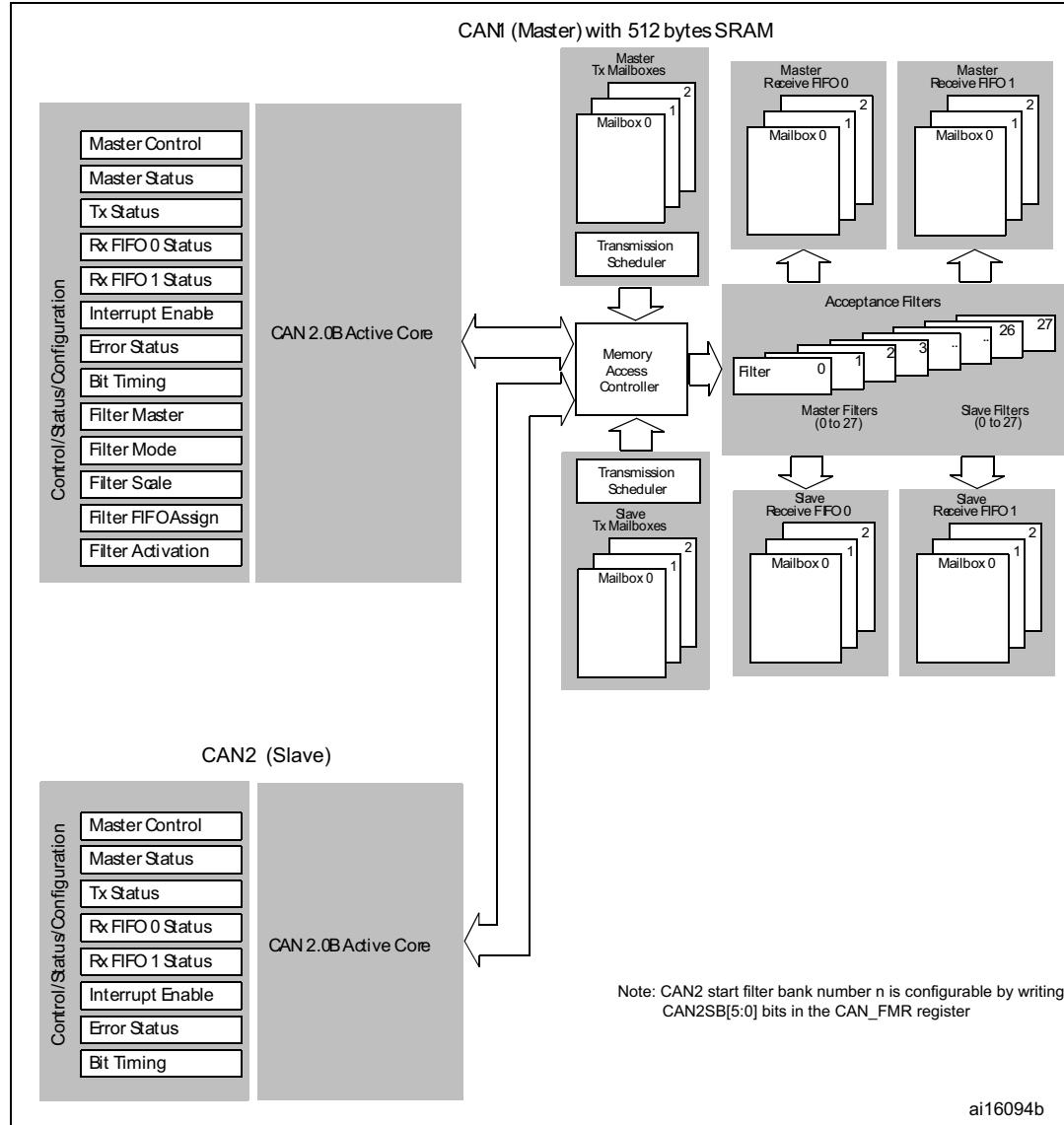
### 36.3.4 Acceptance filters

The bxCAN provides up to 28 scalable/configurable identifier filter banks in dual CAN configuration, for selecting the incoming messages, that the software needs and discarding the others.

#### Receive FIFO

Two receive FIFOs are used by hardware to store the incoming messages. Three complete messages can be stored in each FIFO. The FIFOs are managed completely by hardware.

**Figure 439. Dual-CAN block diagram**



## 36.4 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 is set, bxCAN is in **normal** mode. Before entering **normal** mode, bxCAN always must **synchronize** on the CAN bus. To synchronize, bxCAN waits until the CAN bus is idle, this means that 11 consecutive recessive bits have been monitored on CANRX.

### 36.4.1 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).*

### 36.4.2 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.

### 36.4.3 Sleep mode (low-power)

To reduce power consumption, bxCAN has a low-power mode called Sleep mode. This mode is entered on software request by setting the SLEEP bit in the CAN\_MCR register. In this mode, the bxCAN clock is stopped, however software can still access the bxCAN mailboxes.

If software requests entry to **initialization** mode by setting the INRQ bit while bxCAN is in **Sleep** mode, it must also clear the SLEEP bit.

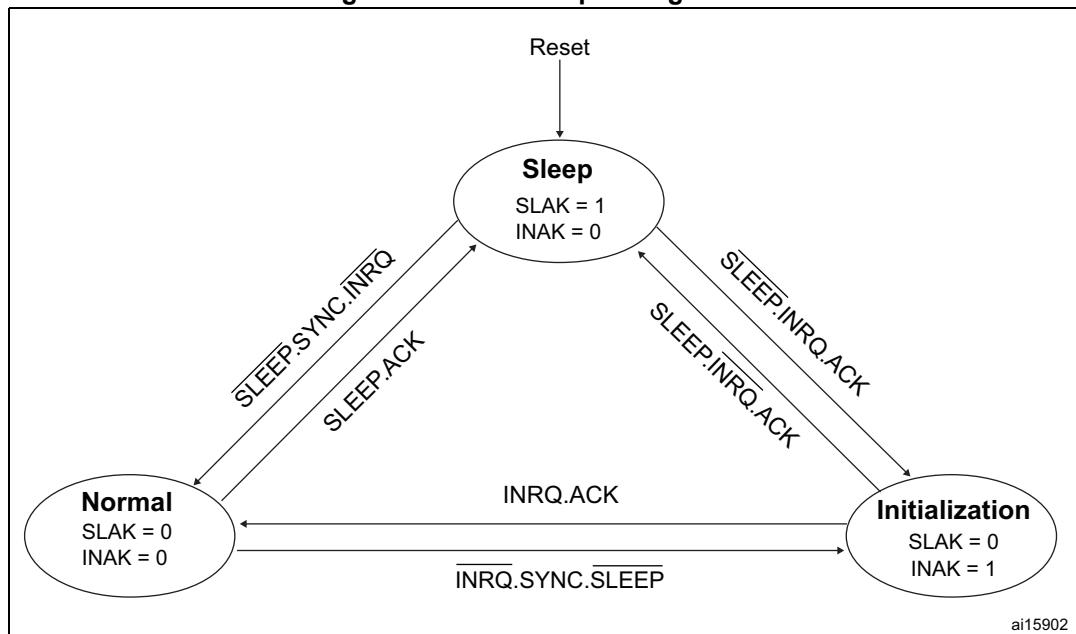
bxCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on detection of CAN bus activity.

On CAN bus activity detection, hardware automatically performs the wake-up sequence by clearing the SLEEP bit if the AWUM bit in the CAN\_MCR register is set. If the AWUM bit is cleared, software has to clear the SLEEP bit when a wake-up interrupt occurs, in order to exit from Sleep mode.

**Note:** *If the wake-up interrupt is enabled (WKUIE bit set in CAN\_IER register) a wake-up interrupt is generated on detection of CAN bus activity, even if the bxCAN automatically performs the wake-up sequence.*

After the SLEEP bit has been cleared, Sleep mode is exited once bxCAN has synchronized with the CAN bus, refer to [Figure 440](#). The Sleep mode is exited once the SLAK bit has been cleared by hardware.

**Figure 440. 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.

## 36.5 Test mode

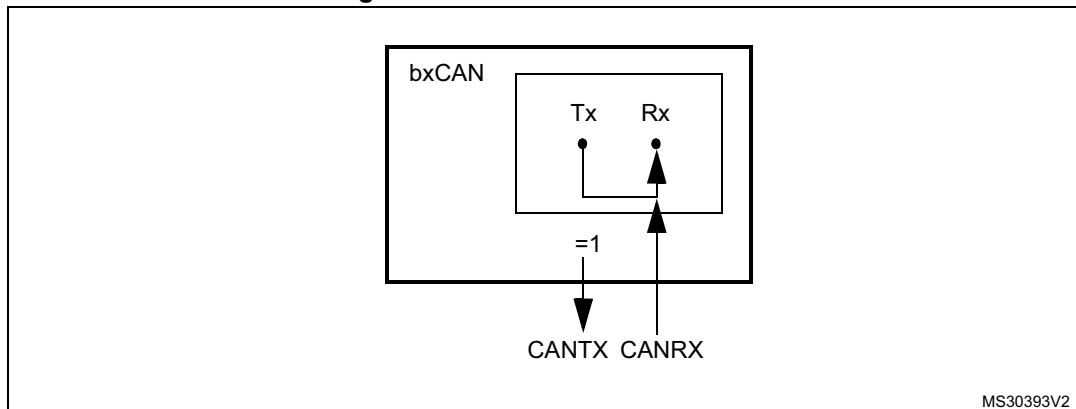
Test mode can be selected by the SILM and LBKM bits in the CAN\_BTR register. These bits must be configured while bxCAN is in Initialization mode. Once test mode has been selected, the INRQ bit in the CAN\_MCR register must be reset to enter Normal mode.

### 36.5.1 Silent mode

The bxCAN can be put in Silent mode by setting the SILM bit in the CAN\_BTR register.

In Silent mode, the bxCAN is able to receive valid data frames and valid remote frames, but sends only recessive bits on the CAN bus and cannot start a transmission. If the bxCAN has to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. Silent mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge bits, Error frames).

**Figure 441. bxCAN in silent mode**

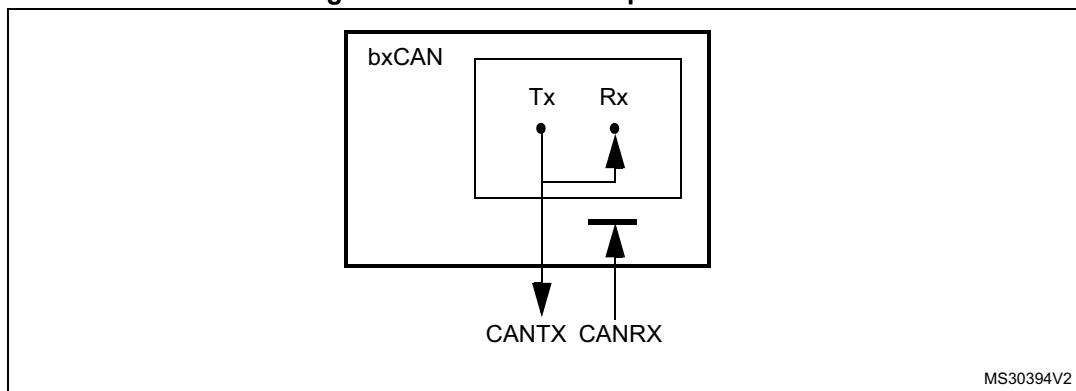


MS30393V2

### 36.5.2 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 442. bxCAN in Loop back mode**



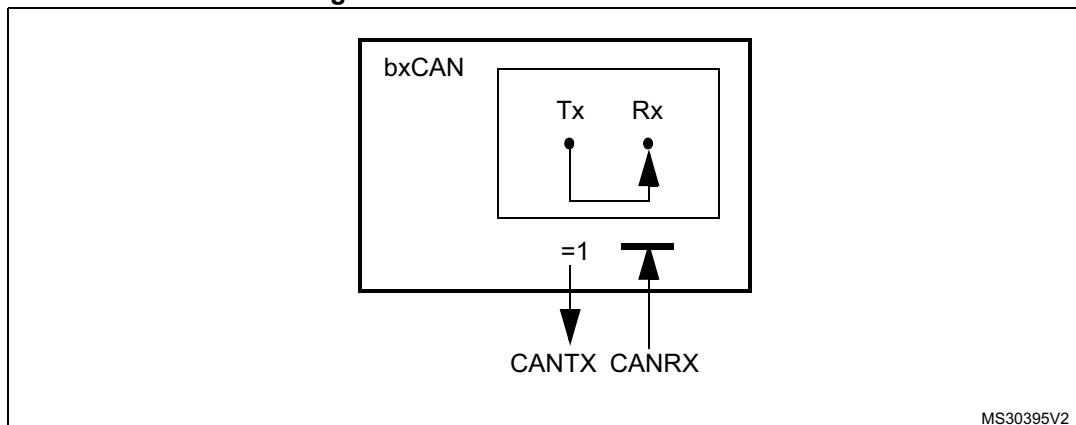
MS30394V2

This mode is provided for self-test functions. To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in Loop back mode. In this mode, the bxCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is disregarded by the bxCAN. The transmitted messages can be monitored on the CANTX pin.

### 36.5.3 Loop back combined with silent mode

It is also possible to combine Loop back mode and Silent mode by setting the LBKM and SILM bits in the CAN\_BTR register. This mode can be used for a “Hot Selftest”, meaning the bxCAN can be tested like in Loop back mode but without affecting a running CAN system connected to the CANTX and CANRX pins. In this mode, the CANRX pin is disconnected from the bxCAN and the CANTX pin is held recessive.

Figure 443. bxCAN in combined mode



## 36.6 Behavior in debug mode

When the microcontroller enters the debug mode (Cortex<sup>®</sup>-M7 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 for the dual mode.
- the DBF bit in CAN\_MCR. For more details, refer to [Section 36.9.2: CAN control and status registers](#).

## 36.7 bxCAN functional description

### 36.7.1 Transmission handling

To transmit a message, the application must select one **empty** transmit mailbox, set up the identifier, the data length code (DLC), and the data before requesting the transmission by setting the corresponding TXRQ bit in the CAN\_TlxR register. Once the mailbox has left **empty** state, the software no longer has write access to the mailbox registers. Immediately after the TXRQ bit has been set, the mailbox enters **Pending** state and waits to become the highest priority mailbox, see *Transmit Priority*. As soon as the mailbox has the highest priority it is **Scheduled** for transmission. The transmission of the message of the scheduled

mailbox starts (enter **transmit** state) when the CAN bus becomes idle. Once the mailbox has been successfully transmitted, it becomes **empty** again. The hardware indicates a successful transmission by setting the RQCP and TXOK bits in the CAN\_TSR register.

If the transmission fails, the cause is indicated by the ALST bit in the CAN\_TSR register in case of an Arbitration Lost, and/or the TERR bit, in case of transmission error detection.

### Transmit priority

- By identifier

When more than one transmit mailbox is pending, the transmission order is given by the identifier of the message stored in the mailbox. The message with the lowest identifier value has the highest priority according to the arbitration of the CAN protocol. If the identifier values are equal, the lower mailbox number is scheduled first.

- By transmit request order

The transmit mailboxes can be configured as a transmit FIFO by setting the TXFP bit in the CAN\_MCR register. In this mode, the priority order is given by the transmit request order. This mode is very useful for segmented transmission.

### Abort

A transmission request can be aborted by the user setting the ABRQ bit in the CAN\_TSR register. In **Pending** or **Scheduled** state, the mailbox is aborted immediately. An abort request while the mailbox is in **transmit** state can have two results. If the mailbox is transmitted successfully the mailbox becomes **empty** with the TXOK bit set in the CAN\_TSR register. If the transmission fails, the mailbox becomes **Scheduled**, the transmission is aborted and becomes **empty** with TXOK cleared. In all cases, the mailbox becomes **empty** again at least at the end of the current transmission.

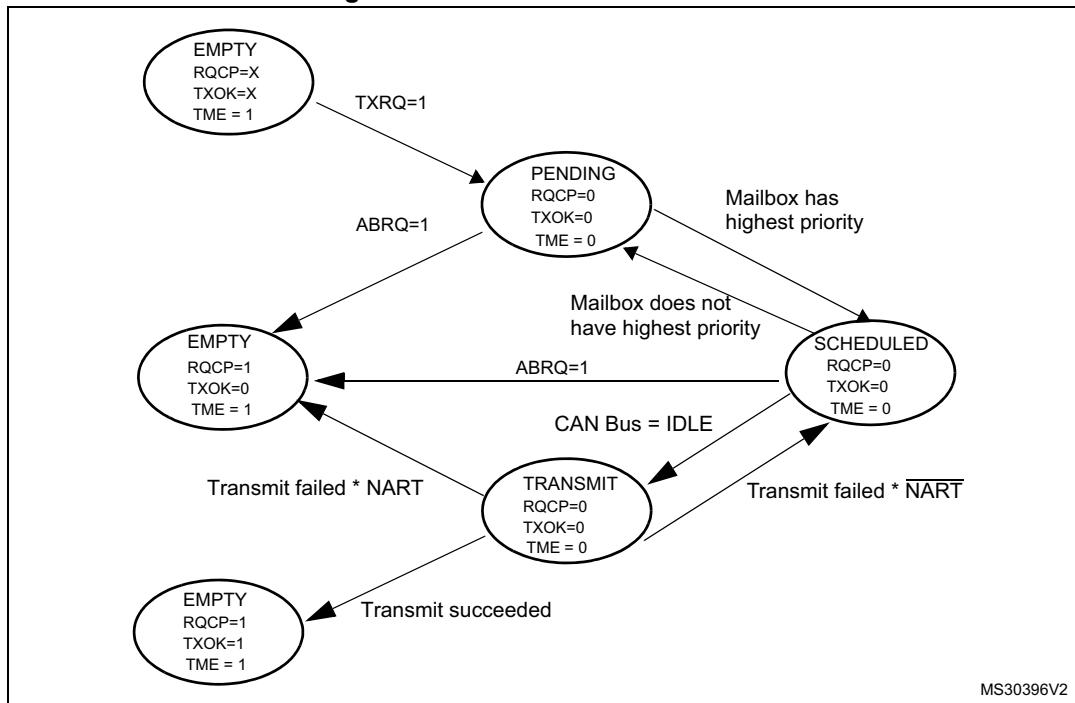
### Non automatic retransmission mode

This mode has been implemented in order to fulfill the requirement of the Time Triggered Communication option of the CAN standard. To configure the hardware in this mode the NART bit in the CAN\_MCR register must be set.

In this mode, each transmission is started only once. If the first attempt fails, due to an arbitration loss or an error, the hardware does not automatically restart the message transmission.

At the end of the first transmission attempt, the hardware considers the request as completed and sets the RQCP bit in the CAN\_TSR register. The result of the transmission is indicated in the CAN\_TSR register by the TXOK, ALST, and TERR bits.

Figure 444. Transmit mailbox states



### 36.7.2 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 36.7.7](#)). The internal counter is captured on the sample point of the Start Of Frame bit in both reception and transmission.

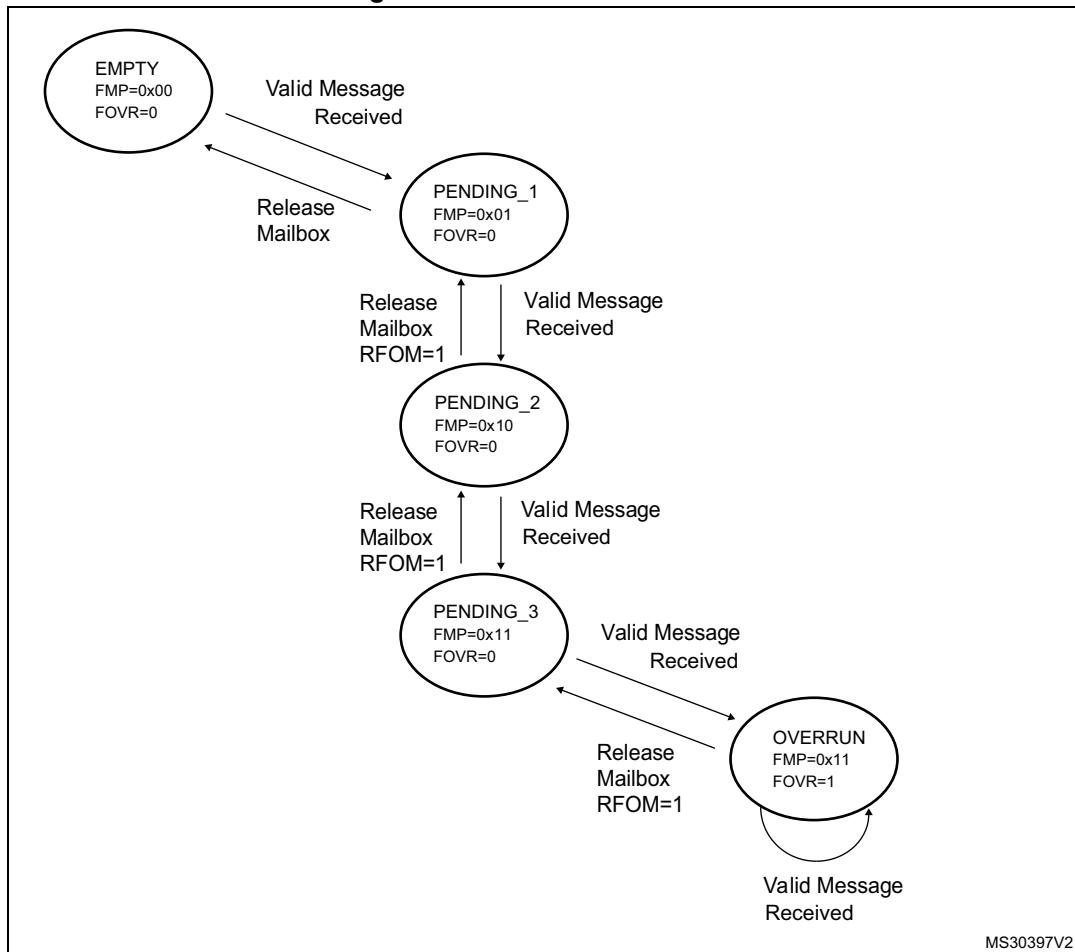
### 36.7.3 Reception handling

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 36.7.4](#).

Figure 445. 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 is stored in the FIFO, which enters **pending\_2** state (FMP[1:0] = 10b). The storage process is repeated for the next valid message putting the FIFO into **pending\_3** state (FMP[1:0] = 11b). At this point, the software must release the output mailbox by setting the RFOM bit, so that a mailbox is free to store the next valid message. Otherwise, the next valid message received causes a loss of message. Refer also to [Section 36.7.5](#).

## Overrun

Once the FIFO is in **pending\_3** state (the three mailboxes are full) the next valid message reception leads to an **overrun**, and a message is lost. The hardware signals the overrun

condition by setting the FOVR bit in the CAN\_RFR register. Which message is lost depends on the configuration of the FIFO:

- If the FIFO lock function is disabled (RFLM bit in the CAN\_MCR register cleared) the last message stored in the FIFO is overwritten by the new incoming message. In this case, the latest messages are always available to the application.
- If the FIFO lock function is enabled (RFLM bit in the CAN\_MCR register set) the most recent message is discarded, and the software has the three oldest messages in the FIFO available.

### Reception related interrupts

Once a message has been stored in the FIFO, the FMP[1:0] bits are updated, and an interrupt request is generated if the FMPIE bit in the CAN\_IER register is set.

When the FIFO becomes full (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.

## 36.7.4 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 446](#).

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 446](#). 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.

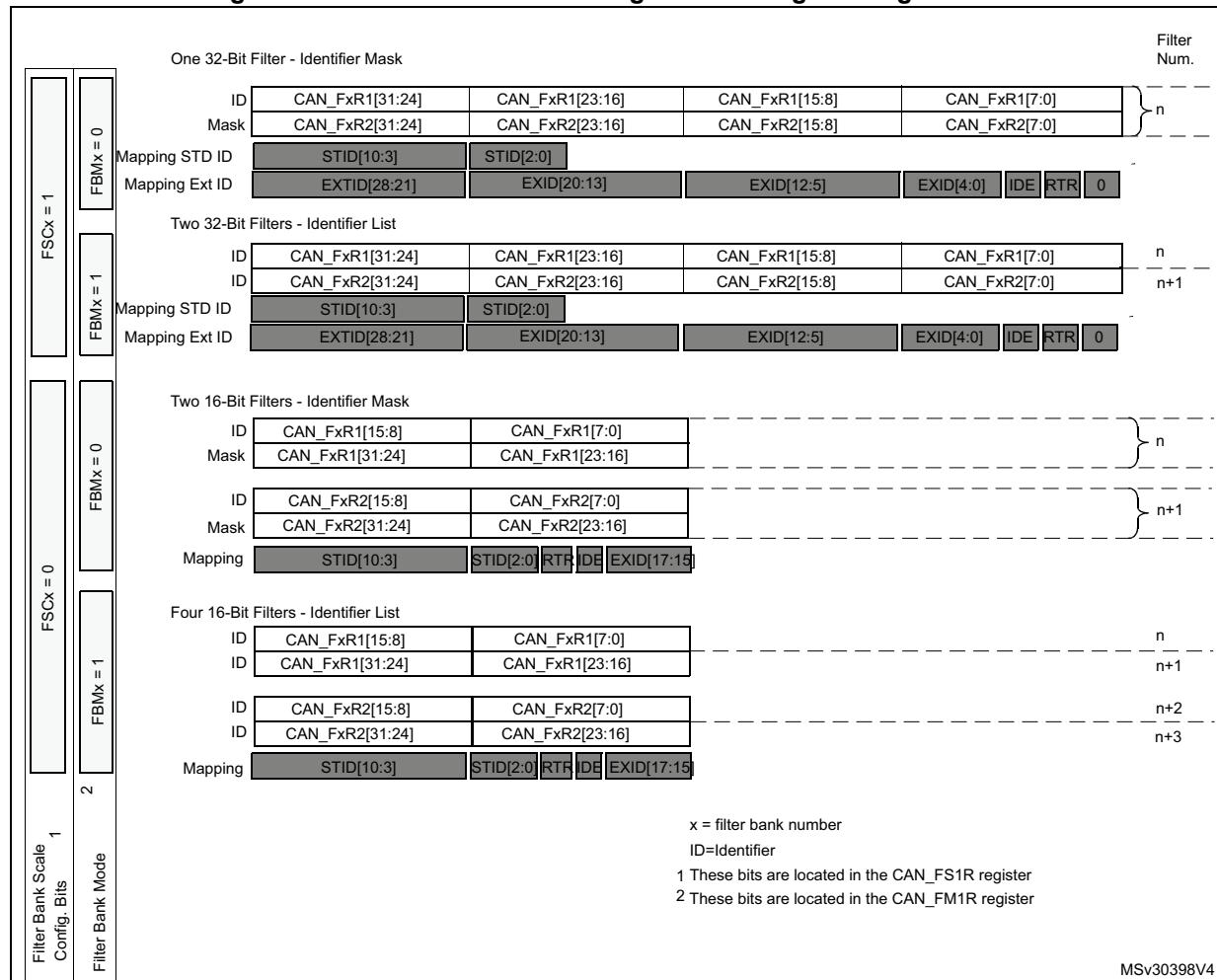
To select single identifiers, configure the Mask/Identifier registers in identifier list mode.

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 446](#).

**Figure 446. Filter bank scale configuration - Register organization**



## Filter match index

Once a message has been received in the FIFO it is available to the application. Typically, application data is copied into SRAM locations. To copy the data to the right location the application has to identify the data by means of the identifier. To avoid this, and to ease the access to the SRAM locations, the CAN controller provides a filter match index.

This index is stored in the mailbox together with the message according to the filter priority rules. Thus each received message has its associated filter match index.

The filter match index can be used in two ways:

- Compare the filter match index with a list of expected values.
- Use the filter match index as an index on an array to access the data destination location.

For non masked filters, the software no longer has to compare the identifier.

If the filter is masked the software reduces the comparison to the masked bits only.

The index value of the filter number does not take into account the activation state of the filter banks. In addition, two independent numbering schemes are used, one for each FIFO. Refer to [Figure 447](#) for an example.

**Figure 447. Example of filter numbering**

Filter Bank	FIFO0	Filter Num.	Filter Bank	FIFO1	Filter Num.
0	ID List (32-bit)	0 1	2	ID Mask (16-bit)	0 1
1	ID Mask (32-bit)	2	4	ID List (32-bit)	2 3
3	ID List (16-bit)	3 4 5 6	7	Deactivated ID Mask (16-bit)	4 5
5	Deactivated ID List (32-bit)	7 8	8	ID Mask (16-bit)	6 7
6	ID Mask (16-bit)	9 10	10	Deactivated ID List (16-bit)	8 9 10 11
9	ID List (32-bit)	11 12	11	ID List (32-bit)	12 13
13	ID Mask (32-bit)	13	12	ID Mask (32-bit)	14

ID=Identifier

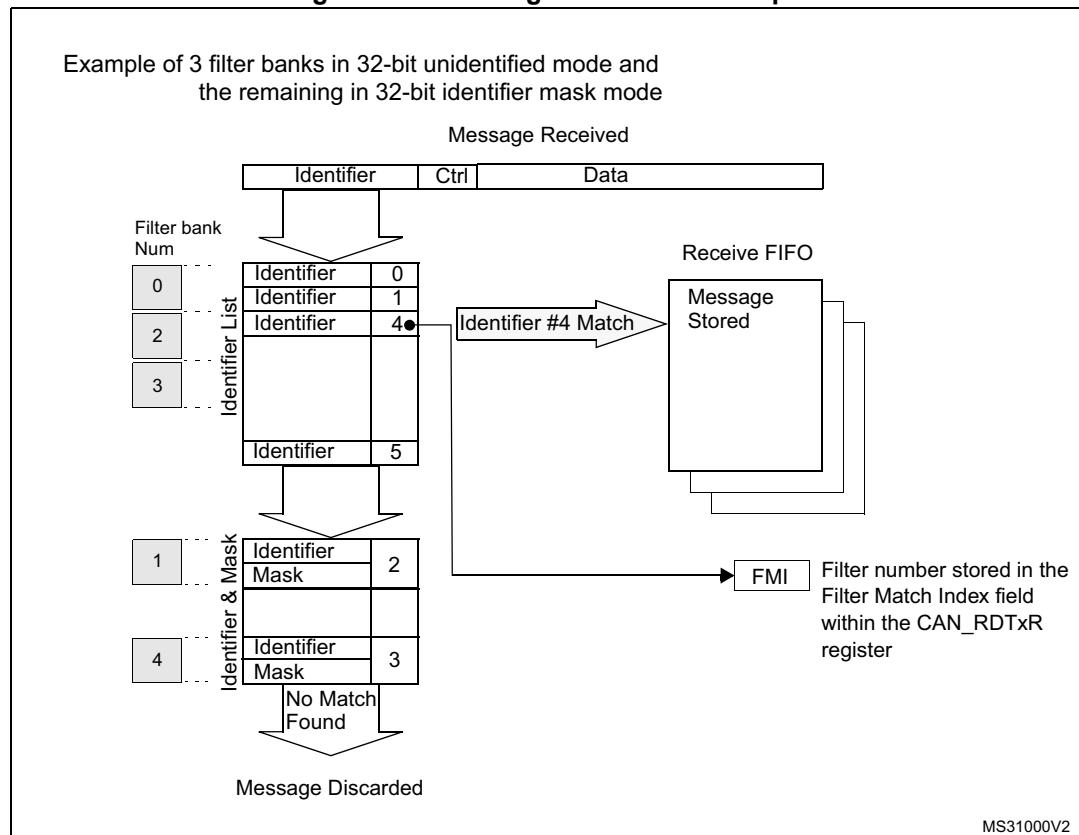
MS30399V2

## Filter priority rules

Depending on the filter combination it may occur that an identifier passes successfully through several filters. In this case the filter match value stored in the receive mailbox is chosen according to the following priority rules:

- A 32-bit filter takes priority over a 16-bit filter.
- For filters of equal scale, priority is given to the Identifier List mode over the Identifier Mask mode
- For filters of equal scale and mode, priority is given by the filter number (the lower the number, the higher the priority).

**Figure 448. 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 #4 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.

### 36.7.5 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 258. Transmit mailbox mapping**

Offset to transmit mailbox base address	Register name
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDhxR

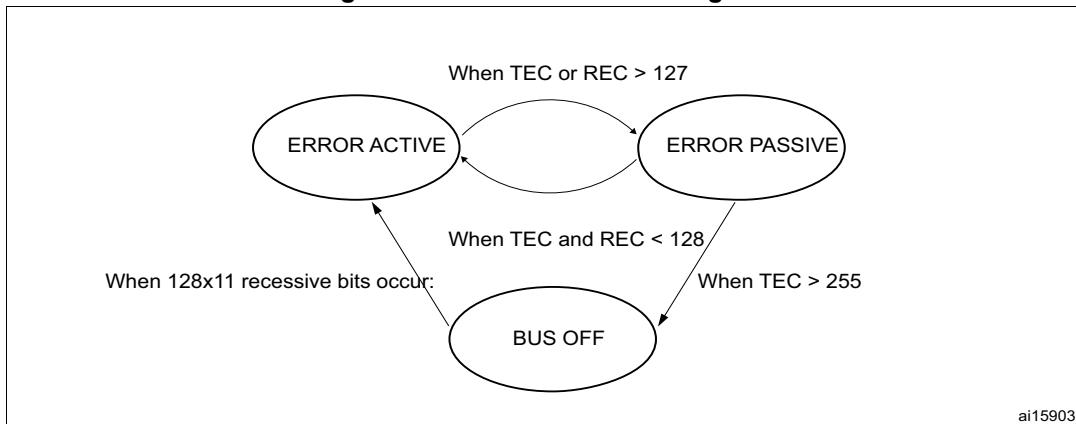
#### Receive mailbox

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 259. 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 449. CAN error state diagram



ai15903

### 36.7.6 Error management

The error management as described in the CAN protocol is handled entirely by hardware using a Transmit Error Counter (TEC value, in CAN\_ESR register) and a Receive Error Counter (REC value, in the CAN\_ESR register), which get incremented or decremented according to the error condition. For detailed information about TEC and REC management, refer to the CAN standard.

Both of them may be read by software to determine the stability of the network. Furthermore, the CAN hardware provides detailed information on the current error status in CAN\_ESR register. By means of the CAN\_IER register (ERRIE bit, etc.), the software can configure the interrupt generation on error detection in a very flexible way.

#### Bus-Off recovery

The Bus-Off state is reached when TEC is greater than 255, this state is indicated by BOFF bit in CAN\_ESR register. In Bus-Off state, the bxCAN is no longer able to transmit and receive messages.

Depending on the ABOM bit in the CAN\_MCR register, bxCAN recovers from Bus-Off (become error active again) either automatically or on software request. But in both cases the bxCAN has to wait at least for the recovery sequence specified in the CAN standard (128 occurrences of 11 consecutive recessive bits monitored on CANRX).

If ABOM is set, the bxCAN starts the recovering sequence automatically after it has entered Bus-Off state.

If ABOM is cleared, the software must initiate the recovering sequence, by requesting bxCAN to enter and to leave initialization mode.

**Note:** *In initialization mode, bxCAN does not monitor the CANRX signal, therefore it cannot complete the recovery sequence. To recover, bxCAN must be in normal mode.*

### 36.7.7 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.

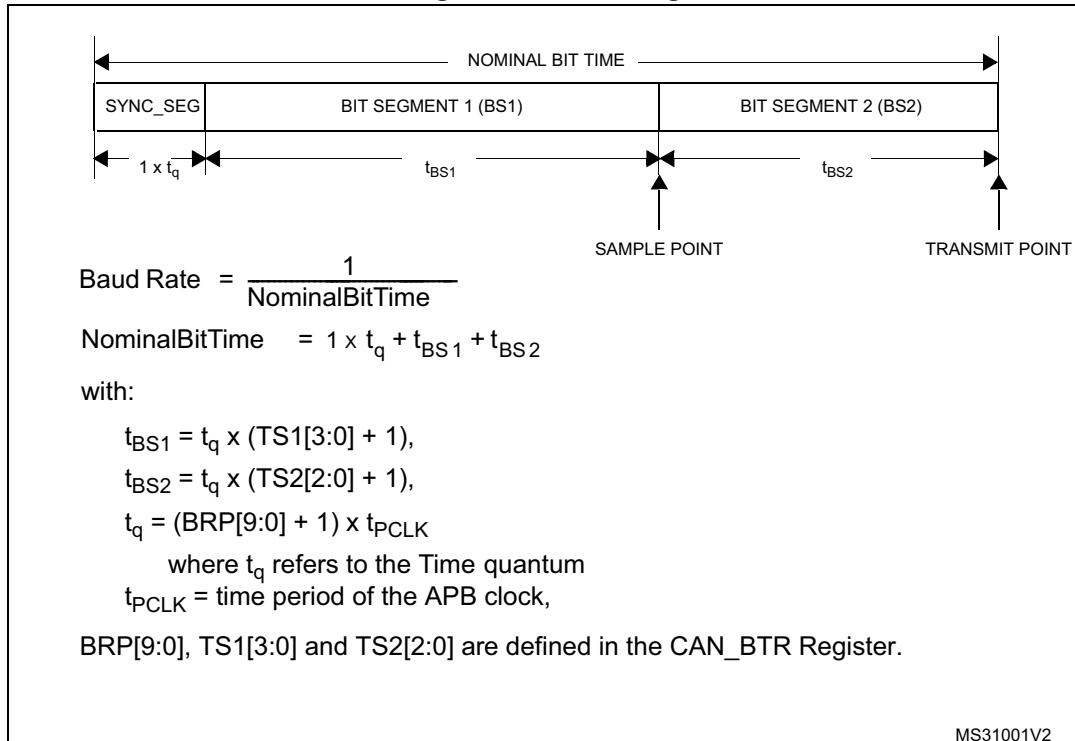
If a valid edge is detected in BS1 instead of SYNC\_SEG, BS1 is extended by up to SJW so that the sample point is delayed.

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.

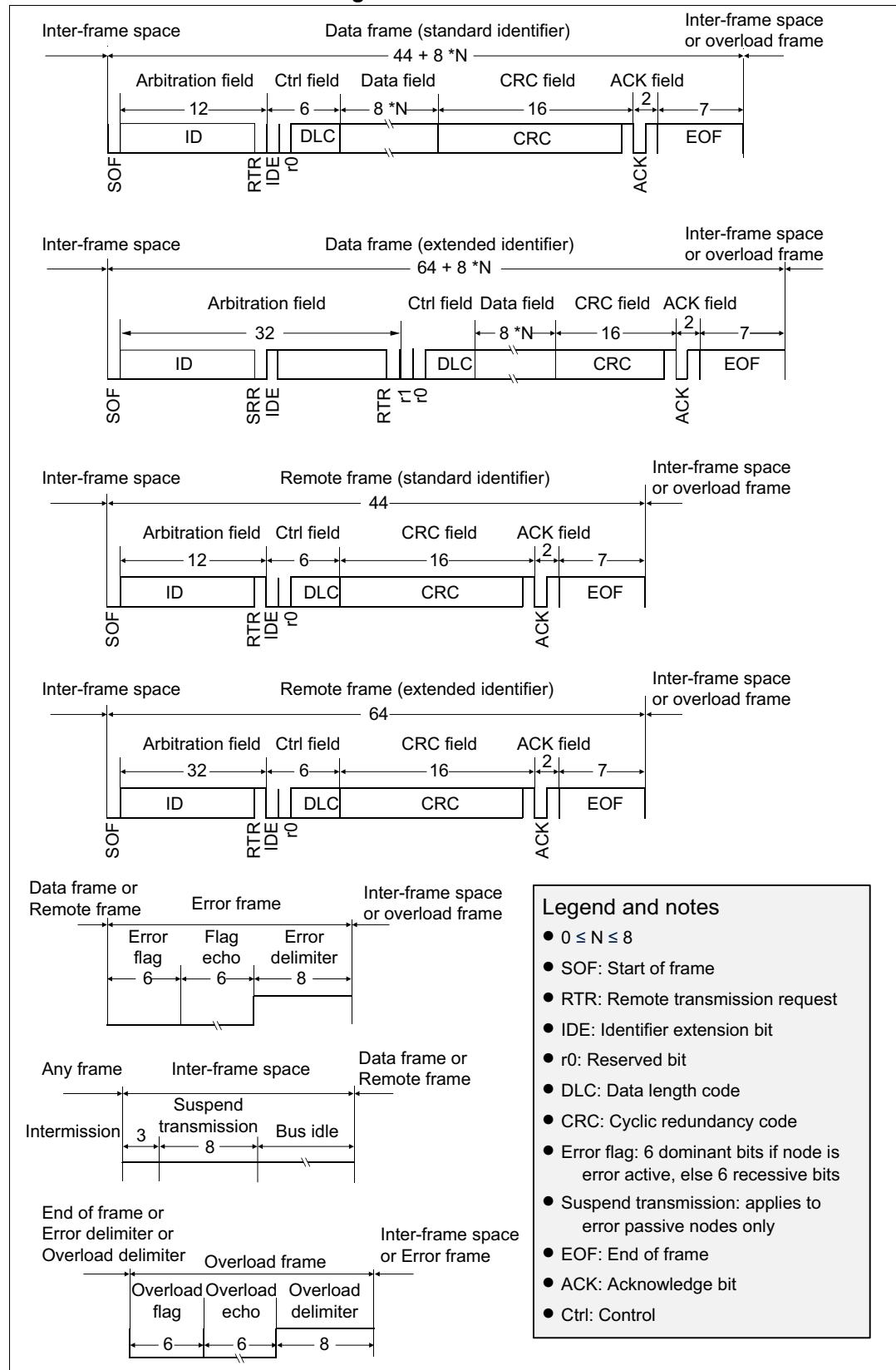
*Note:* *For a detailed description of the CAN bit timing and resynchronization mechanism, refer to the ISO 11898 standard.*

Figure 450. Bit timing



MS31001V2

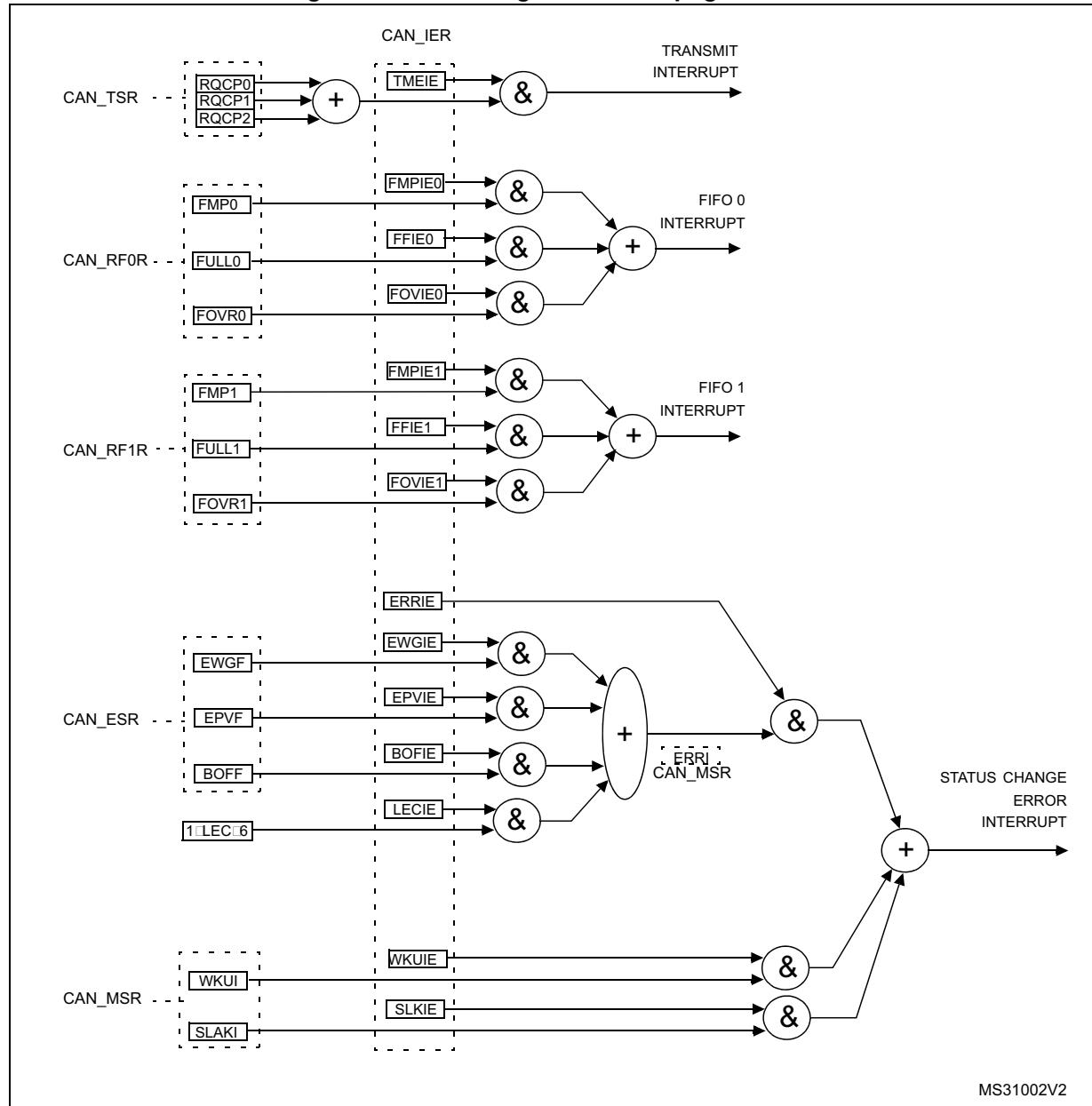
Figure 451. CAN frames



## 36.8 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 452. Event flags and interrupt generation**



- The **transmit interrupt** can be generated by the following events:
  - Transmit mailbox 0 becomes empty, RQCP0 bit in the CAN\_TSR register set.
  - Transmit mailbox 1 becomes empty, RQCP1 bit in the CAN\_TSR register set.
  - Transmit mailbox 2 becomes empty, RQCP2 bit in the CAN\_TSR register set.
- The **FIFO 0 interrupt** can be generated by the following events:
  - Reception of a new message, FMP0 bits in the CAN\_RF0R register are not '00'.
  - FIFO0 full condition, FULL0 bit in the CAN\_RF0R register set.
  - FIFO0 overrun condition, FOVR0 bit in the CAN\_RF0R register set.
- The **FIFO 1 interrupt** can be generated by the following events:
  - Reception of a new message, FMP1 bits in the CAN\_RF1R register are not '00'.
  - FIFO1 full condition, FULL1 bit in the CAN\_RF1R register set.
  - FIFO1 overrun condition, FOVR1 bit in the CAN\_RF1R register set.
- The **error and status change interrupt** can be generated by the following events:
  - Error condition, for more details on error conditions refer to the CAN Error Status register (CAN\_ESR).
  - Wake-up condition, SOF monitored on the CAN Rx signal.
  - Entry into Sleep mode.

## 36.9 CAN registers

The peripheral registers must be accessed by words (32 bits).

### 36.9.1 Register access protection

Erroneous access to certain configuration registers can cause the hardware to temporarily disturb the whole CAN network. Therefore, the CAN\_BTR register can be modified by software only while the CAN hardware is in initialization mode.

Although the transmission of incorrect data does not cause problems at the CAN network level, it can severely disturb the application. A transmit mailbox can be only modified by software while it is in empty state, refer to [Figure 444: 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.

### 36.9.2 CAN control and status registers

Refer to [Section 1.2](#) for a list of abbreviations used in register descriptions.

#### CAN master control register (CAN\_MCR)

Address offset: 0x00

Reset value: 0x0001 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBF
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	Res.	TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ						
rs								rw	rw						

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 **DBF:** Debug freeze

- 0: CAN working during debug
- 1: CAN reception/transmission frozen during debug. Reception FIFOs can still be accessed/controlled normally.

Bit 15 **RESET:** bxCAN software master reset

- 0: Normal operation.
- 1: Force a master reset of the bxCAN -> Sleep mode activated after reset (FMP bits and CAN\_MCR register are initialized to the reset values). This bit is automatically reset to 0.

Bits 14:8 Reserved, must be kept at reset value.

Bit 7 **TTCM:** Time triggered communication mode

- 0: Time Triggered Communication mode disabled.
- 1: Time Triggered Communication mode enabled

*Note: For more information on Time Triggered Communication mode, refer to Section 36.7.2: 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 36.7.6: Error management](#).

Bit 5 **AWUM:** Automatic wake-up mode

- This bit controls the behavior of the CAN hardware on message reception during Sleep mode.
  - 0: The Sleep mode is left on software request by clearing the SLEEP bit of the CAN\_MCR register.
  - 1: The Sleep mode is left automatically by hardware on CAN message detection.
- The SLEEP bit of the CAN\_MCR register and the SLAK bit of the CAN\_MSR register are cleared by hardware.

Bit 4 **NART:** No automatic retransmission

- 0: The CAN hardware automatically retransmits the message until it has been successfully transmitted according to the CAN standard.
- 1: A message is transmitted only once, independently of the transmission result (successful, error or arbitration lost).

Bit 3 **RFLM**: Receive FIFO locked mode

0: Receive FIFO not locked on overrun. Once a receive FIFO is full the next incoming message overwrites the previous one.

1: Receive FIFO locked against overrun. Once a receive FIFO is full the next incoming message is discarded.

Bit 2 **TXFP**: Transmit FIFO priority

This bit controls the transmission order when several mailboxes are pending at the same time.

0: Priority driven by the identifier of the message

1: Priority driven by the request order (chronologically)

Bit 1 **SLEEP**: Sleep mode request

This bit is set by software to request the CAN hardware to enter the Sleep mode. Sleep mode is entered as soon as the current CAN activity (transmission or reception of a CAN frame) has been completed.

This bit is cleared by software to exit Sleep mode.

This bit is cleared by hardware when the AWUM bit is set and a SOF bit is detected on the CAN Rx signal.

This bit is set after reset - CAN starts in Sleep mode.

Bit 0 **INRQ**: Initialization request

The software clears this bit to switch the hardware into normal mode. Once 11 consecutive recessive bits have been monitored on the Rx signal the CAN hardware is synchronized and ready for transmission and reception. Hardware signals this event by clearing the INAK bit in the CAN\_MSR register.

Software sets this bit to request the CAN hardware to enter initialization mode. Once software has set the INRQ bit, the CAN hardware waits until the current CAN activity (transmission or reception) is completed before entering the initialization mode. Hardware signals this event by setting the INAK bit in the CAN\_MSR register.

## CAN master status register (CAN\_MSR)

Address offset: 0x04

Reset value: 0x0000 0C02

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	RX	SAMP	RXM	TXM	Res.	Res.	Res.	SLAKI	WKUI	ERRI	SLAK	INAK
				r	r	r	r				rc_w1	rc_w1	rc_w1	r	r

Bits 31:12 Reserved, must be kept at reset value.

Bit 11 **RX**: CAN Rx signal

Monitors the actual value of the **CAN\_RX** Pin.

Bit 10 **SAMP**: Last sample point

The value of RX on the last sample point (current received bit value).

**Bit 9 RXM:** Receive mode

The CAN hardware is currently receiver.

**Bit 8 TXM:** Transmit mode

The CAN hardware is currently transmitter.

Bits 7:5 Reserved, must be kept at reset value.

**Bit 4 SLAKI:** Sleep acknowledge interrupt

When SLKIE=1, this bit is set by hardware to signal that the bxCAN has entered Sleep mode. When set, this bit generates a status change interrupt if the SLKIE bit in the CAN\_IER register is set.

This bit is cleared by software or by hardware, when SLAK is cleared.

*Note: When SLKIE=0, no polling on SLAKI is possible. In this case the SLAK bit can be polled.*

**Bit 3 WKUI:** Wake-up interrupt

This bit is set by hardware to signal that a SOF bit has been detected while the CAN hardware was in Sleep mode. Setting this bit generates a status change interrupt if the WKUIE bit in the CAN\_IER register is set.

This bit is cleared by software.

**Bit 2 ERRI:** Error interrupt

This bit is set by hardware when a bit of the CAN\_ESR has been set on error detection and the corresponding interrupt in the CAN\_IER is enabled. Setting this bit generates a status change interrupt if the ERRIE bit in the CAN\_IER register is set.

This bit is cleared by software.

**Bit 1 SLAK:** Sleep acknowledge

This bit is set by hardware and indicates to the software that the CAN hardware is now in Sleep mode. This bit acknowledges the Sleep mode request from the software (set SLEEP bit in CAN\_MCR register).

This bit is cleared by hardware when the CAN hardware has left Sleep mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

*Note: The process of leaving Sleep mode is triggered when the SLEEP bit in the CAN\_MCR register is cleared. Refer to the AWUM bit of the CAN\_MCR register description for detailed information for clearing SLEEP bit*

**Bit 0 INAK:** Initialization acknowledge

This bit is set by hardware and indicates to the software that the CAN hardware is now in initialization mode. This bit acknowledges the initialization request from the software (set INRQ bit in CAN\_MCR register).

This bit is cleared by hardware when the CAN hardware has left the initialization mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

## CAN transmit status register (CAN\_TSR)

Address offset: 0x08

Reset value: 0x1C00 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOW2	LOW1	LOW0	TME2	TME1	TME0	CODE[1:0]		ABRQ2	Res.	Res.	Res.	TERR2	ALST2	TXOK2	RQCP2
r	r	r	r	r	r	r	r	rs				rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRQ1	Res.	Res.	Res.	TERR1	ALST1	TXOK1	RQCP1	ABRQ0	Res.	Res.	Res.	TERR0	ALST0	TXOK0	RQCP0
rs				rc_w1	rc_w1	rc_w1	rc_w1	rs				rc_w1	rc_w1	rc_w1	rc_w1

Bit 31 **LOW2**: Lowest priority flag for mailbox 2

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 2 has the lowest priority.

Bit 30 **LOW1**: Lowest priority flag for mailbox 1

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 1 has the lowest priority.

Bit 29 **LOW0**: Lowest priority flag for mailbox 0

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 0 has the lowest priority.

*Note: The LOW[2:0] bits are set to 0 when only one mailbox is pending.*

Bit 28 **TME2**: Transmit mailbox 2 empty

This bit is set by hardware when no transmit request is pending for mailbox 2.

Bit 27 **TME1**: Transmit mailbox 1 empty

This bit is set by hardware when no transmit request is pending for mailbox 1.

Bit 26 **TME0**: Transmit mailbox 0 empty

This bit is set by hardware when no transmit request is pending for mailbox 0.

Bits 25:24 **CODE[1:0]**: Mailbox code

In case at least one transmit mailbox is free, the code value is equal to the number of the next transmit mailbox free.

In case all transmit mailboxes are pending, the code value is equal to the number of the transmit mailbox with the lowest priority.

Bit 23 **ABRQ2**: Abort request for mailbox 2

Set by software to abort the transmission request for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

Setting this bit has no effect when the mailbox is not pending for transmission.

Bits 22:20 Reserved, must be kept at reset value.

Bit 19 **TERR2**: Transmission error of mailbox 2

This bit is set when the previous TX failed due to an error.

Bit 18 **ALST2**: Arbitration lost for mailbox 2

This bit is set when the previous TX failed due to an arbitration lost.

Bit 17 **TXOK2**: Transmission OK of mailbox 2

The hardware updates this bit after each transmission attempt.

0: The previous transmission failed

1: The previous transmission was successful

This bit is set by hardware when the transmission request on mailbox 2 has been completed successfully. Refer to [Figure 444](#).

- 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 444](#).
- Bit 8 **RQCP1**: Request completed mailbox1  
Set by hardware when the last request (transmit or abort) has been performed.  
Cleared by software writing a “1” or by hardware on transmission request (TXRQ1 set in CAN\_TMRD1R register).  
Clearing this bit clears all the status bits (TXOK1, ALST1 and TERR1) for Mailbox 1.
- Bit 7 **ABRQ0**: Abort request for mailbox0  
Set by software to abort the transmission request for the corresponding mailbox.  
Cleared by hardware when the mailbox becomes empty.  
Setting this bit has no effect when the mailbox is not pending for transmission.
- Bits 6:4 Reserved, must be kept at reset value.
- Bit 3 **TERR0**: Transmission error of mailbox0  
This bit is set when the previous TX failed due to an error.
- Bit 2 **ALST0**: Arbitration lost for mailbox0  
This bit is set when the previous TX failed due to an arbitration lost.
- Bit 1 **TXOK0**: Transmission OK of mailbox0  
The hardware updates this bit after each transmission attempt.  
0: The previous transmission failed  
1: The previous transmission was successful  
This bit is set by hardware when the transmission request on mailbox 0 has been completed successfully. Refer to [Figure 444](#).
- Bit 0 **RQCP0**: Request completed mailbox0  
Set by hardware when the last request (transmit or abort) has been performed.  
Cleared by software writing a “1” or by hardware on transmission request (TXRQ0 set in CAN\_TMRD0R register).  
Clearing this bit clears all the status bits (TXOK0, ALST0 and TERR0) for Mailbox 0.

**CAN receive FIFO 0 register (CAN\_RF0R)**

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RFOM0	FOVR0	FULL0	Res.	FMP0[1:0]										
										rs	rc_w1	rc_w1		r	r

Bits 31:6 Reserved, must be kept at reset value.

**Bit 5 RFOM0: Release FIFO 0 output mailbox**

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

**Bit 4 FOVR0: FIFO 0 overrun**

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software.

**Bit 3 FULL0: FIFO 0 full**

Set by hardware when three messages are stored in the FIFO.

This bit is cleared by software.

**Bit 2 Reserved, must be kept at reset value.****Bits 1:0 FMP0[1:0]: FIFO 0 message pending**

These bits indicate how many messages are pending in the receive FIFO.

FMP is increased each time the hardware stores a new message in to the FIFO. FMP is decreased each time the software releases the output mailbox by setting the RFOM0 bit.

**CAN receive FIFO 1 register (CAN\_RF1R)**

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RFOM1	FOVR1	FULL1	Res.	FMP1[1:0]										
										rs	rc_w1	rc_w1		r	r

Bits 31:6 Reserved, must be kept at reset value.

**Bit 5 RFOM1:** Release FIFO 1 output mailbox

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

**Bit 4 FOVR1:** FIFO 1 overrun

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software.

**Bit 3 FULL1:** FIFO 1 full

Set by hardware when three messages are stored in the FIFO.

This bit is cleared by software.

Bit 2 Reserved, must be kept at reset value.

**Bits 1:0 FMP1[1:0]:** FIFO 1 message pending

These bits indicate how many messages are pending in the receive FIFO1.

FMP1 is increased each time the hardware stores a new message in to the FIFO1. FMP is decreased each time the software releases the output mailbox by setting the RFOM1 bit.

### CAN interrupt enable register (CAN\_IER)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SLKIE	WKUIE
														rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRIE	Res.	Res.	Res.	LEC IE	BOF IE	EPV IE	EWG IE	Res.	FOV IE1	FF IE1	FMP IE1	FOV IE0	FF IE0	FMP IE0	TME IE
rw				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:18 Reserved, must be kept at reset value.

**Bit 17 SLKIE:** Sleep interrupt enable

0: No interrupt when SLAKI bit is set.

1: Interrupt generated when SLAKI bit is set.

**Bit 16 WKUIE:** Wake-up interrupt enable

0: No interrupt when WKUI is set.

1: Interrupt generated when WKUI bit is set.

**Bit 15 ERRIE:** Error interrupt enable

0: No interrupt is generated when an error condition is pending in the CAN\_ESR.

1: An interrupt is generation when an error condition is pending in the CAN\_ESR.

Bits 14:12 Reserved, must be kept at reset value.

- Bit 11 **LECIE**: Last error code interrupt enable  
 0: ERRI bit is not set when the error code in LEC[2:0] is set by hardware on error detection.  
 1: ERRI bit is set when the error code in LEC[2:0] is set by hardware on error detection.
- Bit 10 **BOFIE**: Bus-off interrupt enable  
 0: ERRI bit is not set when BOFF is set.  
 1: ERRI bit is set when BOFF is set.
- Bit 9 **EPVIE**: Error passive interrupt enable  
 0: ERRI bit is not set when EPVF is set.  
 1: ERRI bit is set when EPVF is set.
- Bit 8 **EWGIE**: Error warning interrupt enable  
 0: ERRI bit is not set when EWGF is set.  
 1: ERRI bit is set when EWGF is set.
- Bit 7 Reserved, must be kept at reset value.
- Bit 6 **FOVIE1**: FIFO overrun interrupt enable  
 0: No interrupt when FOVR is set.  
 1: Interrupt generation when FOVR is set.
- Bit 5 **FFIE1**: FIFO full interrupt enable  
 0: No interrupt when FULL bit is set.  
 1: Interrupt generated when FULL bit is set.
- Bit 4 **FMPIE1**: FIFO message pending interrupt enable  
 0: No interrupt generated when state of FMP[1:0] bits are not 00b.  
 1: Interrupt generated when state of FMP[1:0] bits are not 00b.
- Bit 3 **FOVIE0**: FIFO overrun interrupt enable  
 0: No interrupt when FOVR bit is set.  
 1: Interrupt generated when FOVR bit is set.
- Bit 2 **FFIE0**: FIFO full interrupt enable  
 0: No interrupt when FULL bit is set.  
 1: Interrupt generated when FULL bit is set.
- Bit 1 **FMPIE0**: FIFO message pending interrupt enable  
 0: No interrupt generated when state of FMP[1:0] bits are not 00b.  
 1: Interrupt generated when state of FMP[1:0] bits are not 00b.
- Bit 0 **TMEIE**: Transmit mailbox empty interrupt enable  
 0: No interrupt when RQCPx bit is set.  
 1: Interrupt generated when RQCPx bit is set.

*Note:* Refer to [Section 36.8: 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.	Res.	Res.	Res.	Res.	BOFF	EPVF	EWGF
									rw	rw	rw		r	r	r

Bits 31:24 **REC[7:0]**: Receive error counter

The implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

Bits 23:16 **TEC[7:0]**: Least significant byte of the 9-bit transmit error counter

The implementing part of the fault confinement mechanism of the CAN protocol.

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **LEC[2:0]**: Last error code

This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field is cleared to 0.

The LEC[2:0] bits can be set to value 0b111 by software. They are updated by hardware to indicate the current communication status.

000: No error

001: Stuff error

010: Form error

011: Acknowledgment error

100: Bit recessive error

101: Bit dominant error

110: CRC error

111: Set by software

Bit 3 Reserved, must be kept at reset value.

Bit 2 **BOFF**: Bus-off flag

This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255, refer to [Section 36.7.6: Error management](#).

Bit 1 **EPVF**: Error passive flag

This bit is set by hardware when the Error passive limit has been reached (Receive Error Counter or Transmit Error Counter > 127).

Bit 0 **EWGF**: Error warning flag

This bit is set by hardware when the warning limit has been reached (Receive Error Counter or Transmit Error Counter  $\geq$  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 36.7.7: Bit timing](#).

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:0 **BRP[9:0]**: Baud rate prescaler

These bits define the length of a time quanta.  
 $t_q = (BRP[9:0]+1) \times t_{PCLK}$

### 36.9.3 CAN mailbox registers

This chapter describes the registers of the transmit and receives mailboxes. Refer to [Section 36.7.5: Message storage](#) for detailed register mapping.

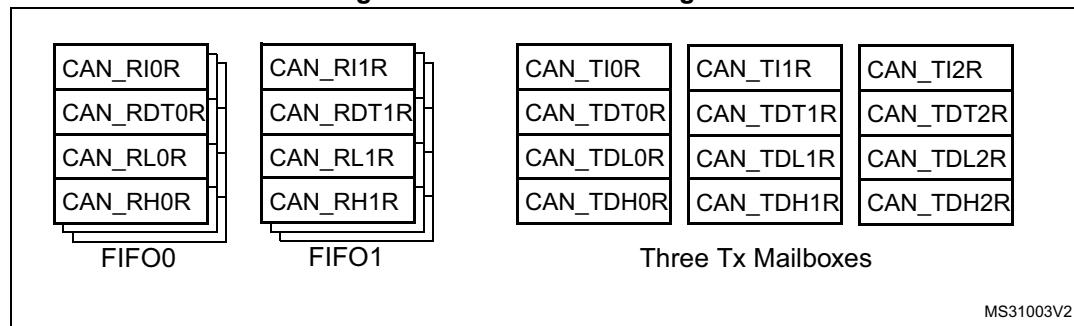
Transmit and receive mailboxes have the same registers except:

- The FMI field in the CAN\_RDTxR register.
- A receive mailbox is always write protected.
- A transmit mailbox is write-enabled only while empty, corresponding TME bit in the CAN\_TSR register set.

There are three TX mailboxes and two RX mailboxes. Each RX Mailbox allows access to a 3-level depth FIFO, the access being offered only to the oldest received message in the FIFO.

Each mailbox consist of four registers.

**Figure 453. 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: 0xXXXX XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TIME[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DLC[3:0]
													rw	rw	rw

Bits 31:16 **TIME[15:0]**: Message time stamp

This field contains the 16-bit timer value captured at the SOF transmission.

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **TGT**: Transmit global time

This bit is active only when the hardware is in the Time Trigger Communication mode, TTCM bit of the CAN\_MCR register is set.

0: Time stamp TIME[15:0] is not sent.

1: Time stamp TIME[15:0] value is sent in the last two data bytes of the 8-byte message: TIME[7:0] in data byte 7 and TIME[15:8] in data byte 6, replacing the data written in CAN\_TDHR[31:16] register (DATA6[7:0] and DATA7[7:0]). DLC must be programmed as 8 in order these two bytes to be sent over the CAN bus.

Bits 7:4 Reserved, must be kept at reset value.

Bits 3:0 **DLC[3:0]**: Data length code

This field defines the number of data bytes a data frame contains or a remote frame request. A message can contain from 0 to 8 data bytes, depending on the value in the DLC field.

### CAN mailbox data low register (CAN\_TDLxR) (x = 0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x188, 0x198, 0x1A8

Reset value: 0xXXXX XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Bits 31:24 **DATA3[7:0]**: Data byte 3  
     Data byte 3 of the message.
- Bits 23:16 **DATA2[7:0]**: Data byte 2  
     Data byte 2 of the message.
- Bits 15:8 **DATA1[7:0]**: Data byte 1  
     Data byte 1 of the message.
- Bits 7:0 **DATA0[7:0]**: Data byte 0  
     Data byte 0 of the message.  
     A message can contain from 0 to 8 data bytes and starts with byte 0.

### CAN mailbox data high register (CAN\_TDhxR) (x = 0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x18C, 0x19C, 0x1AC

Reset value: 0xFFFF XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Bits 31:24 **DATA7[7:0]**: Data byte 7  
     Data byte 7 of the message.  
*Note: If TGT of this message and TTCM are active, DATA7 and DATA6 are replaced by the TIME stamp value.*
- Bits 23:16 **DATA6[7:0]**: Data byte 6  
     Data byte 6 of the message.
- Bits 15:8 **DATA5[7:0]**: Data byte 5  
     Data byte 5 of the message.
- Bits 7:0 **DATA4[7:0]**: Data byte 4  
     Data byte 4 of the message.

### CAN receive FIFO mailbox identifier register (CAN\_RIxR) (x = 0..1)

Address offsets: 0x1B0, 0x1C0

Reset value: 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	Res	
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bits 31:21 **STID[10:0]/EXID[28:18]**: Standard identifier or extended identifier

The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

Bits 20:3 **EXID[17:0]**: Extended identifier

The LSBs of the extended identifier.

Bit 2 **IDE**: Identifier extension

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

Bit 1 **RTR**: Remote transmission request

0: Data frame

1: Remote frame

Bit 0 Reserved, must be kept at reset value.

### CAN receive FIFO mailbox data length control and time stamp register (CAN\_RDTxR) (x = 0..1)

Address offsets: 0x1B4, 0x1C4

Reset value: 0XXXXX XXXX

All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TIME[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FMI[7:0]								Res.	Res.	Res.	Res.	DLC[3:0]			
r	r	r	r	r	r	r	r					r	r	r	r

Bits 31:16 **TIME[15:0]**: Message time stamp

This field contains the 16-bit timer value captured at the SOF detection.

Bits 15:8 **FMI[7:0]**: Filter match index

This register contains the index of the filter the message stored in the mailbox passed through. For more details on identifier filtering, refer to [Section 36.7.4: Identifier filtering](#).

Bits 7:4 Reserved, must be kept at reset value.

Bits 3:0 **DLC[3:0]**: Data length code

This field defines the number of data bytes a data frame contains (0 to 8). It is 0 in the case of a remote frame request.

### CAN receive FIFO mailbox data low register (CAN\_RDLxR) (x = 0..1)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x1B8, 0x1C8

Reset value: 0XXXXX XXXXX

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: 0XXXXX XXXXX

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.

### 36.9.4 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.	Res.	Res.	Res.	Res.	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. When both CAN are used, they define the start bank of each CAN interface:

000001 = 1 filter assigned to CAN1 and 27 assigned to CAN2

011011 = 27 filters assigned to CAN1 and 1 filter assigned to CAN2

- to assign all filters to one CAN set CANSB value to zero and deactivate the non used CAN
- to use CAN1 only: stop the clock on CAN2 and/or set the CAN\_MCR.INRQ on CAN2
- to use CAN2 only: set the CAN\_MCR.INRQ on CAN1 or deactivate the interrupt register CAN\_IER on CAN1

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 446: Filter bank scale configuration - Register organization](#).

Bits 31: 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 446: Filter bank scale configuration - Register organization](#).

### CAN filter FIFO assignment register (CAN\_FFA1R)

Address offset: 0x214

Reset value: 0x0000 0000

This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN\_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	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 is stored in the specified FIFO.

0: Filter assigned to FIFO 0

1: Filter assigned to FIFO 1

### CAN filter activation register (CAN\_FA1R)

Address offset: 0x21C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	FACT 27	FACT 26	FACT 25	FACT 24	FACT 23	FACT 22	FACT 21	FACT 20	FACT 19	FACT 18	FACT 17	FACT 16
				rw											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FACT 15	FACT 14	FACT 13	FACT 12	FACT 11	FACT 10	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 36.7.4: Identifier filtering](#).

A Mask/Identifier register in **mask mode** has the same bit mapping as in **identifier list mode**.

For the register mapping/addresses of the filter banks refer to [Table 260](#).

### 36.9.5 bxCAN register map

Refer to [Section 2.2 on page 69](#) for the register boundary addresses. The registers from offset 0x200 to 0x31C are present only in CAN1.

**Table 260. 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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value	0	0	LOW[2:0]	0	0	0	0	1	TME[2:0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x004	CAN_MSR	Res.	Res.	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	
0x008	CAN_TSR	0	0	LOW[2:0]	0	0	0	0	1	TME[2:0]	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		
0x00C	CAN_RF0R	Res.	Res.	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	
0x010	CAN_RF1R	Res.	Res.	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	
0x014	CAN_IER	Res.	Res.	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	
0x018	CAN_ESR	REC[7:0]				TEC[7:0]				REC[7:0]				TEC[7:0]				REC[7:0]				TEC[7:0]				REC[7:0]			
	Reset value	0	0	SJW[1:0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x01C	CAN_BTR	Res.	Res.	Res.	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	SILM	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
0x020-0x17F	-	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
0x180	CAN_TI0R	STID[10:0]/EXID[28:18]								EXID[17:0]																			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	

Table 260. 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	<b>CAN_TDT0R</b>	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x				
0x188	<b>CAN_TDL0R</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					
0x18C	<b>CAN_TDHO R</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					
0x190	<b>CAN_TI1R</b>	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				
0x194	<b>CAN_TDT1R</b>	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x						
0x198	<b>CAN_TDL1R</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						
0x19C	<b>CAN_TDHO R</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						
0x1A0	<b>CAN_TI2R</b>	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	0					
0x1A4	<b>CAN_TDT2R</b>	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x						
0x1A8	<b>CAN_TDL2R</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						
0x1AC	<b>CAN_TDHO R</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						
0x1B0	<b>CAN_RI0R</b>	STID[10:0]/EXID[28:18]															EXID[17:0]													IDE	RTR	Req.	
	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	-					
0x1B4	<b>CAN_RDT0R</b>	TIME[15:0]															FMI[7:0]													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						

Table 260. 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
0x1B8	<b>CAN_RDL0R</b>	DATA3[7:0]							DATA2[7:0]							DATA1[7:0]							DATA0[7:0]										
		x	x	x	x	x	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]										
		x	x	x	x	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]																								
		x	x	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.				
0x1C4	<b>CAN_RDT1R</b>	TIME[15:0]							FMI[7:0]																								
		x	x	x	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]										
		x	x	x	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]										
		x	x	x	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	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
0x208	<b>CAN_FS1R</b>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	FSC[27:0]																	
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
0x20C	<b>CAN_FFS1R</b>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	FFS[27:0]																	
		0	0	0	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.																	
0x214	<b>CAN_FFA1R</b>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	FFA[27:0]																	
		0	0	0	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.																	
0x21C	<b>CAN_FA1R</b>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	FACT[27:0]																	
		0	0	0	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.																	
0x224-0x23F	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Res.																	

**Table 260. 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
0x240	CAN_F0R1																																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x244	CAN_F0R2																																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x248	CAN_F1R1																																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x24C	CAN_F1R2																																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
.	.																																
0x318	CAN_F27R1																																
	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																																
	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		

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 37 USB on-the-go full-speed/high-speed (OTG\_FS/OTG\_HS)

### 37.1 Introduction

Portions Copyright (c) Synopsys, Inc. All rights reserved. Used with permission.

This section presents the architecture and the programming model of the OTG\_FS/OTG\_HS controller.

The following acronyms are used throughout the section:

FS	Full-speed
LS	Low-speed
HS	High-speed
MAC	Media access controller
OTG	On-the-go
PFC	Packet FIFO controller
PHY	Physical layer
USB	Universal serial bus
UTMI	USB 2.0 Transceiver Macrocell interface (UTMI)
ULPI	UTMI+ Low Pin Interface
LPM	Link power management
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 261: OTG\\_HS speeds supported](#) below. OTG\_FS supports the speeds defined in the [Table 262: 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 261. 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 262. OTG\_FS speeds supported**

-	HS (480 Mb/s)	FS (12 Mb/s)	LS (1.5 Mb/s)
Host mode	-	X	X
Device mode	-	X	-

## 37.2 OTG\_FS/OTG\_HS main features

The main features can be divided into three categories: general, host-mode and device-mode features.

### 37.2.1 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\_FS supports the following PHY interface:
  - An on-chip full-speed PHY
- OTG\_HS supports the following PHY interfaces:
  - An on-chip full-speed PHY
  - A 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 2.0 specification
  - Integrated support for A-B device identification (ID line)
  - Integrated support for host Negotiation protocol (HNP) and session request protocol (SRP)
  - It allows host to turn  $V_{BUS}$  off to conserve battery power in OTG applications
  - It supports OTG monitoring of  $V_{BUS}$  levels with internal comparators
  - It supports dynamic host-peripheral switch of role
- It is software-configurable to operate as:
  - SRP capable USB FS/HS Peripheral (B-device)
  - SRP capable USB FS/HS/LS host (A-device)
  - USB On-The-Go Full-Speed Dual Role device
- It supports FS/HS SOF and LS Keep-alives with
  - SOF pulse PAD connectivity
  - SOF pulse internal connection to timer (TIMx)
  - Configurable framing period
  - Configurable end of frame interrupt
- OTG\_HS embeds an internal DMA with thresholding 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.

### 37.2.2 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.

### 37.2.3 Peripheral-mode features

The OTG\_FS/OTG\_HS interface main features in peripheral-mode are the following:

- 1 bidirectional control endpoint0
- 5[FS] / 8[HS] IN endpoints (EPs) configurable to support bulk, interrupt or isochronous transfers
- 5[FS] / 8[HS] OUT endpoints configurable to support bulk, interrupt or isochronous transfers
- Management of a shared Rx FIFO and a Tx-OUT FIFO for efficient usage of the USB data RAM
- Management of up to 6[FS] / 9[HS] dedicated Tx-IN FIFOs (one for each active IN EP) to put less load on the application
- Support for the soft disconnect feature.

### 37.3 OTG\_FS/OTG\_HS implementation

Table 263. OTG\_FS/OTG\_HS implementation<sup>(1)</sup>

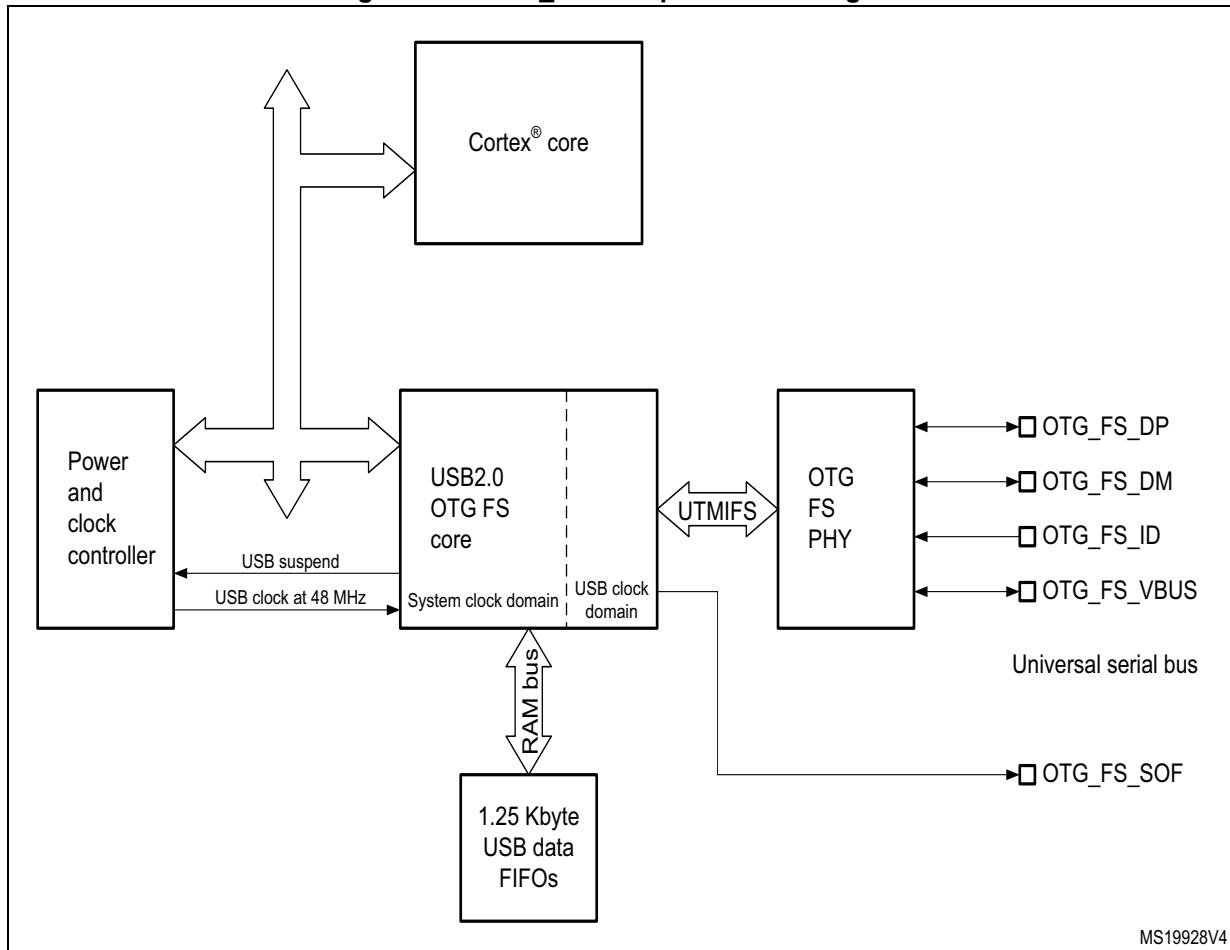
USB features	OTG_FS	OTG_HS
Device bidirectional endpoints (including EP0)	6	9
Host mode channels	12	16
Size of dedicated SRAM	1.2 KB	4 KB
USB 2.0 link power management (LPM) support		X
OTG revision supported		2.0
Attach detection protocol (ADP) support		-
Battery charging detection (BCD) support		-

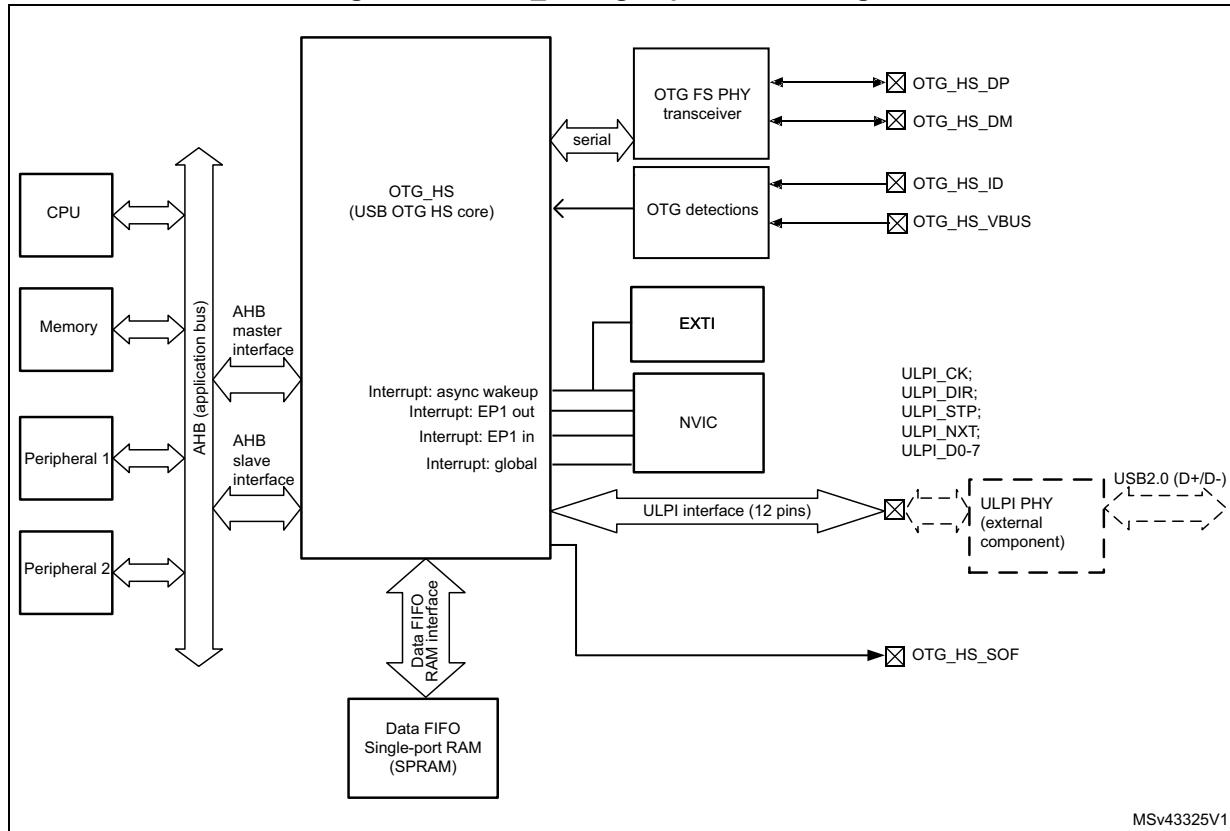
1. “X” = supported, “-” = not supported

## 37.4 OTG\_FS/OTG\_HS functional description

### 37.4.1 OTG\_FS/OTG\_HS block diagram

Figure 454. OTG\_FS full-speed block diagram



**Figure 455. OTG\_HS high-speed block diagram**

MSv43325V1

### 37.4.2 OTG\_FS/OTG\_HS pin and internal signals

**Table 264. OTG\_FS input/output pins**

Signal name	Signal type	Description
OTG_FS_DP	Digital input/output	USB OTG D+ line
OTG_FS_DM	Digital input/output	USB OTG D- line
OTG_FS_ID	Digital input	USB OTG ID
OTG_FS_VBUS	Analog input	USB OTG VBUS
OTG_FS_SOF	Digital output	USB OTG Start Of Frame (visibility)

**Table 265. OTG\_HS input/output pins**

Signal name	Signal type	Description
OTG_HS_DP	Digital input/output	USB OTG D+ line
OTG_HS_DM	Digital input/output	USB OTG D- line
OTG_HS_ID	Digital input	USB OTG ID
OTG_HS_VBUS	Analog input	USB OTG VBUS
OTG_HS_SOF	Digital output	USB OTG Start Of Frame (visibility)

**Table 265. OTG\_HS input/output pins (continued)**

Signal name	Signal type	Description
OTG_HS_ULPI_CK	Digital input	USB OTG ULPI clock
OTG_HS_ULPI_DIR	Digital input	USB OTG ULPI data bus direction control
OTG_HS_ULPI_STP	Digital output	USB OTG ULPI data stream stop
OTG_HS_ULPI_NXT	Digital input	USB OTG ULPI next data stream request
OTG_HS_ULPI_D[0..7]	Digital input/output	USB OTG ULPI 8-bit bi-directional data bus

**Table 266. OTG\_FS/OTG\_HS input/output signals**

Signal name	Signal type	Description
usb_sof	Digital output	USB OTG start-of-frame event for on chip peripherals
usb_wkup	Digital output	USB OTG wake-up event output
usb_gbl_it	Digital output	USB OTG global interrupt
usb_ep1_in_it	Digital output	USB OTG endpoint 1 in interrupt
usb_ep1_out_it	Digital output	USB OTG endpoint 1 out interrupt

### 37.4.3 OTG\_FS/OTG\_HS core

The USB OTG\_FS/OTG\_HS receives the 48 MHz clock from the reset and clock controller (RCC). This 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 37.13: OTG\\_FS/OTG\\_HS interrupts](#).

The CPU submits data over the USB by writing 32-bit words to dedicated OTG locations (push registers). The data are then automatically stored into Tx-data FIFOs configured within the USB data RAM. There is one Tx FIFO push register for each in-endpoint (peripheral mode) or out-channel (host mode).

The CPU receives the data from the USB by reading 32-bit words from dedicated OTG addresses (pop registers). The data are then automatically retrieved from a shared Rx FIFO configured within the 1.25[FS] / 4[HS]-Kbyte USB data RAM. There is one Rx FIFO pop register for each out-endpoint or in-channel.

The USB protocol layer is driven by the serial interface engine (SIE) and serialized over the USB by the transceiver module within the on-chip physical layer (PHY) or external HS PHY.

**Caution:** To guarantee a correct operation for the USB OTG FS peripheral, the AHB frequency must be higher than 14.2 MHz.

**Caution:** To guarantee a correct operation for the USB OTG\_HS peripheral, the AHB frequency should be higher than 30 MHz.

### 37.4.4 Embedded full-speed OTG PHY connected to OTG\_FS

The embedded full-speed OTG PHY is controlled by the OTG FS core and conveys USB control & data signals through the full-speed subset of the UTMI+ Bus (UTMIFS). It provides

the physical support to USB connectivity.

The full-speed OTG PHY includes the following components:

- FS/LS transceiver module used by both host and device. It directly drives transmission and reception on the single-ended USB lines.
- 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.

#### **37.4.5 Embedded full-speed OTG PHY connected to OTG\_HS**

The embedded full-speed OTG PHY is controlled by the OTG\_HS core and conveys USB control & data signals through the full-speed subset of the UTMI+ Bus (UTMIFS). It provides the physical support to USB connectivity.

The full-speed OTG PHY includes the following components:

- FS/LS transceiver module used by both host and device. It directly drives transmission and reception on the single-ended USB lines.
- 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.

#### **37.4.6 OTG detections**

Additionally the OTG\_FS/OTG\_HS uses the following functions:

- integrated ID pull-up resistor used to sample the ID line for A/B device identification.
- $V_{BUS}$  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.

#### **37.4.7 High-speed OTG PHY connected to OTG\_HS**

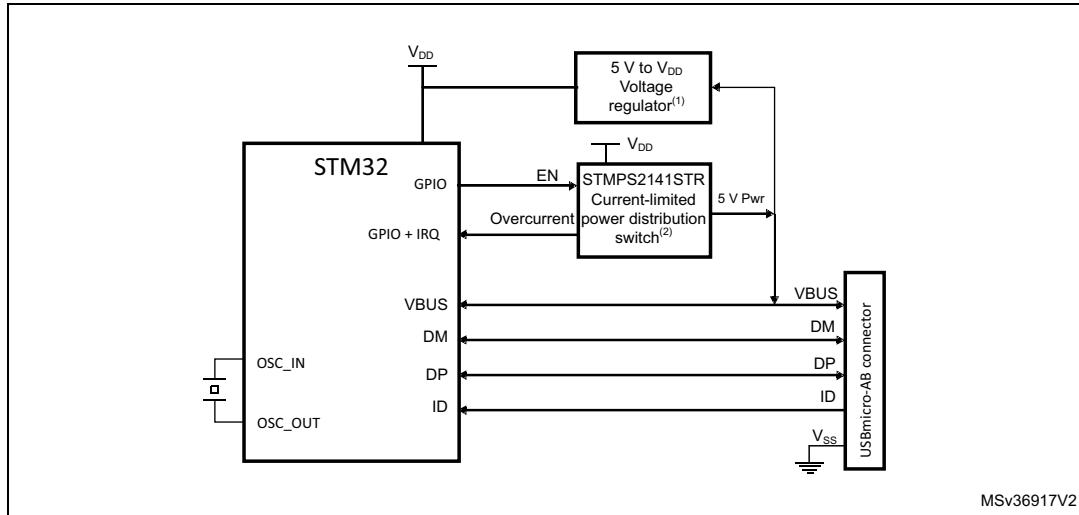
Note: Refer to implementation table to determine if an HS PHY is embedded.

The USB OTG\_HS core includes an ULPI interface to connect an external HS PHY.

Note: In case of multiple OTG\_HS instances, ULPI may not be available on each one. Refer to implementation table.

## 37.5 OTG\_FS/OTG\_HS dual role device (DRD)

Figure 456. OTG\_FS/OTG\_HS A-B device connection



MSv36917V2

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.

### 37.5.1 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.

### 37.5.2 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 37.16: OTG\\_FS/OTG\\_HS programming model](#).

### 37.5.3 SRP dual role device

The SRP capable bit in the global USB configuration register (SRPCAP bit in OTG\_GUSBCFG) enables the OTG\_FS/OTG\_HS core to switch off the generation of  $V_{BUS}$  for the A-device to save power. Note that the A-device is always in charge of driving  $V_{BUS}$  regardless of the host or peripheral role of the OTG\_FS/OTG\_HS.

The SRP A/B-device program model is described in detail in [Section 37.16: OTG\\_FS/OTG\\_HS programming model](#).

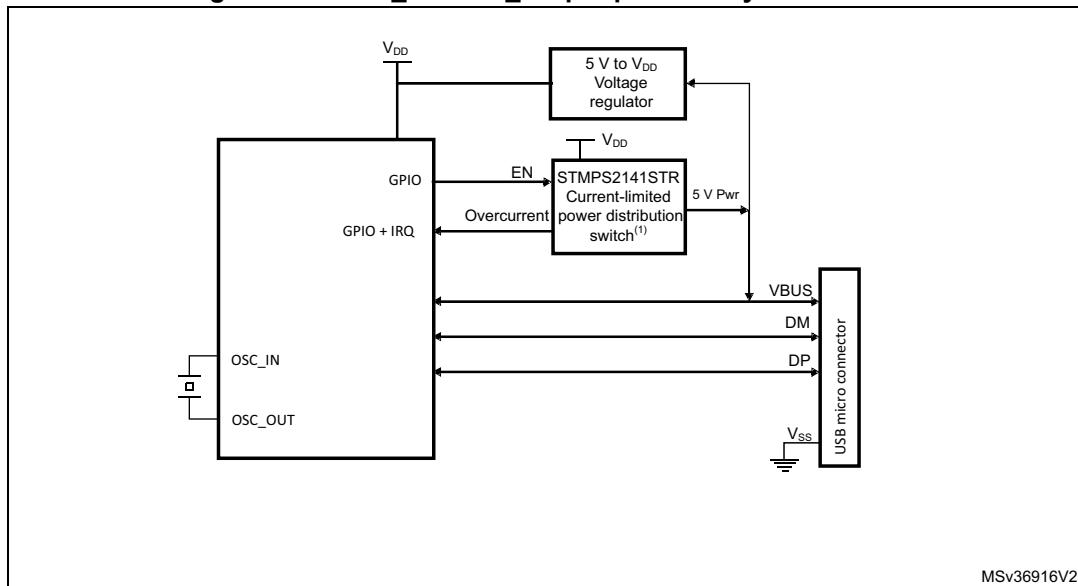
## 37.6 OTG\_FS/OTG\_HS as a USB peripheral

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 457: OTG\\_FS/OTG\\_HS peripheral-only connection](#))
  - The force device mode bit (FDMOD) in the [Section 37.15.4: OTG USB configuration register \(OTG\\_GUSBCFG\)](#) is set to 1, forcing the OTG\_FS/OTG\_HS core to work as an USB peripheral-only. In this case, the ID line is ignored even if it is present on the USB connector.

**Note:** *To build a bus-powered device implementation in case of the B-device or peripheral-only configuration, an external regulator has to be added, that generates the necessary power-supply from  $V_{BUS}$ .*

Figure 457. OTG\_FS/OTG\_HS peripheral-only connection



1. Use a regulator to build a bus-powered device.

### 37.6.1 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.

### 37.6.2 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 wake-up 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.

## 37.6.3 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\_DOEPMASK) 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

### 37.7

## OTG\_FS/OTG\_HS as a 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:

### Automatic host mode direct from ID pin:

The ID pin is not always available, refer to product datasheet for available pins.

- Use cases:
  - Micro connector: A-Device with a Micro-A plug inserted, and connector ID pin connected to the STM32 ID pin which will then be detected in a low state.
- Note that the integrated pulldown resistors are automatically set on the DP/DM lines.

### Manual forcing of host mode when not possible via ID pin:

The force host mode bit (FHMOD) in the [OTG USB configuration register \(OTG\\_GUSBCFG\)](#) forces the OTG\_FS/OTG\_HS core to work as a USB host-only when required.

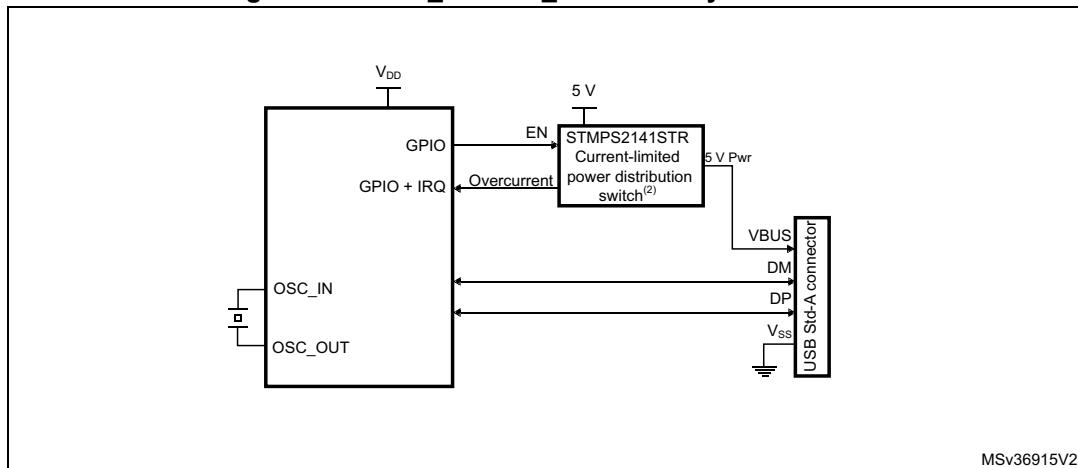
- Use cases:
  - Standard-A connector which always implies Host mode.
  - Micro connector, so ID present on connector, but ID pin not available in pinout, at A-plug insertion (detected by another means, for example: GPIO).
  - A Type-C connector, when host functionality is determined at cable attachment or via power delivery messages.
- Note that the integrated pulldown 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 458. OTG\_FS/OTG\_HS host-only connection**



1.  $V_{DD}$  range is between 2 V and 3.6 V.

### 37.7.1 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.

### 37.7.2 USB host states

#### Host port power

On-chip 5 V  $V_{BUS}$  generation is not supported. For this reason, a charge pump or, if 5 V are available on the application board, a basic power switch, must be added externally to drive the 5 V  $V_{BUS}$  line. The external charge pump can be driven by any GPIO output or via an I<sup>2</sup>C interface connected to an external PMIC (power management IC). When the application decides to power on  $V_{BUS}$ , it must also set the port power bit in the host port control and status register (PPWR bit in OTG\_HPRT).

#### $V_{BUS}$ valid

When HNP or SRP is enabled the  $V_{BUS}$  sensing pin must be connected to  $V_{BUS}$ . The  $V_{BUS}$  input ensures that valid  $V_{BUS}$  levels are supplied by the charge pump during USB operations. Any unforeseen  $V_{BUS}$  voltage drop below the  $V_{BUS}$  valid threshold (4.4 V) leads to an OTG interrupt triggered by the session end detected bit (SEDET bit in OTG\_GOTGINT). The application is then required to remove the  $V_{BUS}$  power and clear the port power bit.

When HNP and SRP are both disabled, the  $V_{BUS}$  sensing pin does not need to be connected to  $V_{BUS}$ .

The charge pump overcurrent flag can also be used to prevent electrical damage. Connect the overcurrent flag output from the charge pump to any GPIO input and configure it to

generate a port interrupt on the active level. The overcurrent ISR must promptly disable the  $V_{BUS}$  generation and clear the port power bit.

### Host detection of a peripheral connection

If SRP or HNP are enabled, even if USB peripherals or B-devices can be attached at any time, the OTG\_FS/OTG\_HS does 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 wake-up). In this case the remote wake-up interrupt (WKUPINT bit in OTG\_GINTSTS) is generated upon detection of a remote wake-up 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.

### 37.7.3 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

### 37.7.4 Host scheduler

The host core features a built-in hardware scheduler which is able to autonomously re-order and manage the USB transaction requests posted by the application. At the beginning of each frame the host executes the periodic (isochronous and interrupt) transactions first, followed by the nonperiodic (control and bulk) transactions to achieve the higher level of priority granted to the isochronous and interrupt transfer types by the USB specification.

The host processes the USB transactions through request queues (one for periodic and one for nonperiodic). Each request queue can hold up to 8 entries. Each entry represents a pending transaction request from the application, and holds the IN or OUT channel number along with other information to perform a transaction on the USB. The order in which the requests are written to the queue determines the sequence of the transactions on the USB interface.

At the beginning of each frame, the host processes the periodic request queue first, followed by the nonperiodic request queue. The host issues an incomplete periodic transfer interrupt (IPXFR bit in OTG\_GINTSTS) if an isochronous or interrupt transaction scheduled for the current frame is still pending at the end of the current frame. The OTG\_FS/OTG\_HS core is fully responsible for the management of the periodic and nonperiodic request queues. The periodic transmit FIFO and queue status register (OTG\_HPTXSTS) and nonperiodic transmit FIFO and queue status register (OTG\_HNPTXSTS) are read-only registers which can be used by the application to read the status of each request queue. They contain:

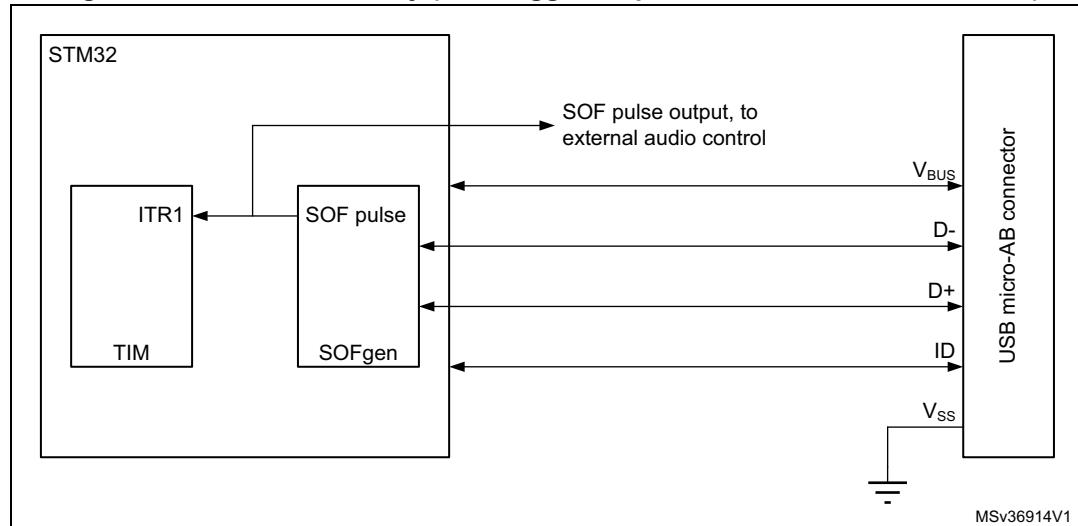
- The number of free entries currently available in the periodic (nonperiodic) request queue (8 max)
- Free space currently available in the periodic (nonperiodic) Tx FIFO (out-transactions)
- IN/OUT token, host channel number and other status information.

As request queues can hold a maximum of 8 entries each, the application can push to schedule host transactions in advance with respect to the moment they physically reach the SB for a maximum of 8 pending periodic transactions plus 8 pending non-periodic transactions.

To post a transaction request to the host scheduler (queue) the application must check that there is at least 1 entry available in the periodic (nonperiodic) request queue by reading the PTXQSAV bits in the OTG\_HNPTXSTS register or NPTQXSAV bits in the OTG\_HNPTXSTS register.

## 37.8 OTG\_FS/OTG\_HS SOF trigger

**Figure 459. 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.

### 37.8.1 Host SOFs

In host mode the number of PHY clocks occurring between the generation of two consecutive SOF (HS/FS) or Keep-alive (LS) tokens is programmable in the host frame interval register (HFIR), thus providing application control over the SOF framing period. An interrupt is generated at any start of frame (SOF bit in OTG\_GINTSTS). The current frame number and the time remaining until the next SOF are tracked in the host frame number register (HFNUM).

A SOF pulse signal, is generated at any SOF starting token and with a width of 20 HCLK cycles. The SOF pulse is also internally connected to the input trigger of the timer, so that the input capture feature, the output compare feature and the timer can be triggered by the SOF pulse.

### 37.8.2 Peripheral SOFs

In device mode, the start of frame interrupt is generated each time an SOF token is received on the USB (SOF bit in OTG\_GINTSTS). The corresponding frame number can be read from the device status register (FNSOF bit in OTG\_DSTS). A SOF pulse signal with a width of 20 HCLK cycles is also generated. The SOF pulse signal is also internally connected to

the TIM input trigger, so that the input capture feature, the output compare feature and the timer can be triggered by the SOF pulse.

The end of periodic frame interrupt (OTG\_GINTSTS/EOPF) is used to notify the application when 80%, 85%, 90% or 95% of the time frame interval elapsed depending on the periodic frame interval field in the device configuration register (PFIVL bit in OTG\_DCFG). This feature can be used to determine if all of the isochronous traffic for that frame is complete.

## 37.9 OTG\_FS/OTG\_HS low-power modes

*Table 267* below defines the STM32 low power modes and their compatibility with the OTG.

**Table 267. Compatibility of STM32 low power modes with the OTG**

Mode	Description	USB compatibility
Run	MCU fully active	Required when USB not in suspend state.
Sleep	USB suspend exit causes the device to exit Sleep mode. Peripheral registers content is kept.	Available while USB is in suspend state.
Stop	USB suspend exit causes the device to exit Stop mode. Peripheral registers content is kept <sup>(1)</sup> .	Available while USB is in suspend state.
Standby	Powered-down. The peripheral must be reinitialized after exiting Standby mode.	Not compatible with USB applications.

1. Within Stop mode there are different possible settings. Some restrictions may also exist, refer to *Section 4: Power controller (PWR)* to understand which (if any) restrictions apply when using OTG.

The following bits and procedures reduce power consumption.

The power consumption of the OTG PHY is controlled by two or three bits in the general core configuration register, depending on OTG revision supported.

- **PHY power down (OTG\_GCCFG/PWRDWN)**  
It switches on/off the full-speed transceiver module of the PHY. It must be preliminarily set to allow any USB operation
- **V<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 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 wake-up 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 wake-up (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.

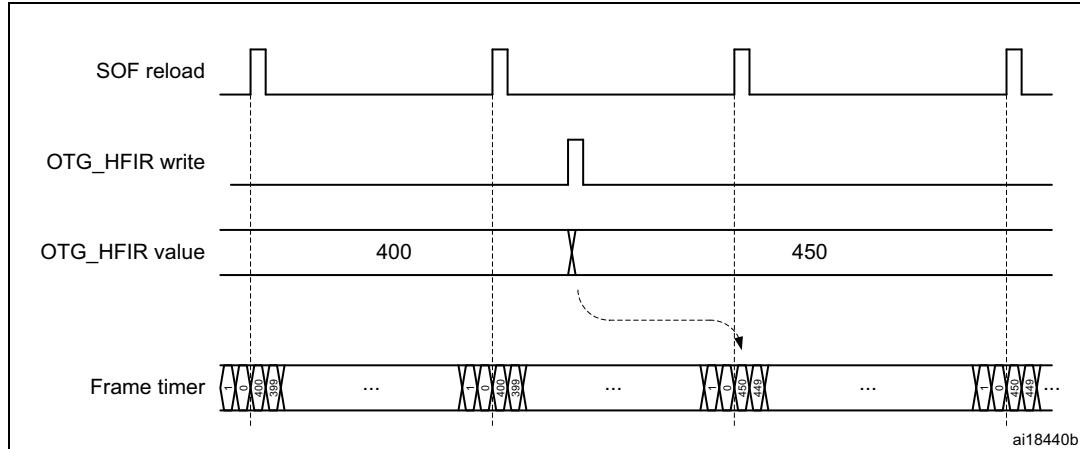
## 37.10 OTG\_FS/OTG\_HS Dynamic update of the OTG\_HFIR register

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 460](#).

For a dynamic update, it is required to set RLDCTRL=1.

**Figure 460. Updating OTG\_HFIR dynamically (RLDCTRL = 1)**



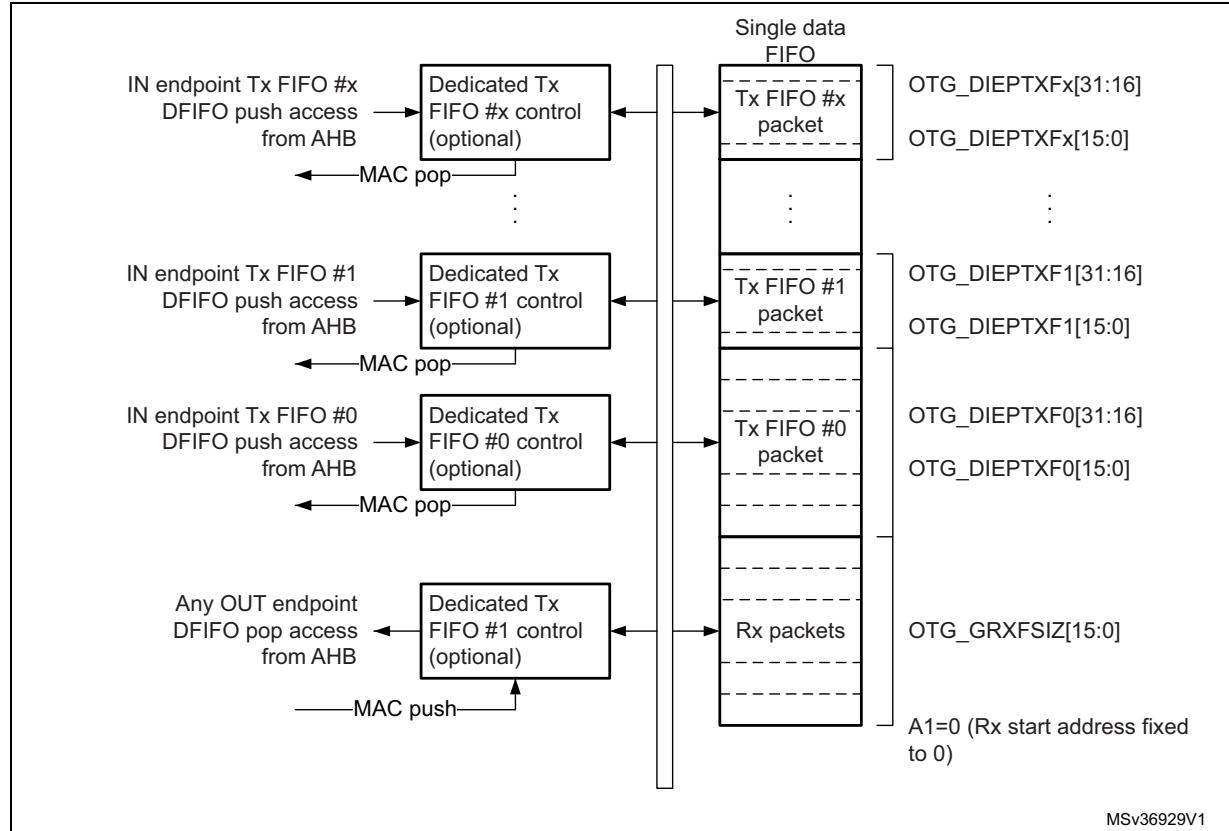
## 37.11 OTG\_FS/OTG\_HS data FIFOs

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.

### 37.11.1 Peripheral FIFO architecture

**Figure 461. Device-mode FIFO address mapping and AHB FIFO access mapping**



#### Peripheral Rx FIFO

The OTG peripheral uses a single receive FIFO that receives the data directed to all OUT endpoints. Received packets are stacked back-to-back until free space is available in the Rx FIFO. The status of the received packet (which contains the OUT endpoint destination number, the byte count, the data PID and the validity of the received data) is also stored by the core on top of the data payload. When no more space is available, host transactions are NACKed and an interrupt is received on the addressed endpoint. The size of the receive FIFO is configured in the receive FIFO size register (OTG\_GRXFSIZ).

The single receive FIFO architecture makes it more efficient for the USB peripheral to fill in the receive RAM buffer:

- All OUT endpoints share the same RAM buffer (shared FIFO)
- The OTG\_FS/OTG\_HS core can fill in the receive FIFO up to the limit for any host sequence of OUT tokens

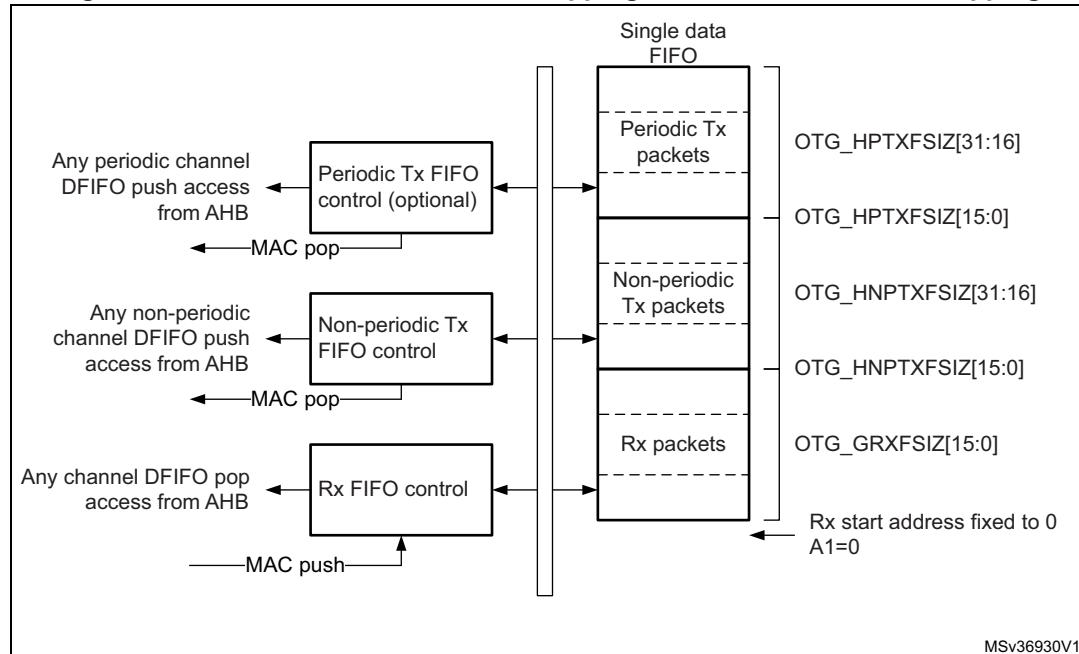
The application keeps receiving the Rx FIFO non-empty interrupt (RXFLVL bit in OTG\_GINTSTS) as long as there is at least one packet available for download. It reads the packet information from the receive status read and pop register (OTG\_GRXSTSP) and finally pops data off the receive FIFO by reading from the endpoint-related pop address.

### Peripheral Tx FIFOs

The core has a dedicated FIFO for each IN endpoint. The application configures FIFO sizes by writing the endpoint 0 transmit FIFO size register (OTG\_DIEPTXF0) for IN endpoint0 and the device IN endpoint transmit FIFOx registers (OTG\_DIEPTXF $x$ ) for IN endpoint- $x$ .

#### 37.11.2 Host FIFO architecture

**Figure 462. Host-mode FIFO address mapping and AHB FIFO access mapping**



MSv36930V1

### 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.

### 37.11.3 FIFO RAM allocation

#### Device mode

**Receive FIFO RAM allocation:** the application must allocate RAM for SETUP packets:

- 10 locations must be reserved in the receive FIFO to receive SETUP packets on control endpoint. The core does not use these locations, which are reserved for SETUP packets, to write any other data.
- One location is to be allocated for Global OUT NAK.
- Status information is written to the FIFO along with each received packet. Therefore, a minimum space of (largest packet size / 4) + 1 must be allocated to receive packets. If multiple isochronous endpoints are enabled, then at least two (largest packet size / 4) + 1 spaces must be allocated to receive back-to-back packets. Typically, two (largest packet size / 4) + 1 spaces are recommended so that when the previous packet is being transferred to the CPU, the USB can receive the subsequent packet.
- Along with the last packet for each endpoint, transfer complete status information is also pushed to the FIFO. One location for each OUT endpoint is recommended.

Device RxFIFO =

$(5 * \text{number of control endpoints} + 8) + ((\text{largest USB packet used} / 4) + 1 \text{ for status information}) + (2 * \text{number of OUT endpoints}) + 1 \text{ for Global NAK}$

Example: The MPS is 1,024 bytes for a periodic USB packet and 512 bytes for a non-periodic USB packet. There are three OUT endpoints, three IN endpoints, one control endpoint, and three host channels.

Device RxFIFO =  $(5 * 1 + 8) + ((1,024 / 4) + 1) + (2 * 4) + 1 = 279$

**Transmit FIFO RAM allocation:** the minimum RAM space required for each IN endpoint Transmit FIFO is the maximum packet size for that particular IN endpoint.

*Note:* More space allocated in the transmit IN endpoint FIFO results in better performance on the USB.

### Host mode

Receive FIFO RAM allocation:

Status information is written to the FIFO along with each received packet. Therefore, a minimum space of  $(\text{largest packet size} / 4) + 1$  must be allocated to receive packets. If multiple isochronous channels are enabled, then at least two  $(\text{largest packet size} / 4) + 1$  spaces must be allocated to receive back-to-back packets. Typically, two  $(\text{largest packet size} / 4) + 1$  spaces are recommended so that when the previous packet is being transferred to the CPU, the USB can receive the subsequent packet.

Along with the last packet in the host channel, transfer complete status information is also pushed to the FIFO. So one location must be allocated for this.

Host RxFIFO =  $(\text{largest USB packet used} / 4) + 1 \text{ for status information} + 1 \text{ transfer complete}$

Example: Host RxFIFO =  $((1,024 / 4) + 1) + 1 = 258$

Transmit FIFO RAM allocation:

The minimum amount of RAM required for the host Non-periodic Transmit FIFO is the largest maximum packet size among all supported non-periodic OUT channels.

Typically, two largest packet sizes worth of space is recommended, so that when the current packet is under transfer to the USB, the CPU can get the next packet.

Non-Periodic TxFIFO = largest non-periodic USB packet used / 4

Example: Non-Periodic TxFIFO =  $(512 / 4) = 128$

The minimum amount of RAM required for host periodic Transmit FIFO is the largest maximum packet size out of all the supported periodic OUT channels. If there is at least one isochronous OUT endpoint, then the space must be at least two times the maximum packet size of that channel.

Host Periodic TxFIFO = largest periodic USB packet used / 4

Example: Host Periodic TxFIFO =  $(1,024 / 4) = 256$

*Note:* More space allocated in the Transmit Non-periodic FIFO results in better performance on the USB.

## 37.12 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

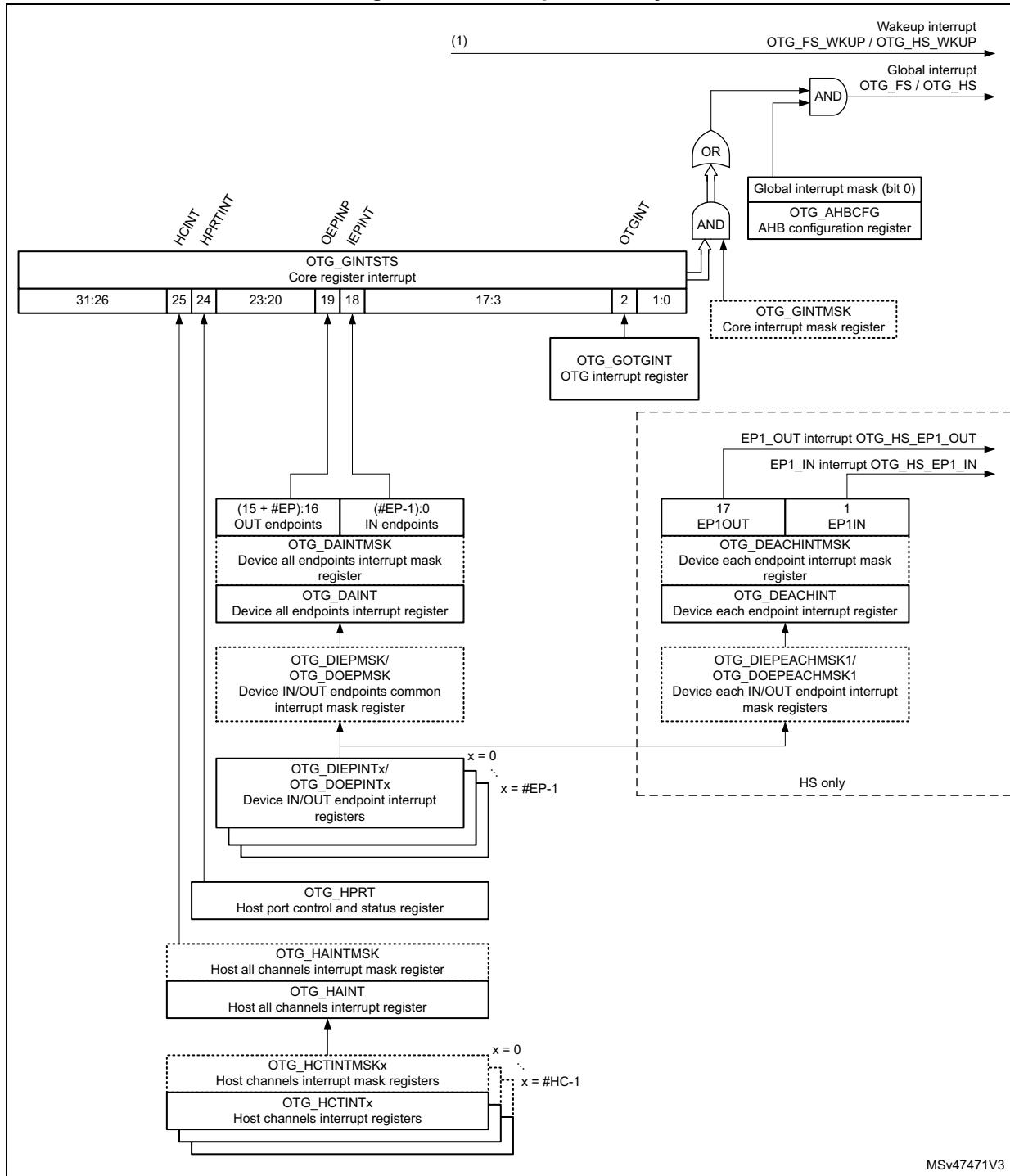
As the OTG\_FS core is able to fill in the 1.25-Kbyte RAM buffer very efficiently, and as 1.25-Kbyte of transmit/receive data is more than enough to cover a full speed frame, the USB system is able to withstand the maximum full-speed data rate for up to one USB frame (1 ms) without any CPU intervention.

## 37.13 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 463* shows the interrupt hierarchy.

Figure 463. Interrupt hierarchy



1. OTG\_FS\_WKUP / OTG\_HS\_WKUP become active (high state) when resume condition occurs during L1 SLEEP or L2 SUSPEND states.

MSv47471V3

## 37.14 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.

### 37.14.1 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 268. Core global control and status registers (CSRs)**

Acronym	Address offset	Register name
OTG_GOTGCTL	0x000	<a href="#">Section 37.15.1: OTG control and status register (OTG_GOTGCTL)</a>
OTG_GOTGINT	0x004	<a href="#">Section 37.15.2: OTG interrupt register (OTG_GOTGINT)</a>
OTG_GAHBCFG	0x008	<a href="#">Section 37.15.3: OTG AHB configuration register (OTG_GAHBCFG)</a>
OTG_GUSBCFG	0x00C	<a href="#">Section 37.15.4: OTG USB configuration register (OTG_GUSBCFG)</a>
OTG_GRSTCTL	0x010	<a href="#">Section 37.15.5: OTG reset register (OTG_GRSTCTL)</a>
OTG_GINTSTS	0x014	<a href="#">Section 37.15.6: OTG core interrupt register (OTG_GINTSTS)</a>
OTG_GINTMSK	0x018	<a href="#">Section 37.15.7: OTG interrupt mask register (OTG_GINTMSK)</a>

**Table 268. Core global control and status registers (CSRs) (continued)**

Acronym	Address offset	Register name
OTG_GRXSTSR	0x01C	<a href="#">Section 37.15.8: OTG receive status debug read register (OTG_GRXSTSR)</a>
		<a href="#">Section 37.15.9: OTG receive status debug read [alternate] (OTG_GRXSTSR)</a>
OTG_GRXSTSP	0x020	<a href="#">Section 37.15.10: OTG status read and pop registers (OTG_GRXSTSP)</a>
		<a href="#">Section 37.15.11: OTG status read and pop registers [alternate] (OTG_GRXSTSP)</a>
OTG_GRXFSIZ	0x024	<a href="#">Section 37.15.12: OTG receive FIFO size register (OTG_GRXFSIZ)</a>
OTG_HNPTXFSIZ/ OTG_DIEPTXF0 <sup>(1)</sup>	0x028	<a href="#">Section 37.15.13: OTG host non-periodic transmit FIFO size register (OTG_HNPTXFSIZ)/Endpoint 0 Transmit FIFO size (OTG_DIEPTXF0)</a>
OTG_HNPTXSTS	0x02C	<a href="#">Section 37.15.14: OTG non-periodic transmit FIFO/queue status register (OTG_HNPTXSTS)</a>
OTG_GCCFG	0x038	<a href="#">Section 37.15.15: OTG general core configuration register (OTG_GCCFG)</a>
OTG_CID	0x03C	<a href="#">Section 37.15.16: OTG core ID register (OTG_CID)</a>
OTG_GLPMCFG	0x54	<a href="#">Section 37.15.17: OTG core LPM configuration register (OTG_GLPMCFG)</a>
OTG_HPTXFSIZ	0x100	<a href="#">Section 37.15.18: OTG host periodic transmit FIFO size register (OTG_HPTXFSIZ)</a>
OTG_DIEPTXFx	0x104 0x108 ... 0x114	<a href="#">Section 37.15.19: OTG device IN endpoint transmit FIFO x size register (OTG_DIEPTXFx) for USB_OTG FS</a>
OTG_DIEPTXFx	0x104 0x108 ... 0x120	<a href="#">Section 37.15.19: OTG device IN endpoint transmit FIFO x size register (OTG_DIEPTXFx) for USB_OTG HS</a>

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 269. Host-mode control and status registers (CSRs)**

Acronym	Offset address	Register name
OTG_HCFG	0x400	<a href="#">Section 37.15.21: OTG host configuration register (OTG_HCFG)</a>
OTG_HFIR	0x404	<a href="#">Section 37.15.22: OTG host frame interval register (OTG_HFIR)</a>
OTG_HFNUM	0x408	<a href="#">Section 37.15.23: OTG host frame number/frame time remaining register (OTG_HFNUM)</a>

**Table 269. Host-mode control and status registers (CSRs) (continued)**

Acronym	Offset address	Register name
OTG_HPTXSTS	0x410	<a href="#">Section 37.15.24: OTG_Host periodic transmit FIFO/queue status register (OTG_HPTXSTS)</a>
OTG_HAINT	0x414	<a href="#">Section 37.15.25: OTG host all channels interrupt register (OTG_HAINT)</a>
OTG_HAINTMSK	0x418	<a href="#">Section 37.15.26: OTG host all channels interrupt mask register (OTG_HAINTMSK)</a>
OTG_HPRT	0x440	<a href="#">Section 37.15.27: OTG host port control and status register (OTG_HPRT)</a>
OTG_HCCHARx	0x500 0x520 ... 0x660	<a href="#">Section 37.15.28: OTG host channel x characteristics register (OTG_HCCHARx) for USB_OTG FS</a>
OTG_HCCHARx	0x500 0x520 ... 0x6E0	<a href="#">Section 37.15.28: OTG host channel x characteristics register (OTG_HCCHARx) for USB_OTG HS</a>
OTG_HCSPLTx	0x504 0x524 ... 0x6E4	<a href="#">Section 37.15.29: OTG host channel x split control register (OTG_HCSPLTx)</a>
OTG_HCINTx	0x508 0x528 ... 0x668	<a href="#">Section 37.15.30: OTG host channel x interrupt register (OTG_HCINTx) for USB_OTG FS</a>
OTG_HCINTx	0x508 0x528 ... 0x6E8	<a href="#">Section 37.15.30: OTG host channel x interrupt register (OTG_HCINTx) for USB_OTG HS</a>
OTG_HCINTMSKx	0x50C 0x52C ... 0x66C	<a href="#">Section 37.15.31: OTG host channel x interrupt mask register (OTG_HCINTMSKx) for USB_OTG FS</a>
OTG_HCINTMSKx	0x50C 0x52C ... 0x6EC	<a href="#">Section 37.15.31: OTG host channel x interrupt mask register (OTG_HCINTMSKx) for USB_OTG HS</a>
OTG_HCTSIZx	0x510 0x530 ... 0x670	<a href="#">Section 37.15.32: OTG host channel x transfer size register (OTG_HCTSIZx) for USB_OTG FS</a>

**Table 269. Host-mode control and status registers (CSRs) (continued)**

Acronym	Offset address	Register name
OTG_HCTSIZx	0x510 0x530 .... 0x6F0	<a href="#">Section 37.15.32: OTG host channel x transfer size register (OTG_HCTSIZx) for USB_OTG HS</a>
OTG_HCDMAX	0x514 0x534 .... 0x6F4	<a href="#">Section 37.15.33: OTG host channel x DMA address register (OTG_HCDMAX)</a>

**Device-mode CSR map**

These registers must be programmed every time the core changes to device mode.

**Table 270. Device-mode control and status registers**

Acronym	Offset address	Register name
OTG_DCFG	0x800	<a href="#">Section 37.15.35: OTG device configuration register (OTG_DCFG)</a>
OTG_DCTL	0x804	<a href="#">Section 37.15.36: OTG device control register (OTG_DCTL)</a>
OTG_DSTS	0x808	<a href="#">Section 37.15.37: OTG device status register (OTG_DSTS)</a>
OTG_DIEPMSK	0x810	<a href="#">Section 37.15.38: OTG device IN endpoint common interrupt mask register (OTG_DIEPMSK)</a>
OTG_DOEPMSK	0x814	<a href="#">Section 37.15.39: OTG device OUT endpoint common interrupt mask register (OTG_DOEPMSK)</a>
OTG_DAINT	0x818	<a href="#">Section 37.15.40: OTG device all endpoints interrupt register (OTG_DAINT)</a>
OTG_DAINTMSK	0x81C	<a href="#">Section 37.15.41: OTG all endpoints interrupt mask register (OTG_DAINTMSK)</a>
OTG_DVBUSDIS	0x828	<a href="#">Section 37.15.42: OTG device V<sub>BUS</sub> discharge time register (OTG_DVBUSDIS)</a>
OTG_DVBUSPULSE	0x82C	<a href="#">Section 37.15.43: OTG device V<sub>BUS</sub> pulsing time register (OTG_DVBUSPULSE)</a>
OTG_DTHRCTL	0x830	<a href="#">Section 37.15.44: OTG device threshold control register (OTG_DTHRCTL)</a>
OTG_DIEPEMPMSK	0x834	<a href="#">Section 37.15.45: OTG device IN endpoint FIFO empty interrupt mask register (OTG_DIEPEMPMSK)</a>
OTG_DEACHINT	0x838	<a href="#">Section 37.15.46: OTG device each endpoint interrupt register (OTG_DEACHINT)</a>
OTG_DEACHINTMSK	0x83C	<a href="#">Section 37.15.47: OTG device each endpoint interrupt mask register (OTG_DEACHINTMSK)</a>

**Table 270. Device-mode control and status registers (continued)**

Acronym	Offset address	Register name
OTG_HS_DIEPEACHM SK1	0x844	<a href="#">Section 37.15.48: OTG device each IN endpoint-1 interrupt mask register (OTG_HS_DIEPEACHMSK1)</a>
OTG_HS_DOEPEACHM SK1	0x884	<a href="#">Section 37.15.49: OTG device each OUT endpoint-1 interrupt mask register (OTG_HS_DOEPEACHMSK1)</a>
OTG_DIEPCTL0	0x900	<a href="#">Section 37.15.50: OTG device control IN endpoint 0 control register (OTG_DIEPCTL0) for USB_OTG FS</a>
OTG_DIEPCTLx	0x920 0x940 ... 0x9A0	<a href="#">Section 37.15.51: OTG device IN endpoint x control register (OTG_DIEPCTLx) for USB_OTG FS</a>
OTG_DIEPCTLx	0x900 0x920 ... 0xA00	<a href="#">Section 37.15.51: OTG device IN endpoint x control register (OTG_DIEPCTLx) for USB_OTG HS</a>
OTG_DIEPINTx	0x908 0x928 .... 0x988	<a href="#">Section 37.15.52: OTG device IN endpoint x interrupt register (OTG_DIEPINTx) for USB_OTG FS</a>
OTG_DIEPINTx	0x908 0x928 ... 0x9E8	<a href="#">Section 37.15.52: OTG device IN endpoint x interrupt register (OTG_DIEPINTx) for USB_OTG HS</a>
OTG_DIEPTSIZ0	0x910	<a href="#">Section 37.15.53: OTG device IN endpoint 0 transfer size register (OTG_DIEPTSIZ0)</a>
OTG_DIEPDMAx	0x914 0x934 ... 0x9F4	<a href="#">Section 37.15.54: OTG device IN endpoint x DMA address register (OTG_DIEPDMAx)</a>
OTG_DTXFSTSx	0x918 0x938 .... 0x998	<a href="#">Section 37.15.55: OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) for USB_OTG FS</a>
OTG_DTXFSTSx	0x918 0x938 .... 0x9F8	<a href="#">Section 37.15.55: OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) for USB_OTG HS</a>
OTG_DIEPTSIZx	0x930 0x950 ... 0x9B0	<a href="#">Section 37.15.56: OTG device IN endpoint x transfer size register (OTG_DIEPTSIZx) for USB_OTG FS</a>

**Table 270. Device-mode control and status registers (continued)**

Acronym	Offset address	Register name
OTG_DIEPTSI <sub>Zx</sub>	0x930 0x950 ... 0x9F0	<a href="#">Section 37.15.56: OTG device IN endpoint x transfer size register (OTG_DIEPTSI<sub>Zx</sub>) for USB_OTG HS</a>
OTG_DOEPCTL0	0xB00	<a href="#">Section 37.15.57: OTG device control OUT endpoint 0 control register (OTG_DOEPCTL0)</a>
OTG_DOEPINTx	0xB08 0xB28 ... 0xBA8	<a href="#">Section 37.15.58: OTG device OUT endpoint x interrupt register (OTG_DOEPINTx) for USB_OTG FS</a>
OTG_DOEPINTx	0xB08 0XB28 ... 0xC08	<a href="#">Section 37.15.58: OTG device OUT endpoint x interrupt register (OTG_DOEPINTx) for USB_OTG HS</a>
OTG_DOEPTSI0	0xB10	<a href="#">Section 37.15.59: OTG device OUT endpoint 0 transfer size register (OTG_DOEPTSI0)</a>
OTG_DOEPDMAx	0xB14 0xB34 ... 0xC14	<a href="#">Section 37.15.60: OTG device OUT endpoint x DMA address register (OTG_DOEPDMAx)</a>
OTG_DOEPCTLx	0xB20 0xB40 ... 0xBA0	<a href="#">Section 37.15.61: OTG device OUT endpoint x control register (OTG_DOEPCTLx) for USB_OTG FS</a>
OTG_DOEPCTLx	0xB20 0xB40 ... 0xC00	<a href="#">Section 37.15.61: OTG device OUT endpoint x control register (OTG_DOEPCTLx) for USB_OTG HS</a>
OTG_DOEPTSI <sub>Zx</sub>	0xB30 0xB50 ... 0xBB0	<a href="#">Section 37.15.62: OTG device OUT endpoint x transfer size register (OTG_DOEPTSI<sub>Zx</sub>) for USB_OTG FS</a>
OTG_DOEPTSI <sub>Zx</sub>	0xB30 0xB50 ... 0xBF0	<a href="#">Section 37.15.62: OTG device OUT endpoint x transfer size register (OTG_DOEPTSI<sub>Zx</sub>) for USB_OTG HS</a>

**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 271. Data FIFO (DFIFO) access register map**

FIFO access register section	Offset address	Access
Device IN endpoint 0/Host OUT Channel 0: DFIFO write access Device OUT endpoint 0/Host IN Channel 0: DFIFO read access	0x1000–0x1FFC	w r
Device IN endpoint 1/Host OUT Channel 1: DFIFO write access Device OUT endpoint 1/Host IN Channel 1: DFIFO read access	0x2000–0x2FFC	w r
...	...	...
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 272. Power and clock gating control and status registers**

Acronym	Offset address	Register name
OTG_PCGCCTL	0xE00–0xE04	<a href="#">Section 37.15.63: OTG power and clock gating control register (OTG_PCGCCTL)</a>

## 37.15 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.

### 37.15.1 OTG control and status register (OTG\_GOTGCTL)

Address offset: 0x000

Reset value: 0x0001 0000

The OTG\_GOTGCTL register controls the behavior and reflects the status of the OTG function of the core.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CUR MOD	OTG VER	BSVLD	ASVLD	DBCT	CID STS
										r	rw	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	EHEN	DHNP EN	HSHNP EN	HNP RQ	HNG SCS	BVALO VAL	BVALO EN	AVALO VAL	AVALO EN	VBVAL OVAL	VBVAL OEN	SRQ	SRQ SCS
			rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	r

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **CURMOD:** Current mode of operation

Indicates the current mode (host or device).

0: Device mode

1: Host mode

Bit 20 **OTGVER:** OTG version

Selects the OTG revision.

0:OTG Version 1.3. OTG1.3 is obsolete for new product development.

1:OTG Version 2.0. In this version the core supports only data line pulsing for SRP.

Bit 19 **BSVLD:** B-session valid

Indicates the device mode transceiver status.

0: B-session is not valid.

1: B-session is valid.

In OTG mode, the user can use this bit to determine if the device is connected or disconnected.

*Note: Only accessible in device mode.*

Bit 18 **ASVLD:** A-session valid

Indicates the host mode transceiver status.

0: A-session is not valid

1: A-session is valid

*Note: Only accessible in host mode.*

Bit 17 **DBCT:** Long/short debounce time

Indicates the debounce time of a detected connection.

0: Long debounce time, used for physical connections (100 ms + 2.5 µs)

1: Short debounce time, used for soft connections (2.5 µs)

*Note: Only accessible in host mode.*

Bit 16 **CIDSTS:** Connector ID status

Indicates the connector ID status on a connect event.

0: The OTG\_FS/OTG\_HS controller is in A-device mode

1: The OTG\_FS/OTG\_HS controller is in B-device mode

*Note: Accessible in both device and host modes.*

Bits 15:13 Reserved, must be kept at reset value.

Bit 12 **EHEN:** Embedded host enable

It is used to select between OTG A device state machine and embedded host state machine.

0: OTG A device state machine is selected

1: Embedded host state machine is selected

Bit 11 **DHNPN:** Device HNP enabled

The application sets this bit when it successfully receives a SetFeature.SethNPEnable command from the connected USB host.

0: HNP is not enabled in the application

1: HNP is enabled in the application

*Note: Only accessible in device mode.*

Bit 10 **HSHNPEN:** host set HNP enable

The application sets this bit when it has successfully enabled HNP (using the SetFeature.SetHNPEnable command) on the connected device.

- 0: Host Set HNP is not enabled
- 1: Host Set HNP is enabled

*Note: Only accessible in host mode.*

Bit 9 **HNPRQ:** HNP request

The application sets this bit to initiate an HNP request to the connected USB host. The application can clear this bit by writing a 0 when the host negotiation success status change bit in the OTG\_GOTGINT register (HNSSCHG bit in OTG\_GOTGINT) is set. The core clears this bit when the HNSSCHG bit is cleared.

- 0: No HNP request
- 1: HNP request

*Note: Only accessible in device mode.*

Bit 8 **HNGSCS:** Host negotiation success

The core sets this bit when host negotiation is successful. The core clears this bit when the HNP request (HNPRQ) bit in this register is set.

- 0: Host negotiation failure
- 1: Host negotiation success

*Note: Only accessible in device mode.*

Bit 7 **BVALOVAL:** B-peripheral session valid override value.

This bit is used to set override value for Bvalid signal when BVALOEN bit is set.

- 0: Bvalid value is '0' when BVALOEN = 1
- 1: Bvalid value is '1' when BVALOEN = 1

*Note: Only accessible in device mode.*

Bit 6 **BVALOEN:** B-peripheral session valid override enable.

This bit is used to enable/disable the software to override the Bvalid signal using the BVALOVAL bit.

- 0:Override is disabled and Bvalid signal from the respective PHY selected is used internally by the core
- 1:Internally Bvalid received from the PHY is overridden with BVALOVAL bit value

*Note: Only accessible in device mode.*

Bit 5 **AVALOVAL:** A-peripheral session valid override value.

This bit is used to set override value for Avalid signal when AVALOEN bit is set.

- 0: Avalid value is '0' when AVALOEN = 1
- 1: Avalid value is '1' when AVALOEN = 1

*Note: Only accessible in host mode.*

Bit 4 **AVALOEN:** A-peripheral session valid override enable.

This bit is used to enable/disable the software to override the Avalid signal using the AVALOVAL bit.

- 0:Override is disabled and Avalid signal from the respective PHY selected is used internally by the core
- 1:Internally Avalid received from the PHY is overridden with AVALOVAL bit value

*Note: Only accessible in host mode.*

Bit 3 **VBVALOVAL**: V<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 the user uses 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.

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.*

### 37.15.2 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.	Res.	DBC DNE	ADTO CHG	HNG DET	Res.						
												rc_w1	rc_w1	rc_w1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	HNSS CHG	SRSS CHG	Res.	Res.	Res.	Res.	SEDET	Res.	Res.	
						rc_w1	rc_w1					rc_w1			

Bits 31:20 Reserved, must be kept at reset value.

Bit 19 **DBC DONE**: 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.

### 37.15.3 OTG AHB configuration register (OTG\_GAHBCFG)

Address offset: 0x008

Reset value: 0x0000 0000

This register can be used to configure the core after power-on or a change in mode. This register mainly contains AHB system-related configuration parameters. Do not change this register after the initial programming. The application must program this register before starting any transactions on either the AHB or the USB.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PTXFE LVL	TXFE LVL	Res.	Res.	Res.	Res.	Res.	Res.	GINT MSK						
							rw	rw							rw

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PTXFE LVL	TXFE LVL	Res.	DMAEN	HBSTLEN[3:0]				GINT MSK						
							rw	rw		rw	rw	rw	rw	rw	rw

Note: Configuration register for USB OTG HS

Bits 31:9 Reserved, must be kept at reset value.

Bit 8 **PTXFELVL:** Periodic Tx FIFO empty level

Indicates when the periodic Tx FIFO empty interrupt bit in the OTG\_GINTSTS register (PTXFE bit in OTG\_GINTSTS) is triggered.

- 0: PTXFE (in OTG\_GINTSTS) interrupt indicates that the Periodic Tx FIFO is half empty
- 1: PTXFE (in OTG\_GINTSTS) interrupt indicates that the Periodic Tx FIFO is completely empty

*Note: Only accessible in host mode.*

Bit 7 **TXFELVL:** Tx FIFO empty level

In device mode, this bit indicates when IN endpoint Transmit FIFO empty interrupt (TXFE in OTG\_DIEPINTx) is triggered:

- 0: The TXFE (in OTG\_DIEPINTx) interrupt indicates that the IN endpoint Tx FIFO is half empty
- 1: The TXFE (in OTG\_DIEPINTx) interrupt indicates that the IN endpoint Tx FIFO is completely empty

In host mode, this bit indicates when the nonperiodic Tx FIFO empty interrupt (NPTXFE bit in OTG\_GINTSTS) is triggered:

- 0: The NPTXFE (in OTG\_GINTSTS) interrupt indicates that the nonperiodic Tx FIFO is half empty
- 1: The NPTXFE (in OTG\_GINTSTS) interrupt indicates that the nonperiodic Tx FIFO is completely empty

Bits 6:1 Reserved, must be kept at reset value for USB OTG FS.

Bit 6 Reserved, must be kept at reset value for USB OTG HS.

Bit 5 **DMAEN:** DMA enabled for USB OTG HS

- 0: The core operates in slave mode
- 1: The core operates in DMA mode

Bits 4:1 **HBSTLEN[3:0]:** Burst length/type for USB OTG HS

0000 Single: Bus transactions use single 32 bit accesses (not recommended)

0001 INCR: Bus transactions use unspecified length accesses (not recommended, uses the INCR AHB bus command)

0011 INCR4: Bus transactions target 4x 32 bit accesses

0101 INCR8: Bus transactions target 8x 32 bit accesses

0111 INCR16: Bus transactions based on 16x 32 bit accesses

Others: Reserved

Bit 0 **GINTMSK:** Global interrupt mask

The application uses this bit to mask or unmask the interrupt line assertion to itself.

Irrespective of this bit's setting, the interrupt status registers are updated by the core.

- 0: Mask the interrupt assertion to the application.
- 1: Unmask the interrupt assertion to the application.

*Note: Accessible in both device and host modes.*

### 37.15.4 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[3:0]				HNP CAP	SRP CAP	Res.	PHY SEL	Res.	Res.	Res.	TOCAL[2:0]		
		rw	rw	rw	rw	rw	rw		r				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.	FD MOD	FH MOD	Res.	Res.	Res.	ULPI IPD	PTCI	PCCI	TSDPS	ULPIE VBUSI	ULPIE VBUSD	ULPI CSM	ULPI AR	ULPI FSL	Res.
	rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PHYL PC	Res.	TRDT[3:0]				HNP CAP	SRP CAP	Res.	PHY SEL	Res.	Res.	Res.	TOCAL[2:0]		
rw		rw	rw	rw	rw	rw	rw		rw				rw	rw	rw

Note: Configuration register for USB OTG HS

Bit 31 Reserved, must be kept at reset value.

Bit 30 **FDMOD:** Force device mode

Writing a 1 to this bit, forces the core to device mode irrespective of the OTG\_ID input pin.

0: Normal mode

1: Force device mode

After setting the force bit, the application must wait at least 25 ms before the change takes effect.

Note: Accessible in both device and host modes.

Bit 29 **FHMOD:** Force host mode

Writing a 1 to this bit, forces the core to host mode irrespective of the OTG\_ID input pin.

0: Normal mode

1: Force host mode

After setting the force bit, the application must wait at least 25 ms before the change takes effect.

Note: Accessible in both device and host modes.

Bits 28:26 Reserved, must be kept at reset value.

**Bit 25 **ULPIIPD:** ULPI interface protect disable for USB OTG HS**

This bit controls the circuitry built in the PHY to protect the ULPI interface when the link tri-states stp and data. Any pull-up or pull-down resistors employed by this feature can be disabled. Refer to the ULPI specification for more details.

- 0: Enables the interface protection circuit
- 1: Disables the interface protection circuit

**Bit 24 **PTCI:** Indicator pass through for USB OTG HS**

This bit controls whether the complement output is qualified with the internal  $V_{BUS}$  valid comparator before being used in the  $V_{BUS}$  state in the RX CMD. Refer to the ULPI specification for more details.

- 0: Complement Output signal is qualified with the Internal  $V_{BUS}$  valid comparator
- 1: Complement Output signal is not qualified with the Internal  $V_{BUS}$  valid comparator

**Bit 23 **PCCI:** Indicator complement for USB OTG HS**

This bit controls the PHY to invert the ExternalVbusIndicator input signal, and generate the complement output. Refer to the ULPI specification for more details.

- 0: PHY does not invert the ExternalVbusIndicator signal
- 1: PHY inverts ExternalVbusIndicator signal

**Bit 22 **TSDPS:** TermSel DLine pulsing selection for USB OTG HS**

This bit selects utmi\_termselect to drive the data line pulse during SRP (session request protocol).

- 0: Data line pulsing using utmi\_txvalid (default)
- 1: Data line pulsing using utmi\_termsel

**Bit 21 **ULPIEVBUSI:** ULPI external  $V_{BUS}$  indicator for USB OTG HS**

This bit indicates to the ULPI PHY to use an external  $V_{BUS}$  overcurrent indicator.

- 0: PHY uses an internal  $V_{BUS}$  valid comparator
- 1: PHY uses an external  $V_{BUS}$  valid comparator

**Bit 20 **ULPIEVBUSD:** ULPI External  $V_{BUS}$  Drive for USB OTG HS**

This bit selects between internal or external supply to drive 5 V on  $V_{BUS}$ , in the ULPI PHY.

- 0: PHY drives  $V_{BUS}$  using internal charge pump (default)
- 1: PHY drives  $V_{BUS}$  using external supply.

**Bit 19 **ULPICSM:** ULPI clock SuspendM for USB OTG HS**

This bit sets the ClockSuspendM bit in the interface control register on the ULPI PHY. This bit applies only in the serial and carkit modes.

- 0: PHY powers down the internal clock during suspend
- 1: PHY does not power down the internal clock

**Bit 18 **ULPIAR:** ULPI Auto-resume for USB OTG HS**

This bit sets the AutoResume bit in the interface control register on the ULPI PHY.

- 0: PHY does not use AutoResume feature
- 1: PHY uses AutoResume feature

**Bit 17 **ULPIFSLS:** ULPI FS/LS select for USB OTG HS**

The application uses this bit to select the FS/LS serial interface for the ULPI PHY. This bit is valid only when the FS serial transceiver is selected on the ULPI PHY.

- 0: ULPI interface
- 1: ULPI FS/LS serial interface

**Bit 16 Reserved, must be kept at reset value.**

Bit 15 **PHYLPC:** PHY Low-power clock select for USB OTG HS

This bit selects either 480 MHz or 48 MHz (low-power) PHY mode. In FS and LS modes, the PHY can usually operate on a 48 MHz clock to save power.

0: 480 MHz internal PLL clock

1: 48 MHz external clock

In 480 MHz mode, the UTMI interface operates at either 60 or 30 MHz, depending on whether the 8- or 16-bit data width is selected. In 48 MHz mode, the UTMI interface operates at 48 MHz in FS and LS modes.

Bit 14 Reserved, must be kept at reset value.

Bits 13:10 **TRDT[3:0]:** USB turnaround time

These bits are used to set the turnaround time in PHY clocks. They must be configured according to [Table 273: TRDT values \(FS\)](#) or [Table 274: 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 mode select for USB OTG FS

This bit is always 1 with read-only access.

Bit 6 **PHYSEL:** Full speed serial transceiver mode select for USB OTG HS

0: USB 2.0 external ULPI high-speed PHY.

1: USB 1.1 full-speed serial mode.

Bit 5 Reserved, must be kept at reset value.

Bit 4 Reserved, must be kept at reset value.

Bit 3 Reserved, must be kept at reset value.

Bits 2:0 **TOCAL[2:0]:** FS timeout calibration

The number of PHY clocks that the application programs in this field is added to the full-speed interpacket timeout duration in the core to account for any additional delays introduced by the PHY. This can be required, because the delay introduced by the PHY in generating the line state condition can vary from one PHY to another.

The USB standard timeout value for full-speed operation is 16 to 18 (inclusive) bit times. The application must program this field based on the speed of enumeration. The number of bit times added per PHY clock is 0.25 bit times.

**Table 273. 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 274. TRDT values (HS)**

AHB frequency range (MHz)		TRDT minimum value
Min	Max	
30	-	0x9

### 37.15.5 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																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	Res.	Res.	TXFNUM[4:0]						TXF FLSH	RXF FLSH	Res.	FCRST	PSRST	CSRST
					rw	rw	rw	rw	rw	rs	rs		rs	rs	rs	

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
AHB IDL	DMAR EQ	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
r	r															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	Res.	Res.	TXFNUM[4:0]						TXF FLSH	RXF FLSH	Res.	FCRST	PSRST	CSRST
					rw	rw	rw	rw	rw	rs	rs		rs	rs	rs	

*Note:* Configuration register for USB OTG HS

Bit 31 **AHBIDL:** AHB master idle

Indicates that the AHB master state machine is in the Idle condition.

*Note:* Accessible in both device and host modes.

Bits 30:11 Reserved, must be kept at reset value for USB OTG FS.

Bit 30 **DMAREQ:** DMA request signal enabled for USB OTG HS

This bit indicates that the DMA request is in progress. Used for debug.

Bits 29:11 Reserved, must be kept at reset value for USB OTG HS.

Bits 10:6 **TXFNUM[4:0]:** Tx FIFO number

This is the FIFO number that must be flushed using the Tx FIFO Flush bit. This field must not be changed until the core clears the Tx FIFO Flush bit.

00000:

- Non-periodic Tx FIFO flush in host mode
- Tx FIFO 0 flush in device mode

00001:

- Periodic Tx FIFO flush in host mode
- Tx FIFO 1 flush in device mode

00010: Tx FIFO 2 flush in device mode

...

01111: Tx FIFO 15 flush in device mode

10000: Flush all the transmit FIFOs in device or host mode.

*Note:* Accessible in both device and host modes.

Bit 5 **TXFFLSH:** Tx FIFO flush

This bit selectively flushes a single or all transmit FIFOs, but cannot do so if the core is in the midst of a transaction.

The application must write this bit only after checking that the core is neither writing to the Tx FIFO nor reading from the Tx FIFO. Verify using these registers:

Read—NAK Effective interrupt ensures the core is not reading from the FIFO

Write—AHBIDL bit in OTG\_GRSTCTL ensures the core is not writing anything to the FIFO.

Flushing is normally recommended when FIFOs are reconfigured. FIFO flushing is also recommended during device endpoint disable. The application must wait until the core clears this bit before performing any operations. This bit takes eight clocks to clear, using the slower clock of phy\_clk or hclk.

*Note:* Accessible in both device and host modes.

**Bit 4 RXFFLSH:** Rx FIFO flush

The application can flush the entire Rx FIFO using this bit, but must first ensure that the core is not in the middle of a transaction.

The application must only write to this bit after checking that the core is neither reading from the Rx FIFO nor writing to the Rx FIFO.

The application must wait until the bit is cleared before performing any other operations. This bit requires 8 clocks (slowest of PHY or AHB clock) to clear.

*Note: Accessible in both device and host modes.*

**Bit 3 Reserved, must be kept at reset value.****Bit 2 FCRST:** Host frame counter reset

The application writes this bit to reset the micro-frame/frame number counter inside the core. When the micro-frame/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 gets cleared by the core in a few clock cycles.

*Note: Only accessible in host mode.*

**Bit 1 PSRST:** Partial soft reset

Resets the internal state machines but keeps the enumeration info. Can be used to recover some specific PHY errors.

*Note: Accessible in both device and host modes.*

**Bit 0 CSRST:** Core soft reset

Resets the HCLK and PHY clock domains as follows:

Clears the interrupts and all the CSR register bits except for the following bits:

- GATEHCLK bit in OTG\_PCGCCTL
- STPPCLK bit in OTG\_PCGCCTL
- FSLSPCS bits in OTG\_HCFG
- DSPD bit in OTG\_DCFG
- SDIS bit in OTG\_DCTL
- OTG\_GCCFG register

All module state machines (except for the AHB slave unit) are reset to the Idle state, and all the transmit FIFOs and the receive FIFO are flushed.

Any transactions on the AHB Master are terminated as soon as possible, after completing the last data phase of an AHB transfer. Any transactions on the USB are terminated immediately.

The application can write to this bit any time it wants to reset the core. This is a self-clearing bit and the core clears this bit after all the necessary logic is reset in the core, which can take several clocks, depending on the current state of the core. Once this bit has been cleared, the software must wait at least 3 PHY clocks before accessing the PHY domain

(synchronization delay). The software must also check that bit 31 in this register is set to 1 (AHB Master is Idle) before starting any operation.

Typically, the software reset is used during software development and also when the user dynamically changes the PHY selection bits in the above listed USB configuration registers. When the user changes 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.*

### 37.15.6 OTG core interrupt register (OTG\_GINTSTS)

Address offset: 0x014

Reset value: 0x0400 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	LPM INT	PTXFE	HCINT	HPRT INT	RST DET	DATAF SUSP	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	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 wake-up detected interrupt

Wake-up 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 wake-up is detected on the USB.

- During LPM(L1):

This interrupt is asserted for either host initiated resume or device initiated remote wake-up 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 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 22 Reserved, must be kept at reset value for USB OTG FS.

Bit 22 **DATAFSUSP:** Data fetch suspended for USB OTG HS

This interrupt is valid only in DMA mode. This interrupt indicates that the core has stopped fetching data for IN endpoints due to the unavailability of TxFIFO space or request queue space. This interrupt is used by the application for an endpoint mismatch algorithm. For example, after detecting an endpoint mismatch, the application:

- Sets a global nonperiodic IN NAK handshake
- Disables IN endpoints
- Flushes the FIFO
- Determines the token sequence from the IN token sequence learning queue
- Re-enables the endpoints

Clears the global nonperiodic IN NAK handshake. If the global nonperiodic IN NAK is cleared, the core has not yet fetched data for the IN endpoint, and the IN token is received: the core generates an “IN token received when FIFO empty” interrupt. The OTG then sends a NAK response to the host. To avoid this scenario, the application can check the FetSusp interrupt in OTG\_GINTSTS, which ensures that the FIFO is full before clearing a global NAK handshake. Alternatively, the application can mask the “IN token received when FIFO empty” interrupt when clearing a global IN NAK handshake.

Bit 21 **IPXFR:** Incomplete periodic transfer

In host mode, the core sets this interrupt bit when there are incomplete periodic transactions still pending, which are scheduled for the current frame.

**INCOMPISOOUT:** Incomplete isochronous OUT transfer

In device mode, the core sets this interrupt to indicate that there is at least one isochronous OUT endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the End of periodic frame interrupt (EOPF) bit in this register.

Bit 20 **IISOIXFR:** Incomplete isochronous IN transfer

The core sets this interrupt to indicate that there is at least one isochronous IN endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the End of periodic frame interrupt (EOPF) bit in this register.

*Note: Only accessible in device mode.*

Bit 19 **OEPINT:** OUT endpoint interrupt

The core sets this bit to indicate that an interrupt is pending on one of the OUT endpoints of the core (in device mode). The application must read the OTG\_DAINT register to determine the exact number of the OUT endpoint on which the interrupt occurred, and then read the corresponding OTG\_DOEPINTx register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding OTG\_DOEPINTx register to clear this bit.

*Note: Only accessible in device mode.*

Bit 18 **IEPINT:** IN endpoint interrupt

The core sets this bit to indicate that an interrupt is pending on one of the IN endpoints of the core (in device mode). The application must read the OTG\_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.*

### 37.15.7 OTG interrupt mask register (OTG\_GINTMSK)

Address offset: 0x018

Reset value: 0x0000 0000

This register works with the core interrupt register to interrupt the application. When an interrupt bit is masked, the interrupt associated with that bit is not generated. However, the core interrupt (OTG\_GINTSTS) register bit corresponding to that interrupt is still set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WUIM	SRQIM	DISCINT	CIDSC HGM	LPMIN TM	PTXFE M	HCIM	PRTIM	RSTDE TM	Res.	IPXFR M/IISO OXFR M	IISOIX FRM	OEPIN T	IEPINT	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPFM	ISOOD RPM	ENUM DNEM	USBRST	USBSU SPM	ESUSPM	Res.	Res.	GONA KEFFM	GINAK EFFM	NPTXF EM	RXFLV LM	SOFM	OTGIN T	MMISM	Res.
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	rw	

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WUIM	SRQIM	DISCINT	CIDSC HGM	LPMIN TM	PTXFE M	HCIM	PRTIM	RSTDE TM	FSUS PM	IPXFR M/IISO OXFR M	IISOIX FRM	OEPIN T	IEPINT	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPFM	ISOOD RPM	ENUM DNEM	USBRST	USBSU SPM	ESUSPM	Res.	Res.	GONA KEFFM	GINAK EFFM	NPTXF EM	RXFLV LM	SOFM	OTGIN T	MMISM	Res.
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	rw	

Note: Configuration register for USB OTG HS

Bit 31 **WUIM**: Resume/remote wake-up detected interrupt mask

0: Masked interrupt

1: Unmasked interrupt

Note: Accessible in both host and device modes.

Bit 30 **SRQIM**: Session request/new session detected interrupt mask

0: Masked interrupt

1: Unmasked interrupt

Note: Accessible in both host and device modes.

Bit 29 **DISCINT**: Disconnect detected interrupt mask

0: Masked interrupt

1: Unmasked interrupt

Note: Only accessible in host mode.

Bit 28 **CIDSC HGM**: Connector ID status change mask

0: Masked interrupt

1: Unmasked interrupt

Note: Accessible in both host and device modes.

Bit 27 **LPMINTM**: LPM interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Accessible in both host and device modes.

Bit 26 **PTXFEM**: Periodic Tx FIFO empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in host mode.

Bit 25 **HCIM**: Host channels interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in host mode.

Bit 24 **PRTIM**: Host port interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in host mode.

Bit 23 **RSTDETM**: Reset detected interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in device mode.

Bit 22 Reserved, must be kept at reset value for USB OTG FS.

Bit 22 **FSUSPM**: Data fetch suspended mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Only accessible in peripheral mode.

Bit 21 **IPXFRM**: Incomplete periodic transfer mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in host mode.

**IISOXXFRM**: Incomplete isochronous OUT transfer mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in device mode.

Bit 20 **IISOIXFRM**: Incomplete isochronous IN transfer mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in device mode.

Bit 19 **OEPINT**: OUT endpoints interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in device mode.

Bit 18 **IEPINT**: IN endpoints interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note:* Only accessible in device mode.

Bits 17:16 Reserved, must be kept at reset value.

Bit 15 **EOPFM:** End of periodic frame interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in device mode.*

Bit 14 **ISOODRPM:** Isochronous OUT packet dropped interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in device mode.*

Bit 13 **ENUMDNEM:** Enumeration done mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in device mode.*

Bit 12 **USBRST:** USB reset mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in device mode.*

Bit 11 **USBSUSPM:** USB suspend mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in device mode.*

Bit 10 **ESUSPM:** Early suspend mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in device mode.*

Bits 9:8 Reserved, must be kept at reset value.

Bit 7 **GONAKEFFM:** Global OUT NAK effective mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in device mode.*

Bit 6 **GINAKEFFM:** Global non-periodic IN NAK effective mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in device mode.*

Bit 5 **NPTXFEM:** Non-periodic Tx FIFO empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Only accessible in host mode.*

Bit 4 **RXFLVLM:** Receive FIFO non-empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

*Note: Accessible in both device and host modes.*

Bit 3 **SOFM:** Start of frame mask

0: Masked interrupt

1: Unmasked interrupt

*Note: Accessible in both device and host modes.*

Bit 2 **OTGINT:** OTG interrupt mask

0: Masked interrupt

1: Unmasked interrupt

*Note: Accessible in both device and host modes.*

Bit 1 **MMISM:** Mode mismatch interrupt mask

0: Masked interrupt

1: Unmasked interrupt

*Note: Accessible in both device and host modes.*

Bit 0 Reserved, must be kept at reset value.

### 37.15.8 OTG receive status debug read register (OTG\_GRXSTSR)

Address offset for read: 0x01C

Reset value: 0x0000 0000

This description is for register OTG\_GRXSTSR in Device mode.

A read to the receive status debug read register returns the contents of the top of the receive FIFO.

The core ignores the receive status read when the receive FIFO is empty and returns a value of 0x0000 0000.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	STSPH ST	Res.	Res.	FRMNUM[3:0]				PKTSTS[3:0]				DPID[1]
				r			r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DPID[0]	BCNT[10:0]										EPNUM[3:0]				
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:28 Reserved, must be kept at reset value.

Bit 27 **STSPHST:** Status phase start

Indicates the start of the status phase for a control write transfer. This bit is set along with the OUT transfer completed PKTSTS pattern.

Bits 26:25 Reserved, must be kept at reset value.

Bits 24:21 **FRMNUM[3:0]:** Frame number

This is the least significant 4 bits of the frame number in which the packet is received on the USB. This field is supported only when isochronous OUT endpoints are supported.

Bits 20:17 **PKTSTS[3:0]**: Packet status

Indicates the status of the received packet

0001: Global OUT NAK (triggers an interrupt)

0010: OUT data packet received

0011: OUT transfer completed (triggers an interrupt)

0100: SETUP transaction completed (triggers an interrupt)

0110: SETUP data packet received

Others: Reserved

Bits 16:15 **DPID[1:0]**: Data PID

Indicates the data PID of the received OUT data packet

00: DATA0

10: DATA1

01: DATA2

11: MDATA

Bits 14:4 **BCNT[10:0]**: Byte count

Indicates the byte count of the received data packet.

Bits 3:0 **EPNUM[3:0]**: Endpoint number

Indicates the endpoint number to which the current received packet belongs.

**37.15.9 OTG receive status debug read [alternate] (OTG\_GRXSTSR)**

Address offset for read: 0x01C

Reset value: 0x0000 0000

This description is for register OTG\_GRXSTSR in Host mode.

A read to the receive status debug read register returns the contents of the top of the receive FIFO.

The core ignores the receive status read when the receive FIFO is empty and returns a value of 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.	PKTSTS[3:0]				DPID	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DPID	BCNT[10:0]														CHNUM[3:0]	
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bits 31:21 Reserved, must be kept at reset value.

Bits 20:17 **PKTSTS[3:0]**: Packet status

Indicates the status of the received packet

0010: IN data packet received

0011: IN transfer completed (triggers an interrupt)

0101: Data toggle error (triggers an interrupt)

0111: Channel halted (triggers an interrupt)

Others: Reserved

Bits 16:15 **DPID[1:0]**: Data PID

Indicates the data PID of the received packet

00: DATA0

10: DATA1

01: DATA2

11: MDATA

Bits 14:4 **BCNT[10:0]**: Byte count

Indicates the byte count of the received IN data packet.

Bits 3:0 **CHNUM[3:0]**: Channel number

Indicates the channel number to which the current received packet belongs.

### 37.15.10 OTG status read and pop registers (OTG\_GRXSTSP)

Address offset for pop: 0x020

Reset value: 0x0000 0000

This description is for register OTG\_GRXSTSP in Device mode.

Similarly to OTG\_GRXSTSR (receive status debug read register) where a read returns the contents of the top of the receive FIFO, a read to OTG\_GRXSTSP (receive status read and pop register) additionally pops the top data entry out of the Rx FIFO.

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.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	STSPH ST	Res.	Res.	FRMNUM[3:0]				PKTSTS[3:0]				DPID[1]
				r			r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DPID[0]	BCNT[10:0]										EPNUM[3:0]				
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:28 Reserved, must be kept at reset value.

Bit 27 **STSPHST**: Status phase start

Indicates the start of the status phase for a control write transfer. This bit is set along with the OUT transfer completed PKTSTS pattern.

Bits 26:25 Reserved, must be kept at reset value.

Bits 24:21 **FRMNUM[3:0]**: Frame number

This is the least significant 4 bits of the frame number in which the packet is received on the USB. This field is supported only when isochronous OUT endpoints are supported.

Bits 20:17 **PKTSTS[3:0]**: Packet status

Indicates the status of the received packet

0001: Global OUT NAK (triggers an interrupt)

0010: OUT data packet received

0011: OUT transfer completed (triggers an interrupt)

0100: SETUP transaction completed (triggers an interrupt)

0110: SETUP data packet received

Others: Reserved

Bits 16:15 **DPID[1:0]**: Data PID

Indicates the data PID of the received OUT data packet

00: DATA0

10: DATA1

01: DATA2

11: MDATA

Bits 14:4 **BCNT[10:0]**: Byte count

Indicates the byte count of the received data packet.

Bits 3:0 **EPNUM[3:0]**: Endpoint number

Indicates the endpoint number to which the current received packet belongs.

### 37.15.11 OTG status read and pop registers [alternate] (OTG\_GRXSTSP)

Address offset for pop: 0x020

Reset value: 0x0000 0000

This description is for register OTG\_GRXSTSP in Host mode.

Similarly to OTG\_GRXSTSR (receive status debug read register) where a read returns the contents of the top of the receive FIFO, a read to OTG\_GRXSTSP (receive status read and pop register) additionally pops the top data entry out of the Rx FIFO.

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.

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.	PKTSTS[3:0]			DPID	
											r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DPID	BCNT[10:0]														CHNUM[3:0]
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:21 Reserved, must be kept at reset value.

Bits 20:17 **PKTSTS[3:0]**: Packet status

Indicates the status of the received packet

0010: IN data packet received

0011: IN transfer completed (triggers an interrupt)

0101: Data toggle error (triggers an interrupt)

0111: Channel halted (triggers an interrupt)

Others: Reserved

Bits 16:15 **DPID[1:0]**: Data PID

Indicates the data PID of the received packet

00: DATA0

10: DATA1

01: DATA2

11: MDATA

Bits 14:4 **BCNT[10:0]**: Byte count

Indicates the byte count of the received IN data packet.

Bits 3:0 **CHNUM[3:0]**: Channel number

Indicates the channel number to which the current received packet belongs.

### 37.15.12 OTG receive FIFO size register (OTG\_GRXFSIZ)

Address offset: 0x024

Reset value: 0x0000 0200 for USB OTG FS

Reset value: 0x0000 0400 for USB OTG HS

The application can program the RAM size that must be allocated to the Rx FIFO.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
RXFD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **RXFD[15:0]**: Rx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Maximum value is 1024

Programmed values must respect the available FIFO memory allocation and must not exceed the power-on value.

### 37.15.13 OTG host non-periodic transmit FIFO size register (OTG\_HNPTXFSIZ)/Endpoint 0 Transmit FIFO size (OTG\_DIEPTXF0)

Address offset: 0x028

Reset value: 0x0200 0200

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NPTXFD/TX0FD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NPTXFSA/TX0FSA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Host mode

Bits 31:16 **NPTXFD[15:0]**: Non-periodic Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Programmed values must respect the available FIFO memory allocation and must not exceed the power-on value.

Bits 15:0 **NPTXFSA[15:0]**: Non-periodic transmit RAM start address

This field configures the memory start address for non-periodic transmit FIFO RAM.

#### Device mode

Bits 31:16 **TX0FD**: Endpoint 0 Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Programmed values must respect the available FIFO memory allocation and must not exceed the power-on value.

Bits 15:0 **TX0FSA**: Endpoint 0 transmit RAM start address

This field configures the memory start address for the endpoint 0 transmit FIFO RAM.

### 37.15.14 OTG non-periodic transmit FIFO/queue status register (OTG\_HNPTXSTS)

Address offset: 0x02C

Reset value: 0x0008 0200 for USB OTG FS

Reset value: 0x0008 0400 for USB OTG HS

**Note:** *In device mode, this register is not valid.*

This read-only register contains the free space information for the non-periodic Tx FIFO and the non-periodic transmit request queue.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	NPTXQTOP[6:0]								NPTQXS AV[7:0]							
	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
NPTXFSAV[15:0]																
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit 31 Reserved, must be kept at reset value.

Bits 30:24 **NPTXQTOP[6:0]:** Top of the non-periodic transmit request queue

Entry in the non-periodic Tx request queue that is currently being processed by the MAC.

Bits 30:27: Channel/endpoint number

Bits 26:25:

00: IN/OUT token

01: Zero-length transmit packet (device IN/host OUT)

11: Channel halt command

Bit 24: Terminate (last entry for selected channel/endpoint)

Bits 23:16 **NPTQXS AV[7:0]:** Non-periodic transmit request queue space available

Indicates the amount of free space available in the non-periodic transmit request queue.

This queue holds both IN and OUT requests.

0: Non-periodic transmit request queue is full

1: 1 location available

2: locations available

n: n locations available ( $0 \leq n \leq 8$ )

Others: Reserved

Bits 15:0 **NPTXFSAV[15:0]:** Non-periodic Tx FIFO space available

Indicates the amount of free space available in the non-periodic Tx FIFO.

Values are in terms of 32-bit words.

0: Non-periodic Tx FIFO is full

1: 1 word available

2: 2 words available

n: n words available (where  $0 \leq n \leq 512$ )

Others: Reserved

### 37.15.15 OTG general core configuration register (OTG\_GCCFG)

Address offset: 0x038

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	VBDEN	Res.	Res.	DCD EN	Res.	PWR DWN									
										rw			rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DCDET										
															r

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 20:19 Reserved, must be kept at reset value.

Bit 18 **DCDEN**: Data contact detection (DCD) mode enable

This bit is set by the software to put the BCD into DCD mode. Only one detection mode (DCD, PD, SD or OFF) must be selected to work correctly.

Bit 17 Reserved, must be kept at reset value.

Bit 16 **PWRDWN**: Power down control of FS PHY

Used to activate the FS PHY in transmission/reception. When reset, the PHY is kept in power-down.

0 = USB FS PHY disabled

1 = USB FS PHY enabled

Bits 15:4 Reserved, must be kept at reset value.

Bits 3:1 Reserved, must be kept at reset value.

Bit 0 **DCDET**: Data contact detection (DCD) status

This bit gives the result of DCD.

0: data lines contact not detected

1: data lines contact detected

### 37.15.16 OTG core ID register (OTG\_CID)

Address offset: 0x03C

Reset value: 0x0000 2000 for USB OTG FS

Reset value: 0x0000 2100 for USB OTG HS

This is a register containing the Product ID as reset value.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRODUCT_ID[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRODUCT_ID[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **PRODUCT\_ID[31:0]**: Product ID field

Application-programmable ID field.

### 37.15.17 OTG core LPM configuration register (OTG\_GLPMCFG)

Address offset: 0x54

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	EN BESL	LPMRCNTSTS[2:0]			SND LPM	LPMRCNT[2:0]			LPMCHIDX[3:0]			L1RSM OK	
			rw	r	r	r	rs	rw	rw	rw	rw	rw	rw	rw	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLP STS	LPMRSP[1:0]		L1DS EN	BESLTHRS[3:0]			L1SS EN	REM WAKE	BESL[3:0]			LPM ACK	LPM EN		
r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:29 Reserved, must be kept at reset value.

Bit 28 **ENBESL**: Enable best effort service latency

This bit enables the BESL feature as defined in the LPM errata:

0:The core works as described in the following document:

*USB 2.0 Link Power Management Addendum Engineering Change Notice to the USB 2.0 specification, July 16, 2007*

1:The core works as described in the LPM Errata:

*Errata for USB 2.0 ECN: Link Power Management (LPM) - 7/2007*

*Note: Only the updated behavior (described in LPM Errata) is considered in this document and so the ENBESL bit must be set to '1' by application SW.*

Bits 27:25 **LPMRCNTSTS[2:0]**: LPM retry count status

Number of LPM host retries still remaining to be transmitted for the current LPM sequence.

*Note: Accessible only in host mode.*

**Bit 24 SNDLPM:** Send LPM transaction

When the application software sets this bit, an LPM transaction containing two tokens, EXT and LPM is sent. The hardware clears this bit once a valid response (STALL, NYET, or ACK) is received from the device or the core has finished transmitting the programmed number of LPM retries.

*Note: This bit must be set only when the host is connected to a local port.*

*Note: Accessible only in host mode.*

**Bits 23:21 LPMRCNT[2:0]:** LPM retry count

When the device gives an ERROR response, this is the number of additional LPM retries that the host performs until a valid device response (STALL, NYET, or ACK) is received.

*Note: Accessible only in host mode.*

**Bits 20:17 LPMCHIDX[3:0]:** LPM Channel Index

The channel number on which the LPM transaction has to be applied while sending an LPM transaction to the local device. Based on the LPM channel index, the core automatically inserts the device address and endpoint number programmed in the corresponding channel into the LPM transaction.

*Note: Accessible only in host mode.*

**Bit 16 L1RSMOK:** Sleep state resume OK

Indicates that the device or host can start resume from Sleep state. This bit is valid in LPM sleep (L1) state. It is set in sleep mode after a delay of 50 µs ( $T_{L1Residency}$ ).

This bit is reset when SLPSTS is 0.

1: The application or host can start resume from Sleep state

0: The application or host cannot start resume from Sleep state

**Bit 15 SLPSTS:** Port sleep status

**Device mode:**

This bit is set as long as a Sleep condition is present on the USB bus. The core enters the Sleep state when an ACK response is sent to an LPM transaction and the  $T_{L1TokenRetry}$  timer has expired. To stop the PHY clock, the application must set the STPPCLK bit in OTG\_PCGCCTL, which asserts the PHY suspend input signal.

The application must rely on SLPSTS and not ACK in LPMRSP to confirm transition into sleep.

The core comes out of sleep:

- When there is any activity on the USB line state
- 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 wake-up signal,
- The application sets the PRST bit or the PRES bit in the OTG\_HPRT register, or
- The application sets the L1Resume/ remote wake-up 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 **LPMRSP[1:0]**: LPM response

**Device mode:**

The response of the core to LPM transaction received is reflected in these two bits.

**Host mode:**

Handshake response received from local device for LPM transaction

11: ACK

10: NYET

01: STALL

00: ERROR (No handshake response)

Bit 12 **L1DSEN**: L1 deep sleep enable

Enables suspending the PHY in L1 Sleep mode. For maximum power saving during L1 Sleep mode, this bit must be set to '1' by application SW in all the cases.

Bits 11:8 **BESLTHRS[3:0]**: BESL threshold

**Device mode:**

The core puts the PHY into deep low power mode in L1 when BESL value is greater than or equal to the value defined in this field BESL\_Thres[3:0].

**Host mode:**

The core puts the PHY into deep low power mode in L1. BESLTHRS[3:0] specifies the time for which resume signaling is to be reflected by host ( $T_{L1HubDrvResume2}$ ) on the USB bus when it detects device initiated resume.

BESLTHRS must not be programmed with a value greater than 1100b in host mode, because this exceeds maximum  $T_{L1HubDrvResume2}$ .

Thres[3:0] host mode resume signaling time (μs):

0000: 75

0001: 100

0010: 150

0011: 250

0100: 350

0101: 450

0110: 950

All other values: reserved

Bit 7 **L1SSEN**: L1 Shallow Sleep enable

Enables suspending the PHY in L1 Sleep mode. For maximum power saving during L1 Sleep mode, this bit must be set to '1' by application SW in all the cases.

Bit 6 **REMWAKE**: bRemoteWake value

**Host mode:**

The value of remote wake up to be sent in the wIndex field of LPM transaction.

**Device mode (read-only):**

This field is updated with the received LPM token bRemoteWake bmAttribute when an ACK, NYET, or STALL response is sent to an LPM transaction.

Bits 5:2 **BESL[3:0]**: Best effort service latency

**Host mode:**

The value of BESL to be sent in an LPM transaction. This value is also used to initiate resume for a duration  $T_{L1HubDrvResume1}$  for host initiated resume.

**Device mode (read-only):**

This field is updated with the received LPM token BESL bmAttribute when an ACK, NYET, or STALL response is sent to an LPM transaction.

BESL[3:0] $T_{BESL}$  ( $\mu$ 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-written 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

### 37.15.18 OTG host periodic transmit FIFO size register (OTG\_HPTXFSIZ)

Address offset: 0x100

Reset value: 0x0200 0400 for USB OTG FS

Reset value: 0x0400 0800 for USB OTG HS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PTXFSIZ[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PTXSA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **PTXFSIZ[15:0]**: Host periodic Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Bits 15:0 **PTXSA[15:0]**: Host periodic Tx FIFO start address

This field configures the memory start address for periodic transmit FIFO RAM.

### 37.15.19 OTG device IN endpoint transmit FIFO x size register (OTG\_DIEPTXFx)

Address offset: 0x104 + 0x04 \* (x - 1), (x = 1 to 5[FS] /8[HS])

Reset value: Block 1: 0x0200 0400

Reset value: Block 2: 0x0200 0600

Reset value: Block 3: 0x0200 0800

Reset value: Block 4: 0x0200 0A00

Reset value: Block 5: 0x0200 0C00

Reset value: Block 6: 0x0200 0E00

Reset value: Block 7: 0x0200 1000

Reset value: Block 8: 0x0200 1200

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INEPTXFD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INEPTXSA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **INEPTXFD[15:0]**: IN endpoint Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Bits 15:0 **INEPTXSA[15:0]**: IN endpoint FIFOx transmit RAM start address

This field contains the memory start address for IN endpoint transmit FIFOx. The address must be aligned with a 32-bit memory location.

### 37.15.20 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:

### 37.15.21 OTG host configuration register (OTG\_HCFG)

Address offset: 0x400

Reset value: 0x0000 0000

This register configures the core after power-on. Do not make changes to this register after initializing the host.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	FSLSS	FSLSPCS[1:0]													
													r	rw	rw

Bits 31:3 Reserved, must be kept at reset value.

Bit 2 **FSLSS**: FS- and LS-only support

The application uses this bit to control the core's enumeration speed. Using this bit, the application can make the core enumerate as an FS host, even if the connected device supports HS traffic. Do not make changes to this field after initial programming.

1: FS/LS-only, even if the connected device can support HS (read-only).

Bits 1:0 **FSLSPCS[1:0]**: FS/LS PHY clock select

When the core is in FS host mode

01: PHY clock is running at 48 MHz

Others: Reserved

When the core is in LS host mode

00: Reserved

01: Select 48 MHz PHY clock frequency

10: Select 6 MHz PHY clock frequency

11: Reserved

Note: The FSLSPCS must be set on a connection event according to the speed of the connected device (after changing this bit, a software reset must be performed).

### 37.15.22 OTG host frame interval register (OTG\_HFIR)

Address offset: 0x404

Reset value: 0x0000 EA60

This register stores the frame interval information for the current speed to which the OTG\_FS/OTG\_HS controller has enumerated.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RLD CTRL
																rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
FRIVL[15:0]																
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 **RLDCTRL**: Reload control

This bit allows dynamic reloading of the HFIR register during run time.

0: The HFIR cannot be reloaded dynamically

1: The HFIR can be dynamically reloaded during run time.

This bit needs to be programmed during initial configuration and its value must not be changed during run time.

**Caution:** RLDCTRL = 0 is not recommended.

Bits 15:0 **FRIVL[15:0]**: Frame interval for USB OTG FS

The value that the application programs to this field, specifies the interval between two consecutive SOFs (FS) or Keep-Alive tokens (LS). This field contains the number of PHY clocks that constitute the required frame interval. The application can write a value to this register only after the port enable bit of the host port control and status register (PENA bit in OTG\_HPRT) has been set. If no value is programmed, the core calculates the value based on the PHY clock specified in the FS/LS PHY clock select field of the host configuration register (FSLSPCS in OTG\_HCFG). Do not change the value of this field after the initial configuration, unless the RLDCTRL bit is set. In such case, the FRIVL is reloaded with each SOF event.

– Frame interval = 1 ms × (FRIVL - 1)

Bits 15:0 **FRIVL[15:0]**: Frame interval for USB OTG HS

The value that the application programs to this field, specifies the interval between two consecutive micro-SOFs (HS) or Keep-Alive tokens (LS). This field contains the number of PHY clocks that constitute the required frame interval. The application can write a value to this register only after the port enable bit of the host port control and status register (PENA bit in OTG\_HPRT) has been set. If no value is programmed, the core calculates the value based on the PHY clock specified in the FS/LS PHY clock select field of the host configuration register (FSLSPCS in OTG\_HCFG). Do not change the value of this field after the initial configuration, unless the RLDCTRL bit is set. In such case, the FRIVL is reloaded with each SOF event.

– Frame interval = 125 µs × (FRIVL - 1) in high speed operation

– Frame interval = 1 ms × (FRIVL - 1) in low/full speed operation

### 37.15.23 OTG host frame number/frame time remaining register (OTG\_HFNUM)

Address offset: 0x408

Reset value: 0x0000 3FFF

This register indicates the current frame number. It also indicates the time remaining (in terms of the number of PHY clocks) in the current frame.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FTREM[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FRNUM[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **FTREM[15:0]**: Frame time remaining

Indicates the amount of time remaining in the current frame, in terms of PHY clocks. This field decrements on each PHY clock. When it reaches zero, this field is reloaded with the value in the Frame interval register and a new SOF is transmitted on the USB.

Bits 15:0 **FRNUM[15:0]**: Frame number

This field increments when a new SOF is transmitted on the USB, and is cleared to 0 when it reaches 0x3FFF.

### 37.15.24 OTG\_Host periodic transmit FIFO/queue status register (OTG\_HPTXSTS)

Address offset: 0x410

Reset value: 0x0008 0100

This read-only register contains the free space information for the periodic Tx FIFO and the periodic transmit request queue.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PTXQTOP[7:0]								PTXQSAV[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PTXFSAVL[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **PTXQTOP[7:0]**: Top of the periodic transmit request queue

This indicates the entry in the periodic Tx request queue that is currently being processed by the MAC.

This register is used for debugging.

Bit 31: Odd/Even frame

0: send in even frame

1: send in odd frame

Bits 30:27: Channel/endpoint number

Bits 26:25: Type

00: IN/OUT

01: Zero-length packet

11: Disable channel command

Bit 24: Terminate (last entry for the selected channel/endpoint)

Bits 23:16 **PTXQSAV[7:0]**: Periodic transmit request queue space available

Indicates the number of free locations available to be written in the periodic transmit request queue. This queue holds both IN and OUT requests.

00: Periodic transmit request queue is full

01: 1 location available

10: 2 locations available

b<sub>n</sub>: n locations available (0 ≤ n ≤ 8)

Others: Reserved

Bits 15:0 **PTXFSAVL[15:0]**: Periodic transmit data FIFO space available

Indicates the number of free locations available to be written to in the periodic Tx FIFO.

Values are in terms of 32-bit words

0000: Periodic Tx FIFO is full

0001: 1 word available

0010: 2 words available

b<sub>n</sub>: n words available (where 0 ≤ n ≤ PTXFD)

Others: Reserved

### 37.15.25 OTG host all channels interrupt register (OTG\_HAIT)

Address offset: 0x414

Reset value: 0x0000 0000

When a significant event occurs on a channel, the host all channels interrupt register interrupts the application using the host channels interrupt bit of the core interrupt register (HCINT bit in OTG\_GINTSTS). This is shown in [Figure 463](#). There is one interrupt bit per channel, up to a maximum of 16 bits. Bits in this register are set and cleared when the application sets and clears bits in the corresponding host channel-x interrupt register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
HAIN[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **HAINT[15:0]**: Channel interrupts

One bit per channel: Bit 0 for Channel 0, bit 15 for Channel 15

### 37.15.26 OTG host all channels interrupt mask register (OTG\_HAINTMSK)

Address offset: 0x418

Reset value: 0x0000 0000

The host all channel interrupt mask register works with the host all channel interrupt register to interrupt the application when an event occurs on a channel. There is one interrupt mask bit per channel, up to a maximum of 16 bits.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HAINTM[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **HAINTM[15:0]**: Channel interrupt mask

0: Masked interrupt

1: Unmasked interrupt

One bit per channel: Bit 0 for channel 0, bit 15 for channel 15

### 37.15.27 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 463](#). The rc\_w1 bits in this register can trigger an interrupt to the application through the host port interrupt bit of the core interrupt register (HPRTINT bit in OTG\_GINTSTS). On a port interrupt, the application must read this register and clear the bit that caused the interrupt. For the rc\_w1 bits, the application must write a 1 to the bit to clear the interrupt.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PSPD[1:0]	PTCTL [3]	
													r	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PTCTL[2:0]	PPWR	PLSTS[1:0]	Res.	PRST	PSUSP	PRES	POC CHNG	POCA	PEN CHNG	PENA	PCDET	PCSTS			
rw	rw	rw	rw	r	r	rw	rs	rw	rc_w1	r	rc_w1	rc_w1	rc_w1	rc_w1	r

Bits 31:19 Reserved, must be kept at reset value.

Bits 18:17 **PSPD[1:0]: Port speed**

Indicates the speed of the device attached to this port.

- 01: Full speed
- 10: Low speed
- 11: Reserved
- 00: High speed

Bits 16:13 **PTCTL[3:0]: Port test control**

The application writes a nonzero value to this field to put the port into a Test mode, and the corresponding pattern is signaled on the port.

- 0000: Test mode disabled
- 0001: Test\_J mode
- 0010: Test\_K mode
- 0011: Test\_SE0\_NAK mode
- 0100: Test\_Packet mode
- 0101: Test\_Force\_Enable
- Others: Reserved

Bit 12 **PPWR: Port power**

The application uses this field to control power to this port, and the core clears this bit on an overcurrent condition.

- 0: Power off
- 1: Power on

Bits 11:10 **PLSTS[1:0]: Port line status**

Indicates the current logic level USB data lines

- Bit 10: Logic level of OTG\_DP
- Bit 11: Logic level of OTG\_DM

Bit 9 Reserved, must be kept at reset value.

**Bit 8 PRST: Port reset**

When the application sets this bit, a reset sequence is started on this port. The application must time the reset period and clear this bit after the reset sequence is complete.

0: Port not in reset

1: Port in reset

The application must leave this bit set for a minimum duration of at least 10 ms to start a reset on the port. The application can leave it set for another 10 ms in addition to the required minimum duration, before clearing the bit, even though there is no maximum limit set by the USB standard.

High speed: 50 ms

Full speed/Low speed: 10 ms

**Bit 7 PSUSP: Port suspend**

The application sets this bit to put this port in suspend mode. The core only stops sending SOFs when this is set. To stop the PHY clock, the application must set the port clock stop bit, which asserts the suspend input pin of the PHY.

The read value of this bit reflects the current suspend status of the port. This bit is cleared by the core after a remote wake-up signal is detected or the application sets the port reset bit or port resume bit in this register or the resume/remote wake-up detected interrupt bit or disconnect detected interrupt bit in the core interrupt register (WKUPINT or DISCINT in OTG\_GINTSTS, respectively).

0: Port not in suspend mode

1: Port in suspend mode

**Bit 6 PRES: Port resume**

The application sets this bit to drive resume signaling on the port. The core continues to drive the resume signal until the application clears this bit.

If the core detects a USB remote wake-up sequence, as indicated by the port resume/remote wake-up detected interrupt bit of the core interrupt register (WKUPINT bit in OTG\_GINTSTS), the core starts driving resume signaling without application intervention and clears this bit when it detects a disconnect condition. The read value of this bit indicates whether the core is currently driving resume signaling.

0: No resume driven

1: Resume driven

When LPM is enabled and the core is in L1 state, the behavior of this bit is as follow:

1. The application sets this bit to drive resume signaling on the port.
2. The core continues to drive the resume signal until a predetermined time specified in BESLTHRS[3:0] field of OTG\_GLPMCFG register.
3. If the core detects a USB remote wake-up sequence, as indicated by the port L1Resume/Remote L1WakeUp detected interrupt bit of the core interrupt register (WKUPINT in OTG\_GINTSTS), the core starts driving resume signaling without application intervention and clears this bit at the end of resume. This bit can be set or cleared by both the core and the application. This bit is cleared by the core even if there is no device connected to the host.

**Bit 5 POCCHNG: Port overcurrent change**

The core sets this bit when the status of the port overcurrent active bit (bit 4) in this register changes.

**Bit 4 POCA: Port overcurrent active**

Indicates the overcurrent condition of the port.

0: No overcurrent condition

1: Overcurrent condition

**Bit 3 PENCHNG: Port enable/disable change**

The core sets this bit when the status of the port enable bit 2 in this register changes.

Bit 2 **PENA:** Port enable

A port is enabled only by the core after a reset sequence, and is disabled by an overcurrent condition, a disconnect condition, or by the application clearing this bit. The application cannot set this bit by a register write. It can only clear it to disable the port. This bit does not trigger any interrupt to the application.

0: Port disabled

1: Port enabled

Bit 1 **PCDET:** Port connect detected

The core sets this bit when a device connection is detected to trigger an interrupt to the application using the host port interrupt bit in the core interrupt register (HPRTINT bit in OTG\_GINTSTS). The application must write a 1 to this bit to clear the interrupt.

Bit 0 **PCSTS:** Port connect status

0: No device is attached to the port

1: A device is attached to the port

**37.15.28 OTG host channel x characteristics register (OTG\_HCCHARx)**

Address offset: 0x500 + 0x20 \* x, (x = 0 to 15[HS] / 11[FS])

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CHENA	CHDIS	ODD FRM	DAD[6:0]						MCNT[1:0]		EPTYP[1:0]		LSDEV	Res.	
rs	rs	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EPDIR	EPNUM[3:0]					MPSIZ[10:0]									
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **CHENA:** Channel enable

This field is set by the application and cleared by the OTG host.

0: Channel disabled

1: Channel enabled

Bit 30 **CHDIS:** Channel disable

The application sets this bit to stop transmitting/receiving data on a channel, even before the transfer for that channel is complete. The application must wait for the Channel disabled interrupt before treating the channel as disabled.

Bit 29 **ODDFRM:** Odd frame

This field is set (reset) by the application to indicate that the OTG host must perform a transfer in an odd frame. This field is applicable for only periodic (isochronous and interrupt) transactions.

0: Even frame

1: Odd frame

Bits 28:22 **DAD[6:0]:** Device address

This field selects the specific device serving as the data source or sink.

Bits 21:20 **MCNT[1:0]: Multicount**

This field indicates to the host the number of transactions that must be executed per frame for this periodic endpoint. For non-periodic transfers, this field is not used

00: Reserved. This field yields undefined results

01: 1 transaction

10: 2 transactions per frame to be issued for this endpoint

11: 3 transactions per frame to be issued for this endpoint

*Note: This field must be set to at least 01.*

Bits 19:18 **EPTYP[1:0]: Endpoint type**

Indicates the transfer type selected.

00: Control

01: Isochronous

10: Bulk

11: Interrupt

Bit 17 **LSDEV: Low-speed device**

This field is set by the application to indicate that this channel is communicating to a low-speed device.

Bit 16 Reserved, must be kept at reset value.

Bit 15 **EPDIR: Endpoint direction**

Indicates whether the transaction is IN or OUT.

0: OUT

1: IN

Bits 14:11 **EPNUM[3:0]: Endpoint number**

Indicates the endpoint number on the device serving as the data source or sink.

Bits 10:0 **MPSIZ[10:0]: Maximum packet size**

Indicates the maximum packet size of the associated endpoint.

**37.15.29 OTG host channel x split control register (OTG\_HCSPLTx)**

Address offset: 0x504 + 0x20 \* x, (x = 0 to 15)

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	COMP LSPLT
SPLIT EN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	rw	
rw															rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
XACTPOS[1:0]		HUBADDR[6:0]								PRTADDR[6:0]						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 31 **SPLITEN**: Split enable

The application sets this bit to indicate that this channel is enabled to perform split transactions.

Bits 30:17 Reserved, must be kept at reset value.

Bit 16 **COMPLSPLT**: Do complete split

The application sets this bit to request the OTG host to perform a complete split transaction.

Bits 15:14 **XACTPOS[1:0]**: Transaction position

This field is used to determine whether to send all, first, middle, or last payloads with each OUT transaction.

11: All. This is the entire data payload of this transaction (which is less than or equal to 188 bytes)

10: Begin. This is the first data payload of this transaction (which is larger than 188 bytes)

00: Mid. This is the middle payload of this transaction (which is larger than 188 bytes)

01: End. This is the last payload of this transaction (which is larger than 188 bytes)

Bits 13:7 **HUBADDR[6:0]**: Hub address

This field holds the device address of the transaction translator's hub.

Bits 6:0 **PRTADDR[6:0]**: Port address

This field is the port number of the recipient transaction translator.

### 37.15.30 OTG host channel x interrupt register (OTG\_HCINTx)

Address offset:  $0x508 + 0x20 * x$ , ( $x = 0$  to  $15$ [HS] /  $11$ [FS])

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 463](#). The application must read this register when the host channels interrupt bit in the core interrupt register (HCINT bit in OTG\_GINTSTS) is set. Before the application can read this register, it must first read the host all channels interrupt (OTG\_HAINT) register to get the exact channel number for the host channel-x interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the OTG\_HAINT and OTG\_GINTSTS registers.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DTERR	FRM OR	BBERR	TXERR	Res.	ACK	NAK	STALL	Res.	CHH	XFRC
				rc_w1	rc_w1	rc_w1	rc_w1	rc_w1		rc_w1	rc_w1	rc_w1		rc_w1	rc_w1

Note: Configuration register for USB OTG FS.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DTERR	FRM OR	BBERR	TXERR	NYET	ACK	NAK	STALL	AHB ERR	CHH	XFRC
					rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Note: Configuration register for USB OTG HS.

Bits 31:11 Reserved, must be kept at reset value.

Bit 10 **DTERR**: Data toggle error.

Bit 9 **FRMOR**: Frame overrun.

Bit 8 **BBERR**: Babble error.

Bit 7 **TXERR**: Transaction error.

Indicates one of the following errors occurred on the USB.

CRC check failure

Timeout

Bit stuff error

False EOP

Bit 6 Reserved, must be kept at reset value for USB OTG FS.

Bit 6 **NYET**: Not yet ready response received interrupt for USB OTG HS.

Bit 5 **ACK**: ACK response received/transmitted interrupt.

Bit 4 **NAK**: NAK response received interrupt.

Bit 3 **STALL**: STALL response received interrupt.

Bit 2 Reserved, must be kept at reset value for USB OTG FS.

Bit 2 **AHBERR**: AHB error for USB OTG HS

This error is generated only in Internal DMA mode when an AHB error occurs during an AHB read/write operation. The application can read the corresponding DMA channel address register to get the error address.

Bit 1 **CHH**: Channel halted

This indicates that the transfer completed:

- because of any USB transaction error
- in response to disable request by the application
- normally

Bit 0 **XFRC**: Transfer completed.

Transfer completed normally without any errors.

### 37.15.31 OTG host channel x interrupt mask register (OTG\_HCINTMSKx)

Address offset: 0x50C + 0x20 \* x, (x = 0 to 15[HS] / 11[FS])

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.	DTERR M	FRM ORM	BBERR M	TXERR M	Res.	ACKM	NAKM	STALL M	Res.	CHHM	XFRC M
					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.	DTERR M	FRM ORM	BBERR M	TXERR M	NYET	ACKM	NAKM	STALL M	AHB ERRM	CHHM	XFRC M
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Note: Configuration register for USB OTG HS

Bits 31:11 Reserved, must be kept at reset value.

Bit 10 **DTERRM**: Data toggle error mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 9 **FRMORM**: Frame overrun mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 8 **BBERRM**: Babble error mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 **TXERRM**: Transaction error mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 6 Reserved, must be kept at reset value for USB OTG FS.

Bit 6 **NYET**: response received interrupt mask for USB OTG HS.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 5 **ACKM**: ACK response received/transmitted interrupt mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 4 **NAKM:** NAK response received interrupt mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 3 **STALLM:** STALL response received interrupt mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 2 **AHBERRM:** AHB error for USB OTG HS.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 2 Reserved, must be kept at reset value for USB OTG FS.

Bit 1 **CHHM:** Channel halted mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 0 **XFRCM:** Transfer completed mask

- 0: Masked interrupt
- 1: Unmasked interrupt

### 37.15.32 OTG host channel x transfer size register (OTG\_HCTSIZx)

Address offset: 0x510 + 0x20 \* x, (x = 0 to 15[HS] / 11[FS])

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DO PNG	DPID[1:0]		PKTCNT[9:0]												XFRSIZ[18:16]
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XFRSIZ[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **DOPNG:** Do Ping

This bit is used only for OUT transfers. Setting this field to 1 directs the host to do PING protocol.

*Note: Do not set this bit for IN transfers. If this bit is set for IN transfers, it disables the channel.*

Bits 30:29 **DPID[1:0]:** Data PID

The application programs this field with the type of PID to use for the initial transaction. The host maintains this field for the rest of the transfer.

00: DATA0

01: DATA2

10: DATA1

11: SETUP (control) / reserved[FS]MDATA[HS] (non-control)

Bits 28:19 **PKTCNT[9:0]:** Packet count

This field is programmed by the application with the expected number of packets to be transmitted (OUT) or received (IN).

The host decrements this count on every successful transmission or reception of an OUT/IN packet. Once this count reaches zero, the application is interrupted to indicate normal completion.

Bits 18:0 **XFRSIZ[18:0]:** Transfer size

For an OUT, this field is the number of data bytes the host sends during the transfer.

For an IN, this field is the buffer size that the application has reserved for the transfer. The application is expected to program this field as an integer multiple of the maximum packet size for IN transactions (periodic and non-periodic).

**37.15.33 OTG host channel x DMA address register (OTG\_HCDMAx)**

Address offset: 0x514 + 0x20 \* x, (x = 0 to 15)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DMAADDR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAADDR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DMAADDR[31:0]:** DMA address

This field holds the start address in the external memory from which the data for the endpoint must be fetched or to which it must be stored. This register is incremented on every AHB transaction.

### 37.15.34 Device-mode registers

These registers must be programmed every time the core changes to device mode

### 37.15.35 OTG device configuration register (OTG\_DCFG)

Address offset: 0x800

Reset value: 0x0220 0000

This register configures the core in device mode after power-on or after certain control commands or enumeration. Do not make changes to this register after initial programming.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRAT IM	Res.	Res.	PFIVL[1:0]					DAD[6:0]				Res.	NZLSO HSK		DSPD[1:0]
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	PERSCHIVL[1:0]	Res.	Res.	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
ERRAT IM	XCVR DLY	Res.	PFIVL[1:0]					DAD[6:0]				Res.	NZLSO HSK		DSPD[1:0]
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw

Note: Configuration register for USB OTG HS

Bits 31:16 Reserved, must be kept at reset value for USB OTG FS.

Bits 31:26 Reserved, must be kept at reset value for USB OTG HS.

Bits 25:24 PERSCHIVL[1:0]: Periodic schedule interval for USB OTG HS

This field specifies the amount of time the Internal DMA engine must allocate for fetching periodic IN endpoint data. Based on the number of periodic endpoints, this value must be specified as 25, 50 or 75% of the (micro) frame.

- When any periodic endpoints are active, the internal DMA engine allocates the specified amount of time in fetching periodic IN endpoint data
- When no periodic endpoint is active, then the internal DMA engine services nonperiodic endpoints, ignoring this field
- After the specified time within a (micro) frame, the DMA switches to fetching nonperiodic endpoints

00: 25% of (micro)frame

01: 50% of (micro)frame

10: 75% of (micro)frame

11: Reserved

Bits 23:16 Reserved, must be kept at reset value for USB OTG HS.

Bit 15 **ERRATIM**: Erratic error interrupt mask

- 1: Mask early suspend interrupt on erratic error
- 0: Early suspend interrupt is generated on erratic error

Bit 14 **XCVRDLY**: Transceiver delay

Enables or disables delay in ULPI timing during device chirp.

0: Disable delay (use default timing)

1: Enable delay to default timing, necessary for some ULPI PHYs

Bit 13 Reserved, must be kept at reset value.

Bits 12:11 **PFIVL[1:0]**: Periodic frame interval

Indicates the time within a frame at which the application must be notified using the end of periodic frame interrupt. This can be used to determine if all the isochronous traffic for that frame is complete.

00: 80% of the frame interval

01: 85% of the frame interval

10: 90% of the frame interval

11: 95% of the frame interval

Bits 10:4 **DAD[6:0]**: Device address

The application must program this field after every SetAddress control command.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **NZLSOHSK**: Non-zero-length status OUT handshake

The application can use this field to select the handshake the core sends on receiving a nonzero-length data packet during the OUT transaction of a control transfer's status stage.

1: Send a STALL handshake on a nonzero-length status OUT transaction and do not send the received OUT packet to the application.

0: Send the received OUT packet to the application (zero-length or nonzero-length) and send a handshake based on the NAK and STALL bits for the endpoint in the device endpoint control register.

Bits 1:0 **DSPD[1:0]**: Device speed

Indicates the speed at which the application requires the core to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the core is connected.

00: Reserved

01: Reserved

10: Reserved

11: Full speed (USB 1.1 transceiver clock is 48 MHz)

Bits 1:0 **DSPD[1:0]**: Device speed

Indicates the speed at which the application requires the core to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the core is connected.

00: High speed

01: Full speed using HS

10: Reserved

11: Full speed using internal FS PHY

### 37.15.36 OTG device control register (OTG\_DCTL)

Address offset: 0x804

Reset value: 0x0000 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DS BESL RJCT	Res.	Res.
													rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	PO PRGDNE	CGO NAK	SGO NAK	CGI NAK	SGI NAK	TCTL[2:0]	GON STS	GIN STS	SDIS	RWU SIG		
				rw	w	w	w	w	rw	rw	rw	r	rw	rw	rw

Bits 31:19 Reserved, must be kept at reset value.

Bit 18 **DSBESLRJCT**: Deep sleep BESL reject

Core rejects LPM request with BESL value greater than BESL threshold programmed. NYET response is sent for LPM tokens with BESL value greater than BESL threshold. By default, the deep sleep BESL reject feature is disabled.

Bits 17:12 Reserved, must be kept at reset value.

Bit 11 **POPRGDNE**: Power-on programming done

The application uses this bit to indicate that register programming is completed after a wake-up from power down mode.

Bit 10 **CGONAK**: Clear global OUT NAK

Writing 1 to this field clears the Global OUT NAK.

Bit 9 **SGONAK**: Set global OUT NAK

Writing 1 to this field sets the Global OUT NAK.

The application uses this bit to send a NAK handshake on all OUT endpoints.

The application must set this bit only after making sure that the Global OUT NAK effective bit in the core interrupt register (GONAKEFF bit in OTG\_GINTSTS) is cleared.

Bit 8 **CGINAK**: Clear global IN NAK

Writing 1 to this field clears the Global IN NAK.

Bit 7 **SGINAK**: Set global IN NAK

Writing 1 to this field sets the Global non-periodic IN NAK. The application uses this bit to send a NAK handshake on all non-periodic IN endpoints.

The application must set this bit only after making sure that the Global IN NAK effective bit in the core interrupt register (GINAKEFF bit in OTG\_GINTSTS) is cleared.

Bits 6:4 **TCTL[2:0]**: Test control

000: Test mode disabled

001: Test\_J mode

010: Test\_K mode

011: Test\_SE0\_NAK mode

100: Test\_Packet mode

101: Test\_Force\_Enable

Others: Reserved

Bit 3 **GONSTS**: Global OUT NAK status

0:A handshake is sent based on the FIFO status and the NAK and STALL bit settings.  
 1:No data is written to the Rx FIFO, irrespective of space availability. Sends a NAK handshake on all packets, except on SETUP transactions. All isochronous OUT packets are dropped.

Bit 2 **GINSTS**: Global IN NAK status

0:A handshake is sent out based on the data availability in the transmit FIFO.  
 1:A NAK handshake is sent out on all non-periodic IN endpoints, irrespective of the data availability in the transmit FIFO.

Bit 1 **SDIS**: Soft disconnect

The application uses this bit to signal the USB OTG core to perform a soft disconnect. As long as this bit is set, the host does not see that the device is connected, and the device does not receive signals on the USB. The core stays in the disconnected state until the application clears this bit.

0:Normal operation. When this bit is cleared after a soft disconnect, the core generates a device connect event to the USB host. When the device is reconnected, the USB host restarts device enumeration.

1:The core generates a device disconnect event to the USB host.

Bit 0 **RWUSIG**: Remote wake-up 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 275** 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 275. 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

### 37.15.37 OTG device status register (OTG\_DSTS)

Address offset: 0x808

Reset value: 0x0000 0010

This register indicates the status of the core with respect to USB-related events. It must be read on interrupts from the device all interrupts (OTG\_DAINT) register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DEVLNSTS[1:0]		FNSOF[13:8]					
								r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FNSOF[7:0]								Res.	Res.	Res.	Res.	EERR	ENUMSPD[1:0]	SUSP STS	
r	r	r	r	r	r	r	r					r	r	r	r

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:22 **DEVLNSTS[1:0]:** Device line status

Indicates the current logic level USB data lines.

Bit [23]: Logic level of D+

Bit [22]: Logic level of D-

Bits 21:8 **FNSOF[13:0]:** Frame number of the received SOF

Bits 7:4 Reserved, must be kept at reset value.

Bit 3 **EERR:** Erratic error

The core sets this bit to report any erratic errors.

Due to erratic errors, the OTG\_FS/OTG\_HS controller goes into suspended state and an interrupt is generated to the application with Early suspend bit of the OTG\_GINTSTS register (ESUSP bit in OTG\_GINTSTS). If the early suspend is asserted due to an erratic error, the application can only perform a soft disconnect recover.

Bits 2:1 **ENUMSPD[1:0]:** Enumerated speed

Indicates the speed at which the OTG\_FS/OTG\_HS controller has come up after speed detection through a chirp sequence.

00: High Speed using HS PHY

01: Full Speed using HS PHY

11: Full speed using embedded FS PHY

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 wake-up signaling bit in the OTG\_DCTL register (RWUSIG bit in OTG\_DCTL).

### 37.15.38 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.	INEPN EM	INEPN MM	ITTXFE MSK	TOM	Res.	EPDM	XFRC 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.	Res.	TXFURM	Res.	INEPN EM	INEPN MM	ITTXFE MSK	TOM	AHB ERRM	EPDM	XFRC 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 12:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

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.

Bit 6 **INEPNEM:** IN endpoint NAK effective mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 5 **INEPNMM:** IN token received with EP mismatch mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 4 **ITTXFEMSK**: IN token received when Tx FIFO empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 3 **TOM**: Timeout condition mask (Non-isochronous endpoints)

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 2 **AHBERRM**: AHB error mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 1 **EPDM**: Endpoint disabled interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 0 **XFRCM**: Transfer completed interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

### 37.15.39 OTG device OUT endpoint common interrupt mask register (OTG\_DOEPMASK)

Address offset: 0x814

Reset value: 0x0000 0000

This register works with each of the OTG\_DOEPINTx registers for all endpoints to generate an interrupt per OUT endpoint. The OUT endpoint interrupt for a specific status in the OTG\_DOEPINTx register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAK MSK	BERR M	Res.	Res.	Res.	OUT PKT ERRM	Res.	Res.	STS PHSR XM	OTEPD M	STUPM	Res.	EPDM	XFR M
		rw	rw				rw			rw	rw	rw		rw	rw

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	NYET MSK	NAK MSK	BERR M	Res.	Res.	Res.	OUT PKT ERRM	Res.	B2B STUPM	STS PHSR XM	OTEPD M	STUPM	AHB ERRM	EPDM	XFR M
	rw	rw	rw				rw		rw	rw	rw	rw	rw	rw	rw

Note: Configuration register for USB OTG HS

Bits 31:15 Reserved, must be kept at reset value.

Bit 14 **NYETMSK**: NYET interrupt mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 13 **NAKMSK**: NAK interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 12 **BERRM**: Babble error interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bits 11:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **OUTPKTERRM**: Out packet error mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 Reserved, must be kept at reset value.

Bit 6 **B2BSTUPM**: Back-to-back SETUP packets received mask for USB OTG HS

- Applies to control OUT endpoints only.
- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 5 **STSPHSRXM**: Status phase received for control write mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 4 **OTEPDM**: OUT token received when endpoint disabled mask. Applies to control OUT endpoints only.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 3 **STUPM**: STUPM: SETUP phase done mask. Applies to control endpoints only.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 2 **AHBERRM**: AHB error mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 1 **EPDM**: Endpoint disabled interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 0 **XFRCM**: Transfer completed interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

### 37.15.40 OTG device all endpoints interrupt register (OTG\_DAINT)

Address offset: 0x818

Reset value: 0x0000 0000

When a significant event occurs on an endpoint, a OTG\_DAINT register interrupts the application using the device OUT endpoints interrupt bit or device IN endpoints interrupt bit of the OTG\_GINTSTS register (OEPINT or IEPINT in OTG\_GINTSTS, respectively). There is one interrupt bit per endpoint, up to a maximum of 16 bits for OUT endpoints and 16 bits for IN endpoints. For a bidirectional endpoint, the corresponding IN and OUT interrupt bits are used. Bits in this register are set and cleared when the application sets and clears bits in the corresponding device endpoint-x interrupt register (OTG\_DIEPINTx/OTG\_DOEPINTx).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OEPINT[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IEPINT[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **OEPINT[15:0]**: OUT endpoint interrupt bits

One bit per OUT endpoint:

Bit 16 for OUT endpoint 0, bit 19 for OUT endpoint 3.

Bits 15:0 **IEPINT[15:0]**: IN endpoint interrupt bits

One bit per IN endpoint:

Bit 0 for IN endpoint 0, bit 3 for endpoint 3.

### 37.15.41 OTG all endpoints interrupt mask register (OTG\_DAINTMSK)

Address offset: 0x81C

Reset value: 0x0000 0000

The OTG\_DAINTMSK register works with the device endpoint interrupt register to interrupt the application when an event occurs on a device endpoint. However, the OTG\_DAINT register bit corresponding to that interrupt is still set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OEPM[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IEPM[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **OEPM[15:0]**: OUT EP interrupt mask bits

One per OUT endpoint:

Bit 16 for OUT EP 0, bit 19 for OUT EP 3

0: Masked interrupt

1: Unmasked interrupt

Bits 15:0 **IEPM[15:0]**: IN EP interrupt mask bits

One bit per IN endpoint:

Bit 0 for IN EP 0, bit 3 for IN EP 3

0: Masked interrupt

1: Unmasked interrupt

### 37.15.42 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.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VBUSDT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **VBUSDT[15:0]**: Device V<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 V<sub>BUS</sub> load, this value may need adjusting.

### 37.15.43 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.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DVBUSP[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **DVBUSP[15:0]**: Device V<sub>BUS</sub> pulsing time. This feature is only relevant to OTG1.3.

Specifies the V<sub>BUS</sub> pulsing time during SRP. This value equals:  
V<sub>BUS</sub> pulsing time in PHY clocks / 1 024

### 37.15.44 OTG device threshold control register (OTG\_DTHRCTL)

Address offset: 0x0830

Reset value: 0x0000 0000

**Note:** Configuration register applies only to USB OTG HS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
Res.	Res.	Res.	Res.	ARPEN	Res.	RXTHRLEN[8:0]												RXTHREN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Res.	Res.	Res.	Res.	Res.	TXTHRLEN[8:0]												ISOTHREN	NONISOTHREN
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:28 Reserved, must be kept at reset value.

Bit 27 **ARPEN**: Arbiter parking enable

This bit controls internal DMA arbiter parking for IN endpoints. When thresholding is enabled and this bit is set to one, then the arbiter parks on the IN endpoint for which there is a token received on the USB. This is done to avoid getting into underrun conditions. By default parking is enabled.

Bit 26 Reserved, must be kept at reset value.

Bits 25:17 **RXTHRLEN[8:0]**: Receive threshold length

This field specifies the receive thresholding size in 32-bit words. This field also specifies the amount of data received on the USB before the core can start transmitting on the AHB. The threshold length has to be at least eight 32-bit words. The recommended value for RXTHRLEN is to be the same as the programmed AHB burst length (HBSTLEN bit in OTG\_GAHBCFG).

Bit 16 **RXTHREN**: Receive threshold enable

When this bit is set, the core enables thresholding in the receive direction.

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:2 **TXTHRLEN[8:0]**: Transmit threshold length

This field specifies the transmit thresholding size in 32-bit words. This field specifies the amount of data in bytes to be in the corresponding endpoint transmit FIFO, before the core can start transmitting on the USB. The threshold length has to be at least eight 32-bit words. This field controls both isochronous and nonisochronous IN endpoint thresholds. The recommended value for TXTHRLEN is to be the same as the programmed AHB burst length (HBSTLEN bit in OTG\_GAHBCFG).

Bit 1 **ISOTHREN**: ISO IN endpoint threshold enable

When this bit is set, the core enables thresholding for isochronous IN endpoints.

Bit 0 **NONISOTHREN**: Nonisochronous IN endpoints threshold enable

When this bit is set, the core enables thresholding for nonisochronous IN endpoints.

### 37.15.45 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[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **INEPTXFEM[15:0]**: IN EP Tx FIFO empty interrupt mask bits

These bits act as mask bits for OTG\_DIEPINTx.

TXFE interrupt one bit per IN endpoint:

Bit 0 for IN endpoint 0, bit 3 for IN endpoint 3

0: Masked interrupt

1: Unmasked interrupt

### 37.15.46 OTG device each endpoint interrupt register (OTG\_DEACHINT)

Address offset: 0x0838

Reset value: 0x0000 0000

*Note:* Configuration register applies only to USB OTG HS.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OEP1 INT	Res.													
														r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	IEP1 INT	Res.													
														r	

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **OEP1INT:** OUT endpoint 1 interrupt bit

Bits 16:2 Reserved, must be kept at reset value.

Bit 1 **IEP1INT:** IN endpoint 1 interrupt bit

Bit 0 Reserved, must be kept at reset value.

### 37.15.47 OTG device each endpoint interrupt mask register (OTG\_DEACHINTMSK)

Address offset: 0x083C

Reset value: 0x0000 0000

There is one interrupt bit for endpoint 1 IN and one interrupt bit for endpoint 1 OUT.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OEP1 INTM	Res.													
														rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	IEP1 INTM	Res.													
														rw	

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **OEP1INTM:** OUT endpoint 1 interrupt mask bit

Bits 16:2 Reserved, must be kept at reset value.

Bit 1 **IEP1INTM:** IN endpoint 1 interrupt mask bit

Bit 0 Reserved, must be kept at reset value.

### 37.15.48 OTG device each IN endpoint-1 interrupt mask register (OTG\_HS\_DIEPEACHMSK1)

Address offset: 0x844

Reset value: 0x0000 0000

This register works with the OTG\_DIEPINT1 register to generate a dedicated interrupt OTG\_HS\_EP1\_IN for endpoint #1. The IN endpoint interrupt for a specific status in the OTG\_DOEPINT1 register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAKM	Res.	Res.	Res.	Res.	TXFURM	Res.	INEPNEM	Res.	ITTXFEMSK	TOM	AHBERRM	EPDM	XFRCM
		rw					rw		rw		rw	rw	rw	rw	rw

Bits 31:14 Reserved, must be kept at reset value.

Bit 13 **NAKM:** NAK interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bits 12:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **TXFURM:** FIFO underrun mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 Reserved, must be kept at reset value.

Bit 6 **INEPNEM:** IN endpoint NAK effective mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 5 Reserved, must be kept at reset value.

Bit 4 **ITTXFEMSK:** IN token received when Tx FIFO empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 3 **TOM:** Timeout condition mask (Non-isochronous endpoints)

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 2 **AHBERRM**: AHB error mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 1 **EPDM**: Endpoint disabled interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 0 **XFRCM**: Transfer completed interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

### 37.15.49 OTG device each OUT endpoint-1 interrupt mask register (OTG\_HS\_DOEPEACHMSK1)

Address offset: 0x884

Reset value: 0x0000 0000

This register works with the OTG\_DOEPINT1 register to generate a dedicated interrupt OTG\_HS\_EP1\_OUT for endpoint #1. The OUT endpoint interrupt for a specific status in the OTG\_DOEPINT1 register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	NYET MSK	NAK MSK	BERR M	Res.	Res.	Res.	OUT PKT ERRM	Res.	B2B STUPM	Res.	OTEPD M	STUPM	AHB ERRM	EPDM	XFR M
rw	rw	rw					rw		rw		rw	rw	rw	rw	rw

Bits 31:15 Reserved, must be kept at reset value.

Bit 14 **NYETMSK**: NYET interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 13 **NAKMSK**: NAK interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 12 **BERRM**: Babble error interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bits 11:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **OUTPKTERRM**: Out packet error mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 Reserved, must be kept at reset value.

Bit 6 **B2BSTUPM:** Back-to-back SETUP packets received mask

Applies to control OUT endpoints only.

0: Masked interrupt

1: Unmasked interrupt

Bit 5 Reserved, must be kept at reset value.

Bit 4 **OTEPDM:** OUT token received when endpoint disabled mask

Applies to control OUT endpoints only.

0: Masked interrupt

1: Unmasked interrupt

Bit 3 **STUPM:** STUPM: SETUP phase done mask

Applies to control endpoints only.

0: Masked interrupt

1: Unmasked interrupt

Bit 2 **AHBERRM:** AHB error mask

0: Masked interrupt

1: Unmasked interrupt

Bit 1 **EPDM:** Endpoint disabled interrupt mask

0: Masked interrupt

1: Unmasked interrupt

Bit 0 **XFRCM:** Transfer completed interrupt mask

0: Masked interrupt

1: Unmasked interrupt

### 37.15.50 OTG device control IN endpoint 0 control register (OTG\_DIEPCTL0)

Address offset: 0x900

Reset value: 0x0000 0000

This section describes the OTG\_DIEPCTL0 register for USB\_OTG FS. Nonzero control endpoints use registers for endpoints 1–3.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPENA	EPDIS	Res.	Res.	SNAK	CNAK	TXFNUM[3:0]				STALL	Res.	EPTYP[1:0]		NAK STS	Res.
rs	rs			w	w	rw	rw	rw	rw	rs		r	r	r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBA EP	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MPSIZ[1:0]	
r														rw	rw

**Bit 31 EPENA:** Endpoint enable

The application sets this bit to start transmitting data on the endpoint 0.

The core clears this bit before setting any of the following interrupts on this endpoint:

- Endpoint disabled
- Transfer completed

**Bit 30 EPDIS:** Endpoint disable

The application sets this bit to stop transmitting data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the endpoint disabled interrupt. The application must set this bit only if endpoint enable is already set for this endpoint.

Bits 29:28 Reserved, must be kept at reset value.

**Bit 27 SNAK:** Set NAK

A write to this bit sets the NAK bit for the endpoint.

Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for an endpoint after a SETUP packet is received on that endpoint.

**Bit 26 CNAK:** Clear NAK

A write to this bit clears the NAK bit for the endpoint.

**Bits 25:22 TXFNUM[3:0]:** Tx FIFO number

This value is set to the FIFO number that is assigned to IN endpoint 0.

**Bit 21 STALL:** STALL handshake

The application can only set this bit, and the core clears it when a SETUP token is received for this endpoint. If a NAK bit, a Global IN NAK or Global OUT NAK is set along with this bit, the STALL bit takes priority.

Bit 20 Reserved, must be kept at reset value.

**Bits 19:18 EPTYP[1:0]:** Endpoint type

Hardcoded to '00' for control.

**Bit 17 NAKSTS:** NAK status

Indicates the following:

0: The core is transmitting non-NAK handshakes based on the FIFO status

1: The core is transmitting NAK handshakes on this endpoint.

When this bit is set, either by the application or core, the core stops transmitting data, even if there are data available in the Tx FIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 16 Reserved, must be kept at reset value.

Bit 15 **USBAEP**: USB active endpoint

This bit is always set to 1, indicating that control endpoint 0 is always active in all configurations and interfaces.

Bits 14:2 Reserved, must be kept at reset value.

Bits 1:0 **MPSIZ[1:0]**: Maximum packet size

The application must program this field with the maximum packet size for the current logical endpoint.

00: 64 bytes

01: 32 bytes

10: 16 bytes

11: 8 bytes

*Note:* Configuration register applies only to USB OTG FS

### 37.15.51 OTG device IN endpoint x control register (OTG\_DIEPCTLx)

Address offset: 0x900 + 0x20 \* x, (x = 1 to 5[FS] / 0 to 8[HS])

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	SDD0 PID/SEVN FIRM	SNAK	CNAK	TXFNUM[3:0]				STALL	Res.	EPTYP[1:0]		NAKSTS	EO NUM/DPID
rs	rs	w	w	w	w	rw	rw	rw	rw	rw		rw	rw	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBAEP	Res.	Res.	Res.	Res.	MPSIZ[10:0]										
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **EPENA**: Endpoint enable

The application sets this bit to start transmitting data on an endpoint.

The core clears this bit before setting any of the following interrupts on this endpoint:

- SETUP phase done
- Endpoint disabled
- Transfer completed

Bit 30 **EPDIS**: Endpoint disable

The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the endpoint disabled interrupt. The application must set this bit only if endpoint enable is already set for this endpoint.

Bit 29 **SODDFRM**: Set odd frame

Applies to isochronous IN and OUT endpoints only.

Writing to this field sets the Even/Odd frame (EONUM) field to odd frame.

Bit 28 **SD0PID:** Set DATA0 PID

Applies to interrupt/bulk IN endpoints only.

Writing to this field sets the endpoint data PID (DPID) field in this register to DATA0.

**SEVNFRM:** Set even frame

Applies to isochronous IN endpoints only.

Writing to this field sets the Even/Odd frame (EONUM) field to even frame.

Bit 27 **SNAK:** Set NAK

A write to this bit sets the NAK bit for the endpoint.

Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for OUT endpoints on a transfer completed interrupt, or after a SETUP is received on the endpoint.

Bit 26 **CNAK:** Clear NAK

A write to this bit clears the NAK bit for the endpoint.

Bits 25:22 **TXFNUM[3:0]:** 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.

Bit 20 Reserved, must be kept at reset value.

Bits 19:18 **EPTYP[1:0]:** Endpoint type

This is the transfer type supported by this logical endpoint.

00: Control

01: Isochronous

10: Bulk

11: Interrupt

Bit 17 **NAKSTS:** NAK status

It indicates the following:

0: The core is transmitting non-NAK handshakes based on the FIFO status.

1: The core is transmitting NAK handshakes on this endpoint.

When either the application or the core sets this bit:

For non-isochronous IN endpoints: The core stops transmitting any data on an IN endpoint, even if there are data available in the Tx FIFO.

For isochronous IN endpoints: The core sends out a zero-length data packet, even if there are data available in the Tx FIFO.

Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 16 **EONUM:** Even/odd frame

Applies to isochronous IN endpoints only.

Indicates the frame number in which the core transmits/receives isochronous data for this endpoint. The application must program the even/odd frame number in which it intends to transmit/receive isochronous data for this endpoint using the SEVNFRM and SODDFRM fields in this register.

0: Even frame

1: Odd frame

**DPID:** Endpoint data PID

Applies to interrupt/bulk IN endpoints only.

Contains the PID of the packet to be received or transmitted on this endpoint. The application must program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. The application uses the SD0PID register field to program either DATA0 or DATA1 PID.

0: DATA0

1: DATA1

Bit 15 **USBAEP:** USB active endpoint

Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit.

Bits 14:11 Reserved, must be kept at reset value.

Bits 10:0 **MPSIZ[10:0]:** Maximum packet size

The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.

**37.15.52 OTG device IN endpoint x interrupt register (OTG\_DIEPINTx)**

Address offset: 0x908 + 0x20 \* x, (x = 0 to 5[FS] /8[HS])

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 463](#). The application must read this register when the IN endpoints interrupt bit of the core interrupt register (IEPINT in OTG\_GINTSTS) is set. Before the application can read this register, it must first read the device all endpoints interrupt (OTG\_DAINT) register to get the exact endpoint number for the device endpoint-x interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the OTG\_DAINT and OTG\_GINTSTS registers.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAK	Res.	PKTD RPSTS	Res.	Res.	Res.	TXFE	IN EPNE	IN EPNM	ITTXFE	TOC	Res.	EP DISD	XFRC
		rc_w1		rc_w1				r	rc_w1	rc_w1	rc_w1	rc_w1		rc_w1	rc_w1

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAK	Res.	PKTD RPSTS	Res.	Res.	TXFIF OUD RN	TXFE	IN EPNE	IN EPNM	ITTXFE	TOC	AHB ERR	EP DISD	XFRC
		rc_w1		rc_w1			rc_w1	r	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Note: Configuration register for USB OTG HS

Bits 31:14 Reserved, must be kept at reset value.

Bit 13 **NAK:** NAK input

The core generates this interrupt when a NAK is transmitted or received by the device. In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to unavailability of data in the Tx FIFO.

Bit 12 Reserved, must be kept at reset value.

Bit 11 **PKTDRPSTS:** Packet dropped status

This bit indicates to the application that an ISOC OUT packet has been dropped. This bit does not have an associated mask bit and does not generate an interrupt.

Bit 10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **TXFIFOUDRN:** Transmit Fifo Underrun (TxfifoUndrn)

The core generates this interrupt when it detects a transmit FIFO underrun condition for this endpoint. Dependency: This interrupt is valid only when Thresholding is enabled

Bit 7 **TXFE:** Transmit FIFO empty

This interrupt is asserted when the Tx FIFO for this endpoint is either half or completely empty. The half or completely empty status is determined by the Tx FIFO Empty Level bit in the OTG\_GAHBCFG register (TXFELVL bit in OTG\_GAHBCFG).

Bit 6 **INEPNE:** IN endpoint NAK effective

This bit can be cleared when the application clears the IN endpoint NAK by writing to the CNAK bit in OTG\_DIEPCTLx.

This interrupt indicates that the core has sampled the NAK bit set (either by the application or by the core). The interrupt indicates that the IN endpoint NAK bit set by the application has taken effect in the core.

This interrupt does not guarantee that a NAK handshake is sent on the USB. A STALL bit takes priority over a NAK bit.

Bit 5 **INEPNM:** IN token received with EP mismatch

Indicates that the data in the top of the non-periodic TxFIFO belongs to an endpoint other than the one for which the IN token was received. This interrupt is asserted on the endpoint for which the IN token was received.

Bit 4 **ITTXFE:** IN token received when Tx FIFO is empty

Indicates that an IN token was received when the associated Tx FIFO (periodic/non-periodic) was empty. This interrupt is asserted on the endpoint for which the IN token was received.

Bit 3 **TOC**: Timeout condition

Indicates that the core has detected a timeout condition on the USB for the last IN token on this endpoint.

Bit 2 **AHBERR**: AHB error for USB OTG HS

This is generated only in internal DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.

Bit 1 **EPDISD**: Endpoint disabled interrupt

This bit indicates that the endpoint is disabled per the application's request.

Bit 0 **XFRS**: Transfer completed interrupt

This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.

### 37.15.53 OTG device IN endpoint 0 transfer size register (OTG\_DIEPTSIZE0)

Address offset: 0x910

Reset value: 0x0000 0000

The application must modify this register before enabling endpoint 0. Once endpoint 0 is enabled using the endpoint enable bit in the device control endpoint 0 control registers (EPENA in OTG\_DIEPCTL0), the core modifies this register. The application can only read this register once the core has cleared the endpoint enable bit.

Nonzero endpoints use the registers for endpoints 1–3.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	PKTCNT[1:0]	Res.	Res.	Res.	Res.										
											rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	rw	rw	rw	rw	rw										
											XFRSIZ[6:0]				
											rw	rw	rw	rw	rw

Bits 31:21 Reserved, must be kept at reset value.

Bits 20:19 **PKTCNT[1:0]**: Packet count

Indicates the total number of USB packets that constitute the transfer size amount of data for endpoint 0.

This field is decremented every time a packet (maximum size or short packet) is read from the Tx FIFO.

Bits 18:7 Reserved, must be kept at reset value.

Bits 6:0 **XFRSIZ[6:0]**: Transfer size

Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet.

The core decrements this field every time a packet from the external memory is written to the Tx FIFO.

### 37.15.54 OTG device IN endpoint x DMA address register (OTG\_DIEPDMAx)

Address offset: 0x914 + 0x20 \* x, (x = 0 to 8)

Reset value: 0x0000 0000

*Note:* Configuration register applies only to USB OTG HS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DMAADDR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAADDR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DMAADDR[31:0]: DMA Address**

This field holds the start address in the external memory from which the data for the endpoint must be fetched. This register is incremented on every AHB transaction.

### 37.15.55 OTG device IN endpoint transmit FIFO status register (OTG\_DTXFSTSx)

Address offset: 0x918 + 0x20 \* x, (x = 0 to 5[FS] /8[HS])

Reset value: 0x0000 0200

This read-only register contains the free space information for the device IN endpoint Tx FIFO.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INEPTFSAV[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **INEPTFSAV[15:0]: IN endpoint Tx FIFO space available**

Indicates the amount of free space available in the endpoint Tx FIFO.

Values are in terms of 32-bit words:

0x0: Endpoint Tx FIFO is full

0x1: 1 word available

0x2: 2 words available

0xn: n words available

Others: Reserved

### 37.15.56 OTG device IN endpoint x transfer size register (OTG\_DIEPTSI<sub>x</sub>)

Address offset: 0x910 + 0x20 \* x, (x = 1 to 5[FS] /8[HS])

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<sub>x</sub> registers (EPENA bit in OTG\_DIEPCTL<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.	MCNT[1:0]		PKTCNT[9:0]												XFRSIZ[18:16]
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XFRSIZ[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 Reserved, must be kept at reset value.

Bits 30:29 **MCNT[1:0]: Multi count**

For periodic IN endpoints, this field indicates the number of packets that must be transmitted per frame on the USB. The core uses this field to calculate the data PID for isochronous IN endpoints.

01: 1 packet

10: 2 packets

11: 3 packets

Bits 28:19 **PKTCNT[9:0]: Packet count**

Indicates the total number of USB packets that constitute the transfer size amount of data for this endpoint.

This field is decremented every time a packet (maximum size or short packet) is read from the Tx FIFO.

Bits 18:0 **XFRSIZ[18:0]: Transfer size**

This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet.

The core decrements this field every time a packet from the external memory is written to the Tx FIFO.

### 37.15.57 OTG device control OUT endpoint 0 control register (OTG\_DOEPCTL0)

Address offset: 0xB00

Reset value: 0x0000 8000

This section describes the OTG\_DOEPCTL0 register. Nonzero control endpoints use registers for endpoints 1–3.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPENA	EPDIS	Res.	Res.	SNAK	CNAK	Res.	Res.	Res.	Res.	STALL	SNPM	EPTYP[1:0]	NAK STS	Res.	
w	r			w	w					rs	rw	r	r	r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBA EP	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MPSIZ[1:0]	
r														r	r

Bit 31 **EPENA:** Endpoint enable

The application sets this bit to start transmitting data on endpoint 0.

The core clears this bit before setting any of the following interrupts on this endpoint:

- SETUP phase done
- Endpoint disabled
- Transfer completed

Bit 30 **EPDIS:** Endpoint disable

The application cannot disable control OUT endpoint 0.

Bits 29:28 Reserved, must be kept at reset value.

Bit 27 **SNAK:** Set NAK

A write to this bit sets the NAK bit for the endpoint.

Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit on a transfer completed interrupt, or after a SETUP is received on the endpoint.

Bit 26 **CNAK:** Clear NAK

A write to this bit clears the NAK bit for the endpoint.

Bits 25:22 Reserved, must be kept at reset value.

Bit 21 **STALL:** STALL handshake

The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 20 **SNPM:** Snoop mode

This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.

Bits 19:18 **EPTYP[1:0]:** Endpoint type

Hardcoded to 2'b00 for control.

Bit 17 **NAKSTS**: NAK status

Indicates the following:

0: The core is transmitting non-NAK handshakes based on the FIFO status.

1: The core is transmitting NAK handshakes on this endpoint.

When either the application or the core sets this bit, the core stops receiving data, even if there is space in the Rx FIFO to accommodate the incoming packet. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 16 Reserved, must be kept at reset value.

Bit 15 **USBAEP**: USB active endpoint

This bit is always set to 1, indicating that a control endpoint 0 is always active in all configurations and interfaces.

Bits 14:2 Reserved, must be kept at reset value.

Bits 1:0 **MPSIZ[1:0]**: Maximum packet size

The maximum packet size for control OUT endpoint 0 is the same as what is programmed in control IN endpoint 0.

00: 64 bytes

01: 32 bytes

10: 16 bytes

11: 8 bytes

**37.15.58 OTG device OUT endpoint x interrupt register (OTG\_DOEPINTx)**

Address offset: 0xB08 + 0x20 \* x, (x = 0 to 5[FS] /8[HS])

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 463](#). The application must read this register when the OUT endpoints interrupt bit of the OTG\_GINTSTS register (OEPINT bit in OTG\_GINTSTS) is set. Before the application can read this register, it must first read the OTG\_DAINT register to get the exact endpoint number for the OTG\_DOEPINTx register. The application must clear the appropriate bit in this register to clear the corresponding bits in the OTG\_DAINT and OTG\_GINTSTS registers.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAK	BERR	Res.	Res.	Res.	Res.	Res.	Res.	STSPH SRX	OTEPE DIS	STUP	Res.	EP DISD	XFRC
		rc_w1	rc_w1							rc_w1	rc_w1	rc_w1		rc_w1	rc_w1

Note: Configuration register for USB OTG FS.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STPK TRX	NYET	NAK	BERR	Res.	Res.	Res.	OUT PKT ERR	Res.	B2B STUP	STS PH SRX	OTE P DIS	STUP	AHB ERR	EP DISD	XFRC
rc_w1	rc_w1	rc_w1	rc_w1				rc_w1		rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Note: Configuration register for USB OTG HS.

Bits 31:16 Reserved, must be kept at reset value.

Bit 15 **STPKTRX:** Setup packet received for USB OTG HS

Applicable for control OUT endpoints in only in the Buffer DMA Mode. Set by the OTG\_HS, this bit indicates that this buffer holds 8 bytes of setup data. There is only one setup packet per buffer. On receiving a setup packet, the OTG\_HS closes the buffer and disables the corresponding endpoint after SETUP\_COMPLETE status is seen in the Rx FIFO. OTG\_HS puts a SETUP\_COMPLETE status into the Rx FIFO when it sees the first IN or OUT token after the SETUP packet for that particular endpoint. The application must then re-enable the endpoint to receive any OUT data for the control transfer and reprogram the buffer start address. Because of the above behavior, OTG\_HS can receive any number of back to back setup packets and one buffer for every setup packet is used.

Bit 14 **NYET:** NYET interrupt for USB OTG HS

This interrupt is generated when a NYET response is transmitted for a non isochronous OUT endpoint.

Bit 13 **NAK:** NAK input

The core generates this interrupt when a NAK is transmitted or received by the device. In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to unavailability of data in the Tx FIFO.

Bit 12 **BERR:** Babble error interrupt

The core generates this interrupt when babble is received for the endpoint.

Bits 11:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **OUTPKTERR:** OUT packet error for USB OTG HS

This interrupt is asserted when the core detects an overflow or a CRC error for an OUT packet. This interrupt is valid only when thresholding is enabled.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **B2BSTUP:** Back-to-back SETUP packets received for USB OTG HS

Applies to control OUT endpoint only.

This bit indicates that the core has received more than three back-to-back SETUP packets for this particular endpoint.

Bit 5 **STS PH SRX:** Status phase received for control write

This interrupt is valid only for control OUT endpoints. This interrupt is generated only after OTG\_FS/OTG\_HS has transferred all the data that the host has sent during the data phase of a control write transfer, to the system memory buffer. The interrupt indicates to the application that the host has switched from data phase to the status phase of a control write transfer. The application can use this interrupt to ACK or STALL the status phase, after it has decoded the data phase.

Bit 4 **OTEPDIS**: OUT token received when endpoint disabled

Applies only to control OUT endpoints.

Indicates that an OUT token was received when the endpoint was not yet enabled. This interrupt is asserted on the endpoint for which the OUT token was received.

Bit 3 **STUP**: SETUP phase done

Applies to control OUT endpoint only. Indicates that the SETUP phase for the control endpoint is complete and no more back-to-back SETUP packets were received for the current control transfer. On this interrupt, the application can decode the received SETUP data packet.

Bit 2 **AHBERR**: AHB error for USB OTG HS

This is generated only in internal DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.

Bit 1 **EPDISD**: Endpoint disabled interrupt

This bit indicates that the endpoint is disabled per the application's request.

Bit 0 **XFRIC**: Transfer completed interrupt

This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.

### 37.15.59 OTG device OUT endpoint 0 transfer size register (OTG\_DOEPTSIZ0)

Address offset: 0xB10

Reset value: 0x0000 0000

The application must modify this register before enabling endpoint 0. Once endpoint 0 is enabled using the endpoint enable bit in the OTG\_DOEPCTL0 registers (EPENA bit in OTG\_DOEPCTL0), the core modifies this register. The application can only read this register once the core has cleared the endpoint enable bit.

Nonzero endpoints use the registers for endpoints 1–5[FS] /8[HS].

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	STUPCNT[1:0]	Res.	Res.	Res.	PKTCNT	Res.	Res.	Res.								
	rw	rw											rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	rw	rw	rw	rw	rw	rw	rw
										xfrsiz[6:0]						

Bit 31 Reserved, must be kept at reset value.

Bits 30:29 **STUPCNT[1:0]:** SETUP packet count

This field specifies the number of back-to-back SETUP data packets the endpoint can receive.

- 01: 1 packet
- 10: 2 packets
- 11: 3 packets

Bits 28:20 Reserved, must be kept at reset value.

Bit 19 **PKTCNT:** Packet count

This field is decremented to zero after a packet is written into the Rx FIFO.

Bits 18:7 Reserved, must be kept at reset value.

Bits 6:0 **XFRSIZ[6:0]:** Transfer size

Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet.

The core decrements this field every time a packet is read from the Rx FIFO and written to the external memory.

### 37.15.60 OTG device OUT endpoint x DMA address register (OTG\_DOEPDMAx)

Address offset: 0xB14 + 0x20 \* x, (x = 0 to 8)

Reset value: 0x0000 0000

*Note:* Configuration register applies only to USB OTG HS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DMAADDR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAADDR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DMAADDR[31:0]:** DMA Address

This field holds the start address in the external memory from which the data for the endpoint must be fetched. This register is incremented on every AHB transaction.

### 37.15.61 OTG device OUT endpoint x control register (OTG\_DOEPCTLx)

Address offset: 0xB00 + 0x20 \* x, (x = 1 to 5[FS] /8[HS])

Reset value: 0x0000 0000

The application uses this register to control the behavior of each logical endpoint other than endpoint 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPENA	EPDIS	SD1 PID/ SODD FRM	SD0 PID/ SEVN FRM	SNAK	CNAK	Res.	Res.	Res.	Res.	STALL	SNPM	EPTYP[1:0]	NAK STS	EO NUM/ DPID	
rs	rs	w	w	w	w					rw	rw	rw	rw	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBA EP	Res.	Res.	Res.	Res.								MPSIZ[10:0]			
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **EPENA:** Endpoint enable

Applies to IN and OUT endpoints.

The application sets this bit to start transmitting data on an endpoint.

The core clears this bit before setting any of the following interrupts on this endpoint:

- SETUP phase done
- Endpoint disabled
- Transfer completed

Bit 30 **EPDIS:** Endpoint disable

The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the endpoint disabled interrupt. The application must set this bit only if endpoint enable is already set for this endpoint.

Bit 29 **SD1PID:** Set DATA1 PID

Applies to interrupt/bulk IN and OUT endpoints only. Writing to this field sets the endpoint data PID (DPID) field in this register to DATA1.

**SODDFRM:** Set odd frame

Applies to isochronous IN and OUT endpoints only. Writing to this field sets the Even/Odd frame (EONUM) field to odd frame.

Bit 28 **SD0PID:** Set DATA0 PID

Applies to interrupt/bulk OUT endpoints only.

Writing to this field sets the endpoint data PID (DPID) field in this register to DATA0.

**SEVNFRM:** Set even frame

Applies to isochronous OUT endpoints only.

Writing to this field sets the Even/Odd frame (EONUM) field to even frame.

Bit 27 **SNAK:** Set NAK

A write to this bit sets the NAK bit for the endpoint.

Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for OUT endpoints on a transfer completed interrupt, or after a SETUP is received on the endpoint.

Bit 26 **CNAK:** Clear NAK

A write to this bit clears the NAK bit for the endpoint.

Bits 25:22 Reserved, must be kept at reset value.

Bit 21 **STALL:** STALL handshake

Applies to non-control, non-isochronous OUT endpoints only (access type is rw).

The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, Global IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the core.

Applies to control endpoints only (access type is rs).

The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 20 **SNPM:** Snoop mode

This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.

Bits 19:18 **EPTYP[1:0]:** Endpoint type

This is the transfer type supported by this logical endpoint.

00: Control

01: Isochronous

10: Bulk

11: Interrupt

Bit 17 **NAKSTS:** NAK status

Indicates the following:

0: The core is transmitting non-NAK handshakes based on the FIFO status.

1: The core is transmitting NAK handshakes on this endpoint.

When either the application or the core sets this bit:

The core stops receiving any data on an OUT endpoint, even if there is space in the Rx FIFO to accommodate the incoming packet.

Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 16 **EONUM:** Even/odd frame

Applies to isochronous IN and OUT endpoints only.

Indicates the frame number in which the core transmits/receives isochronous data for this endpoint. The application must program the even/odd frame number in which it intends to transmit/receive isochronous data for this endpoint using the SEVNFRM and SODDFRM fields in this register.

0: Even frame

1: Odd frame

**DPID:** Endpoint data PID

Applies to interrupt/bulk OUT endpoints only.

Contains the PID of the packet to be received or transmitted on this endpoint. The application must program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. The application uses the SD0PID register field to program either DATA0 or DATA1 PID.

0: DATA0

1: DATA1

Bit 15 **USBAEP**: USB active endpoint

Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit.

Bits 14:11 Reserved, must be kept at reset value.

Bits 10:0 **MPSIZ[10:0]**: Maximum packet size

The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.

### 37.15.62 OTG device OUT endpoint x transfer size register (OTG\_DOEPTSIZx)

Address offset: 0xB10 + 0x20 \* x, (x = 1 to 5[FS] /8[HS])

Reset value: 0x0000 0000

The application must modify this register before enabling the endpoint. Once the endpoint is enabled using endpoint enable bit of the OTG\_DOEPCTLx registers (EPENA bit in OTG\_DOEPCTLx), the core modifies this register. The application can only read this register once the core has cleared the endpoint enable bit.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.		RXDPID/ STUPCNT[1:0]													XFRSIZ[18:16]	
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
XFRSIZ[15:0]																
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 Reserved, must be kept at reset value.

Bits 30:29 **RXDPID[1:0]:** Received data PID

Applies to isochronous OUT endpoints only.

This is the data PID received in the last packet for this endpoint.

00: DATA0

01: DATA2

10: DATA1

11: MDATA

**STUPCNT[1:0]:** SETUP packet count

Applies to control OUT endpoints only.

This field specifies the number of back-to-back SETUP data packets the endpoint can receive.

01: 1 packet

10: 2 packets

11: 3 packets

Bits 28:19 **PKTCNT[9:0]:** Packet count

Indicates the total number of USB packets that constitute the transfer size amount of data for this endpoint.

This field is decremented every time a packet (maximum size or short packet) is written to the Rx FIFO.

Bits 18:0 **XFRSIZ[18:0]:** Transfer size

This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet.

The core decrements this field every time a packet is read from the Rx FIFO and written to the external memory.

### 37.15.63 OTG power and clock gating control register (OTG\_PCGCCTL)

Address offset: 0xE00

Reset value: 0x200B 8000

This register is available in host and device modes.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
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							

Bits 31:8 Reserved, must be kept at reset value.

Bit 7 **SUSP:** Deep Sleep

This bit indicates that the PHY is in Deep Sleep when in L1 state.

Bit 6 **PHYSLEEP:** PHY in Sleep

This bit indicates that the PHY is in the Sleep state.

Bit 5 **ENL1GTG**: Enable sleep clock gating

When this bit is set, core internal clock gating is enabled in Sleep state if the core cannot assert utmi\_l1\_suspend\_n. When this bit is not set, the PHY clock is not gated in Sleep state.

Bit 4 **PHYSUSP**: PHY suspended

Indicates that the PHY has been suspended. This bit is updated once the PHY is suspended after the application has set the STPPCLK bit.

Bits 3:2 Reserved, must be kept at reset value.

Bit 1 **GATEHCLK**: Gate HCLK

The application sets this bit to gate HCLK to modules other than the AHB Slave and Master and wake-up 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.

**37.15.64 OTG\_FS/OTG\_HS register map**

The table below gives the USB OTG register map and reset values.

**Table 276. OTG\_FS/OTG\_HS register map and reset values**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TRDT	HBSTLEN	TOCAL	SRQ	0	SRQSCS
0x000	<b>OTG_GOTGCTL</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CURMOD	OTGVER	BSVLD	ASVLD	DBCT	CIDSTS	EHEN	DHNPN	HSHNPN	HNPRQ	SRSSCHG	HNGSCS	BVALOVAL	SEDET	AVALOVAL	0	0	0	0	0	0							
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
0x004	<b>OTG_GOTGINT</b>	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							
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
0x008	<b>OTG_GAHBCFG</b>	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							
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
0x008	<b>OTG_GAHBCFG</b>	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							
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
0x00C	<b>OTG_GUSBCFG</b>	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					
		Reset value	0	0	0	0	0	0	0	0	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 276. OTG\_FS/OTG\_HS register map and reset values (continued)**

Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x01C	OTG_GRXSTSR (Device mode)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FRMNUM	PKTSTS	DPID	BCNT	EPNUM																			
	Reset value					0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	OTG_GRXSTSR (Host mode)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PKTSTS	DPID	BCNT	CHNUM																				
	Reset value									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x020	OTG_GRXSTSP (Device mode)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FRMNUM	PKTSTS	DPID	BCNT	EPNUM																			
	Reset value					0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	OTG_GRXSTSP (Host mode)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PKTSTS	DPID	BCNT	CHNUM																				
	Reset value									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
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.	RXFD			
	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		
0x02C	OTG_HNPTXSTS	Res.								NPTXQTOP		NPTQXSAV		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		
0x038	OTG_GCCFG	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	VBDEN	DCDEN	PWRDWN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DCDET			
	Reset value									0	0	0	X																				
0x03C	OTG_CID									PRODUCT_ID (for USB OTG FS)																							
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x03C	OTG_CID									PRODUCT_ID (for USB OTG HS)																							
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x054	OTG_GLPMCFG	Res.	Res.	Res.	ENBESL	LPMR CNTSTS	SNDLPM	LPM RCNT	LPMCHIDX	L1RSMOK	SLPSTS	LPM RSP	L1DSEN	BESLTHRS	L1SSEN	REMWAKE																	
	Reset value				0	0	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	1	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	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x104	OTG_DIEPTXF1																																
	Reset value	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
0x108	OTG_DIEPTXF2																																
	Reset value	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0x114	OTG_DIEPTXF5																																
	Reset value	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0x120	OTG_DIEPTXF7																																
	Reset value	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	
0x400	OTG_HCFG	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FSLSS	0	0	0	
	Reset value																												0	0	0	0	
0x404	OTG_HFIR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																
0x408	OTG_HFNUM																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
0x410	OTG_HPTXSTS																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	
0x414	OTG_HAIT	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	HAIINT			
	Reset value																																
0x418	OTG_HAINTMSK	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	HAIINTM			
	Reset value																																
0x440	OTG_HPRT	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PSPD	PPWR	PLSTS	PRST	PSUSP	PRES	POCCHNG	POCA	PENCHNG	PENA	PCDET	PCSTS	0	0	0	0	
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x500	OTG_HCCHAR0	CHENA	CHDIS	ODDFRM	DAD	MCNT	EP1YP	LSDEV	EPDIR	EPNUM	EPNUM	EPNUM	EPNUM	EPNUM	EPNUM	MPSIZ																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)**

**Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)**

Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	
0x6E8	OTG_HCINT15	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
0x6EC	OTG_HCINTMSK15	Reset value	DOPNG	Res.																									
0x6F0	OTG_HCTSIZ15	Reset value	0 0	DPID	PKTCNT																								
0x6F4	OTG_HCDMA15	Reset value	0 0	OTG_DCFG	PERS CHI VL																								
0x800	OTG_DCFG	Reset value	0 0	OTG_DCTL	DEV LN STS																								
0x800	OTG_DSTS	Reset value	0 0	OTG_DIEPMSK	FNSOF																								
0x804	OTG_DIEPMSK	Reset value	0 0	OTG_DOEPMSK	NAKMSK																								
0x808	OTG_DOEPMSK	Reset value	0 0	OTG_DIEPMSK	BERRM																								
0x810	OTG_DIEPMSK	Reset value	0 0	OTG_DIEPMSK	POPRGDNE																								
0x814	OTG_DIEPMSK	Reset value	0 0	OTG_DIEPMSK	TOM	EERR	GONSTS																						
0x818	OTG_DIEPMSK	Reset value	0 0	OTG_DIEPMSK	AHBERRM	AHBERRM	ENUMSPD																						
0x822	OTG_DIEPMSK	Reset value	0 0	OTG_DIEPMSK	XFRCM	XFRCM	SUSPSTS																						

**Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)**

Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x900	<b>OTG_DIEPCTL0</b>	EPENA	0	EPDIS	0	SODDFRM/SD1PID	0	SODDFRM/SD1PID	0	SODDFRM/SEVNFRM	0	SODPID/SEVNFRM	0	CNAK	0	STALL	0	Res.	0	EPDIS	0	EPDIS	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		
0x908	<b>OTG_DIEPINT0</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x910	<b>OTG_DIEPTSIZ0</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x914	<b>OTG_DIEPDMA</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x918	<b>OTG_DTXFSTS0</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x920	<b>OTG_DIEPCTL1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x928	<b>OTG_DIEPINT1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x930	<b>OTG_DIEPTSIZ1</b>	MCNT	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	0	0		
0x938	<b>OTG_DTXFSTS1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x940	<b>OTG_DIEPCTL2</b>	EPENA	0	EPDIS	0	SODDFRM	0	SD0PID/SEVNFRM	0	SNAK	0	CNAK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x9A0	<b>OTG_DIEPCTL5</b>	EPENA	EPDIS	SODDFRM	SD0PID/SEVNFRM	SNAK	CNAK	TXFNUM				STALL	Res.	EPTYP	NAKSTS	EONUM/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		
0x9A8	<b>OTG_DIEPINT5</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																																
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.	INEPTFSAV				
		Reset value																																
0x9E0	<b>OTG_DIEPCTL7</b>	EPENA	EPDIS	SODDFRM	SD0PID/SEVNFRM	SNAK	CNAK	TXFNUM				STALL	Res.	EPTYP	NAKSTS	EONUM/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	0	
0x9B0	<b>OTG_DIEPTSIZ5</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.	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		
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.	MPSIZ				
		Reset value																																

**Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)**

Table 276. OTG\_FS/OTG\_HS register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xB34	OTG_DOEPDMA1																																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0xBA0	OTG_DOEPCTL5	EPENA																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0xBA8	OTG_DOEPINT5	Res.																															
		Reset value																															
0xBB0	OTG_DOEPTSIZ5	Res.	RXPID/STUPCNT																														
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0xC00	OTG_DOEPCTL8	EPENA																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0xC08	OTG_DOEPINT8	Res.	Res.																														
		Reset value																															
0xC10	OTG_DOEPTSIZ8	Res.	RXPID/STUPCNT																														
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0xC14	OTG_DOEPDMA8																																
		Reset value	0	0	0	0	0	0	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 276. OTG\_FS/OTG\_HS register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0xE00	OTG_PCGCCTL	Res.	SUSP	PHYSLEEP	ENL1GTG	ENL1GTG	PHYSUSP	Res.	Res.	GATEHCLK	STPPCLK	0																						
	Reset value																																	

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 37.16 OTG\_FS/OTG\_HS programming model

### 37.16.1 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.

### 37.16.2 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.

### 37.16.3 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
  - Periodic Frame Interval
2. Program the Device threshold control register. This is required only if you are using DMA mode and you are planning to enable thresholding.
3. Clear the DCTL.SDIS bit. The core issues a connect after this bit is cleared.
4. Program the OTG\_GINTMSK register to unmask the following interrupts:
  - USB reset
  - Enumeration done
  - Early suspend
  - USB suspend
  - SOF
5. 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.
6. 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 1487](#).

At this point, the device is ready to accept SOF packets and perform control transfers on control endpoint 0.

### 37.16.4 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.

### 37.16.5 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 which automatically clears the CHENA bit to 0.

The application is expected to disable a channel on any of the following conditions:

1. When an STALL, TXERR, BBERR or DTERR interrupt in OTG\_HCINTx is received for an IN or OUT channel. The application must be able to receive other interrupts (DTERR, Nak, data, TXERR) for the same channel before receiving the halt.
2. When an XFRC interrupt in OTG\_HCINTx is received during a non periodic IN transfer or high-bandwidth interrupt IN transfer
3. When a DISCINT (disconnect device) interrupt in OTG\_GINTSTS is received. (The application is expected to disable all enabled channels).
4. When the application aborts a transfer before normal completion.

### Ping protocol

When the OTG\_HS host operates in high speed, the application must initiate the ping protocol when communicating with high-speed bulk or control (data and status stage) OUT endpoints. The application must initiate the ping protocol when it receives a NAK/NYET/TXERR interrupt. When the OTG\_HS host receives one of the above responses, it does not continue any transaction for a specific endpoint, drops all posted or fetched OUT requests (from the request queue), and flushes the corresponding data (from the transmit FIFO). This is valid in slave mode only. In Slave mode, the application can send a ping token either by setting the DOPING bit in OTG\_HCTSIZx before enabling the channel or by just writing the OTG\_HCTSIZx register with the DOPING bit set when the channel is already enabled. This enables the OTG\_HS host to write a ping request entry to the request queue. The application must wait for the response to the ping token (a NAK, ACK, or TXERR interrupt) before continuing the transaction or sending another ping token. The application can continue the data transaction only after receiving an ACK from the OUT endpoint for the requested ping. In DMA mode operation, the application does not need to set the DOPING bit in OTG\_HCTSIZx for a NAK/NYET response in case of bulk/control OUT. The OTG\_HS host automatically sets the DOPING bit in OTG\_HCTSIZx, and issues the ping tokens for bulk/control OUT. The OTG\_HS host continues sending ping tokens until it receives an ACK, and then switches automatically to the data transaction.

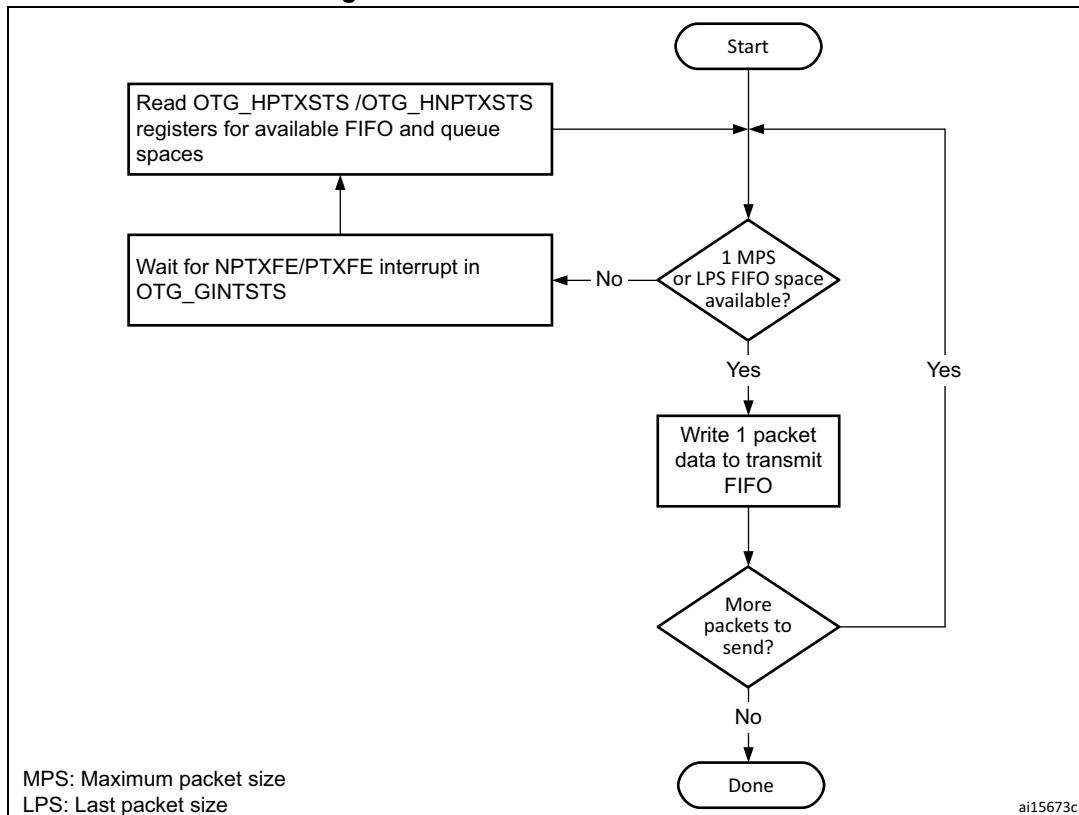
### Operational model

The application must initialize a channel before communicating to the connected device. This section explains the sequence of operation to be performed for different types of USB transactions.

- **Writing the transmit FIFO**

The OTG\_FS/OTG\_HS host automatically writes an entry (OUT request) to the periodic/non-periodic request queue, along with the last 32-bit word write of a packet. The application must ensure that at least one free space is available in the periodic/non-periodic request queue before starting to write to the transmit FIFO. The application must always write to the transmit FIFO in 32-bit words. If the packet size is non-32-bit word aligned, the application must use padding. The OTG\_FS/OTG\_HS host determines the actual packet size based on the programmed maximum packet size and transfer size.

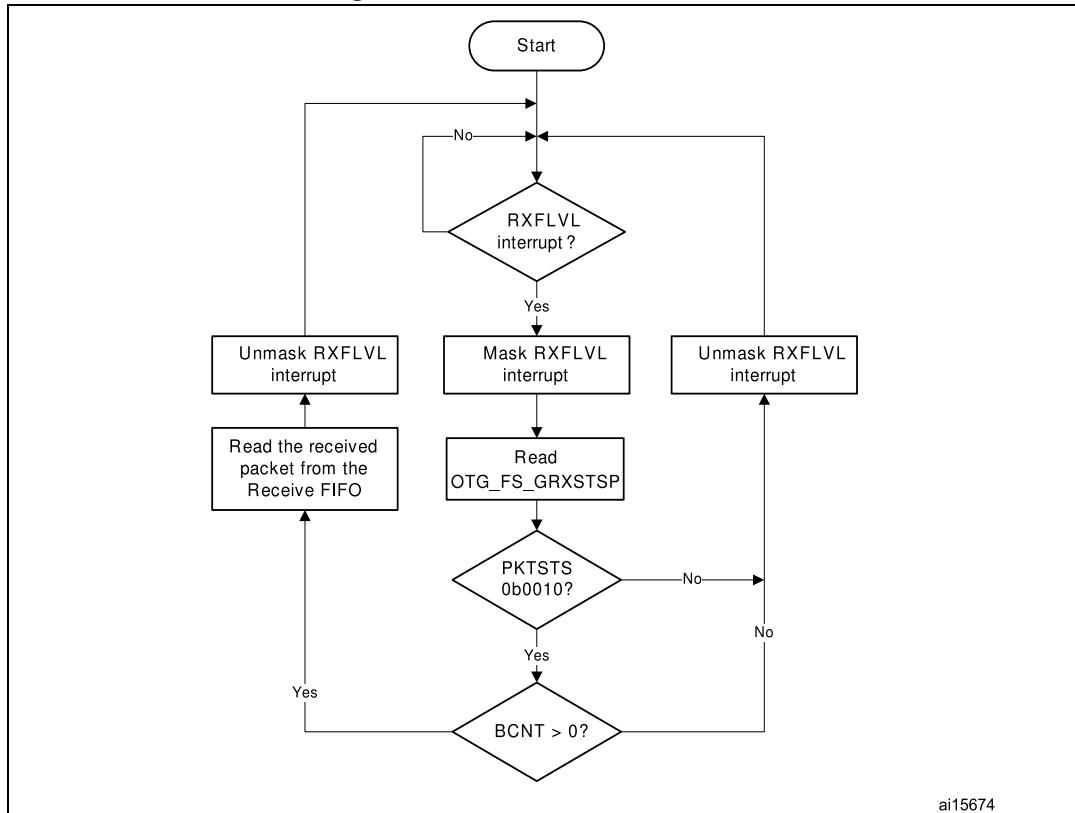
Figure 464. Transmit FIFO write task



- **Reading the receive FIFO**

The application must ignore all packet statuses other than IN data packet (bx0010).

Figure 465. 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 466](#). 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:

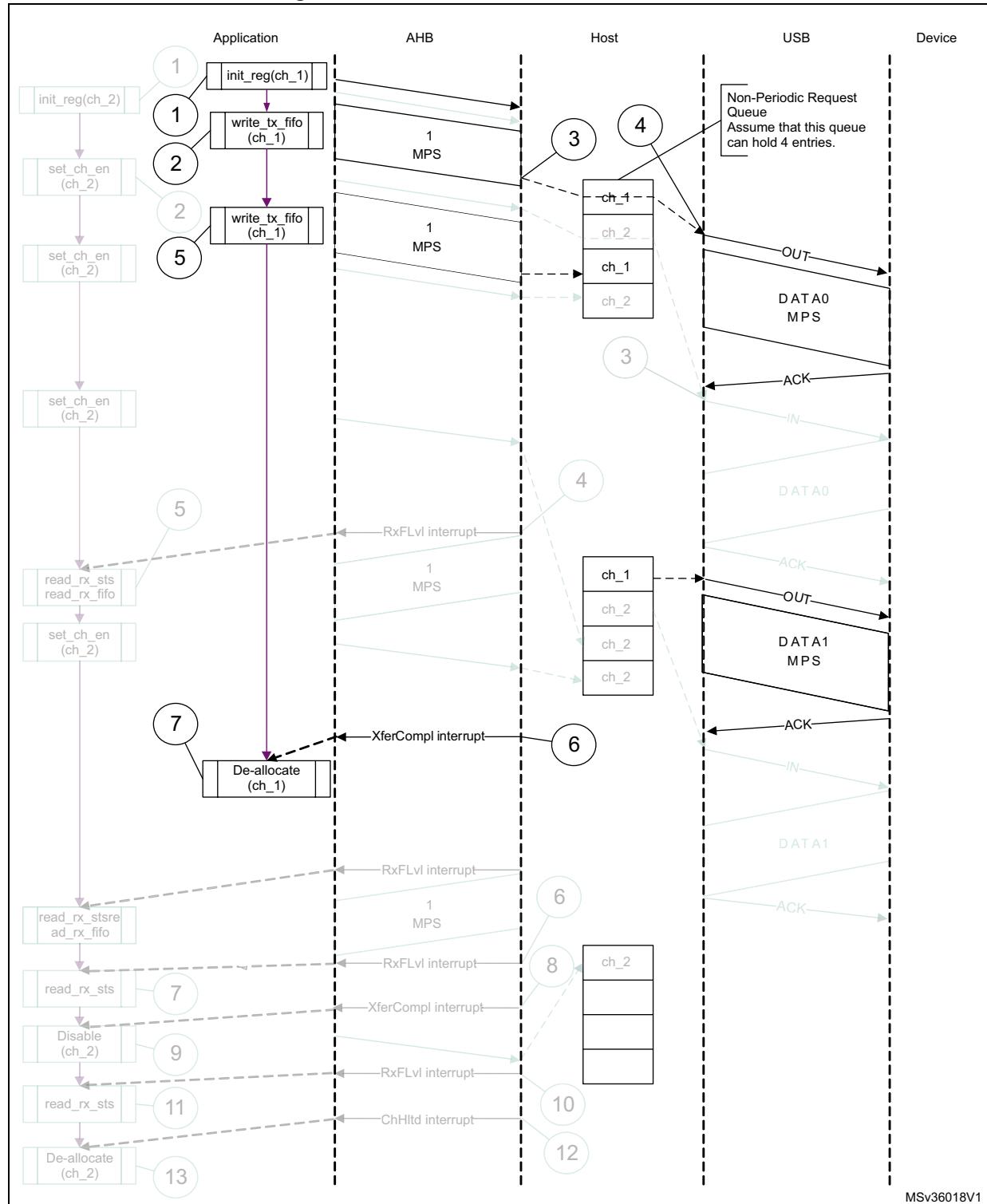
- The application is attempting to send two maximum-packet-size packets (transfer size = 1,024 bytes).
- The non-periodic transmit FIFO can hold two packets (1 Kbyte for HS/128 bytes for FS).
- The non-periodic request queue depth = 4.

- **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 466. 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
}
```

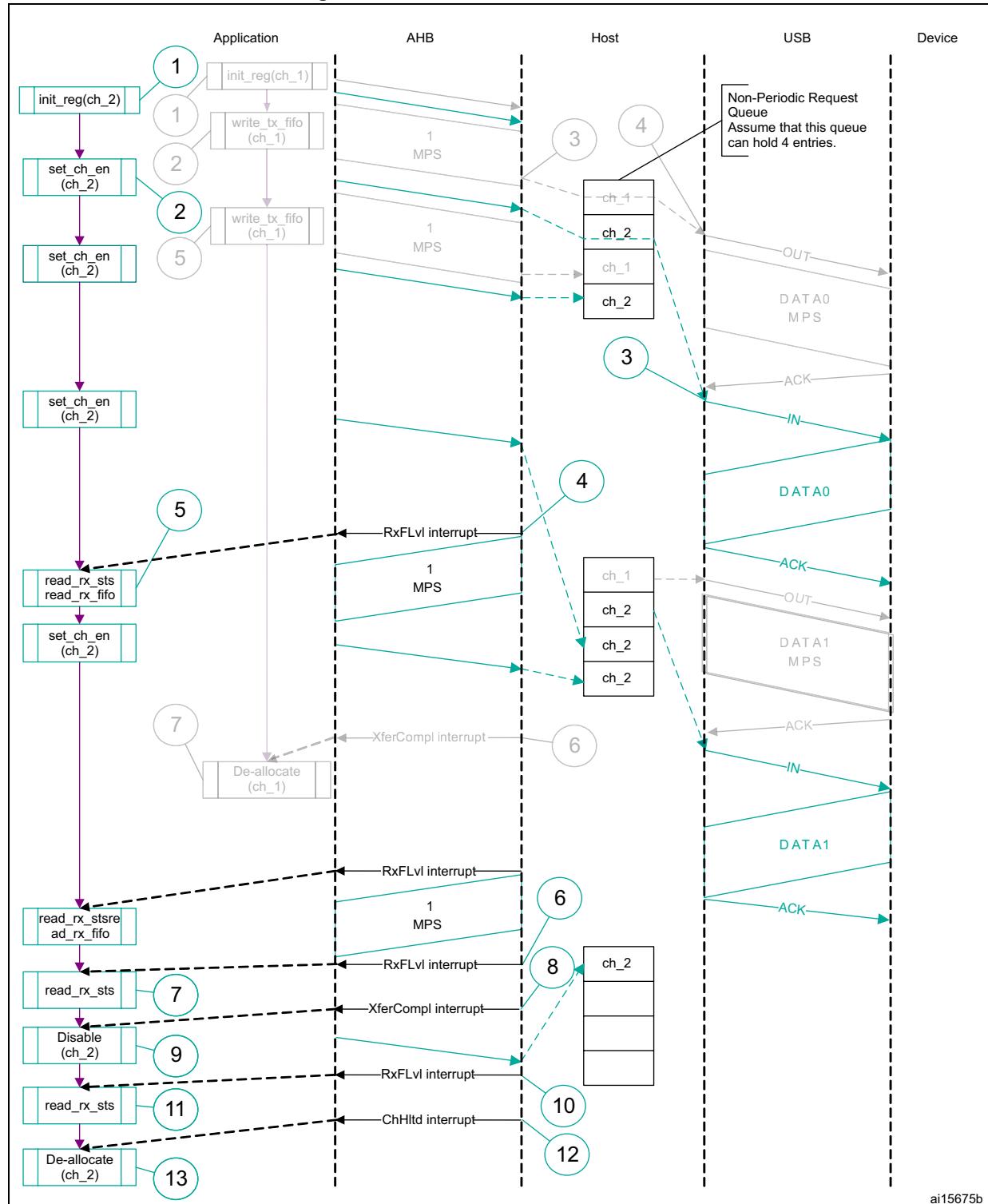
The application is expected to write the requests as and when the request queue space is available and until the XFRC interrupt is received.

- **Bulk and control IN transactions**

A typical bulk or control IN pipelined transaction-level operation is shown in [Figure 467](#). 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 467. 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**

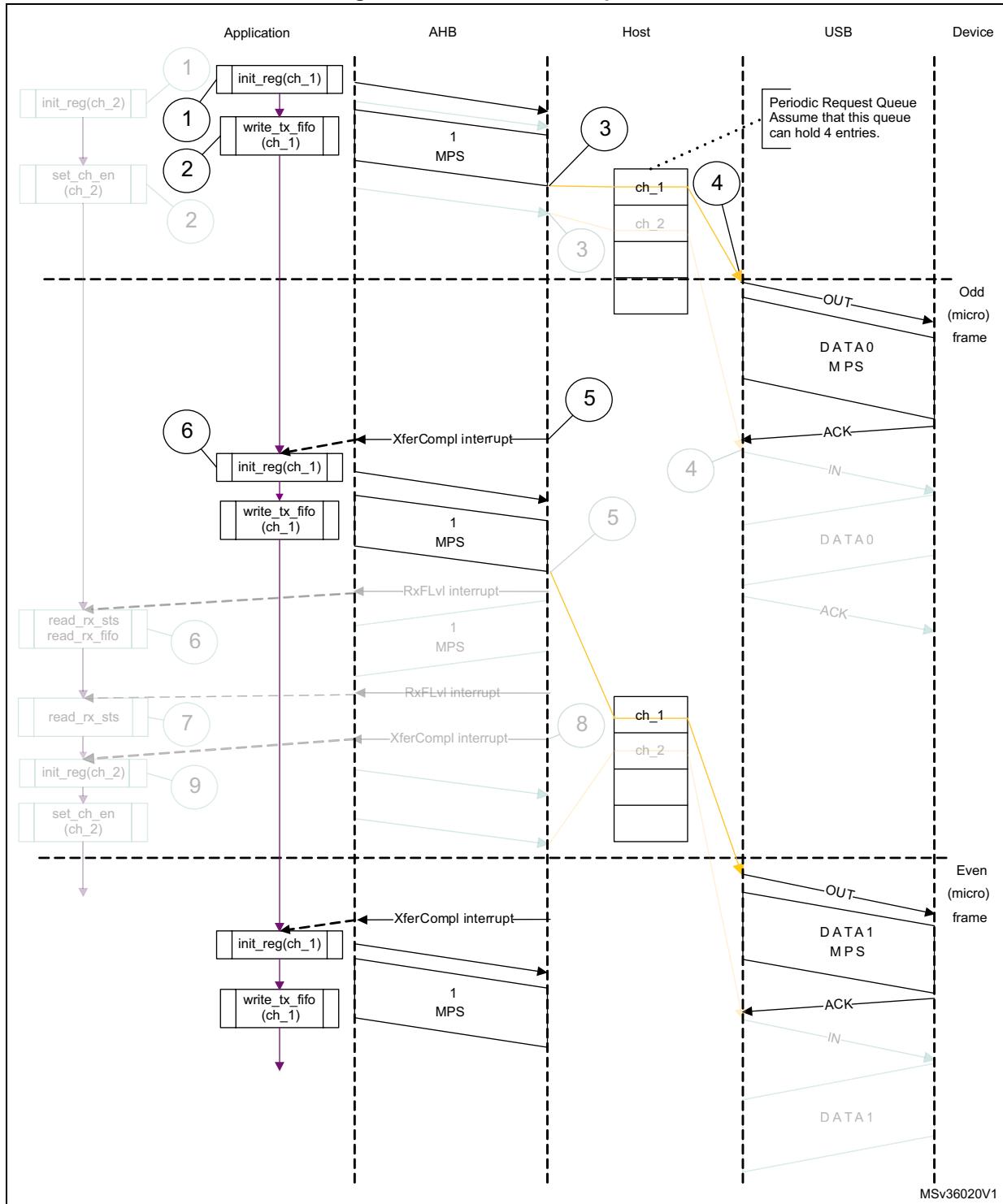
A typical interrupt OUT operation is shown in [Figure 468](#). The assumptions are:

- 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 Kbyte)
- 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 468. 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 CCH
        Disable Channel
    }
}
else
{
    if (STALL or FRMOR or NAK or DTERR or BBERR)
    {
        Mask ACK
        Unmask CCH
        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 CCH
        Disable Channel
    }
}
```

```

if (CCH)
{
    Mask CCH
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
        Re-initialize Channel (in next b_interval - 1 /Frame)
}
else
    if (ACK)
    {
        Reset Error Count
        Mask ACK
    }

```

- **Interrupt IN transactions**

The assumptions are:

- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame, starting with odd (transfer size = 1 024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status words per packet (1 031 bytes).
- Periodic request queue depth = 4.

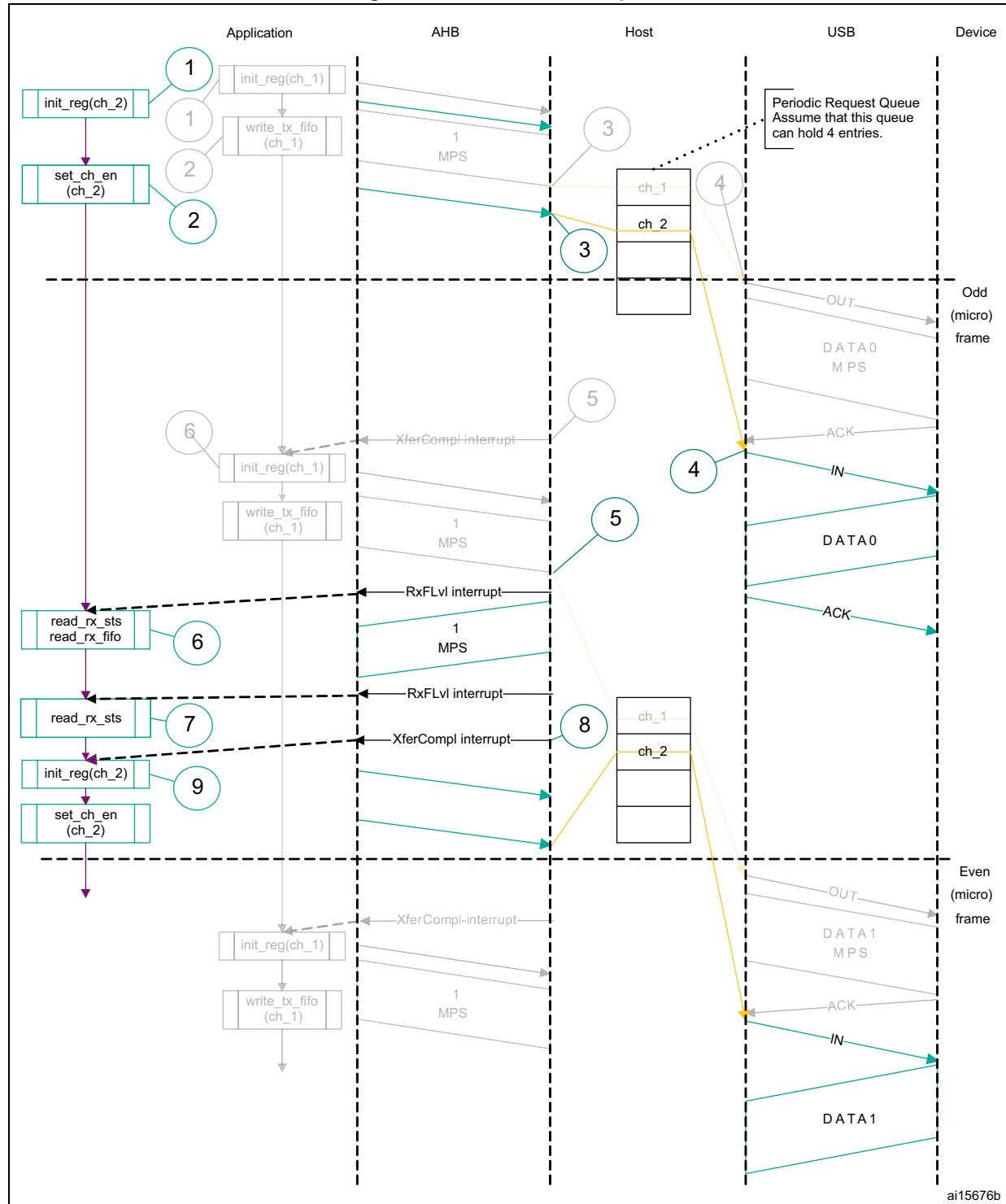
- **Normal interrupt IN operation**

The sequence of operations is as follows:

1. Initialize channel 2. The application must set the ODDFRM bit in OTG\_HCCHAR2.
2. Set the CHENA bit in OTG\_HCCHAR2 to write an IN request to the periodic request queue.
3. The OTG\_FS/OTG\_HS host writes an IN request to the periodic request queue for each OTG\_HCCHAR2 register write with the CHENA bit set.
4. The OTG\_FS/OTG\_HS host attempts to send an IN token in the next (odd) frame.
5. As soon as the IN packet is received and written to the receive FIFO, the OTG\_FS/OTG\_HS host generates an RXFLVL interrupt.
6. In response to the RXFLVL interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RXFLVL interrupt before reading the receive FIFO, and unmask after reading the entire packet.
7. The core generates the RXFLVL interrupt for the transfer completion status entry in the receive FIFO. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (PKTSTS in GRXSTS ≠ 0b0010).
8. The core generates an XFRC interrupt as soon as the receive packet status is read.
9. In response to the XFRC interrupt, read the PKTCNT field in OTG\_HCTSIZ2. If the PKTCNT bit in OTG\_HCTSIZ2 is not equal to 0, disable the channel before re-

initializing the channel for the next transfer, if any). If PKTCNT bit in OTG\_HCTSIZ2 = 0, reinitialize the channel for the next transfer. This time, the application must reset the ODDFRM bit in OTG\_HCCHAR2.

Figure 469. 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 470](#). The assumptions are:

- The application is attempting to send one packet every frame (up to 1 maximum)

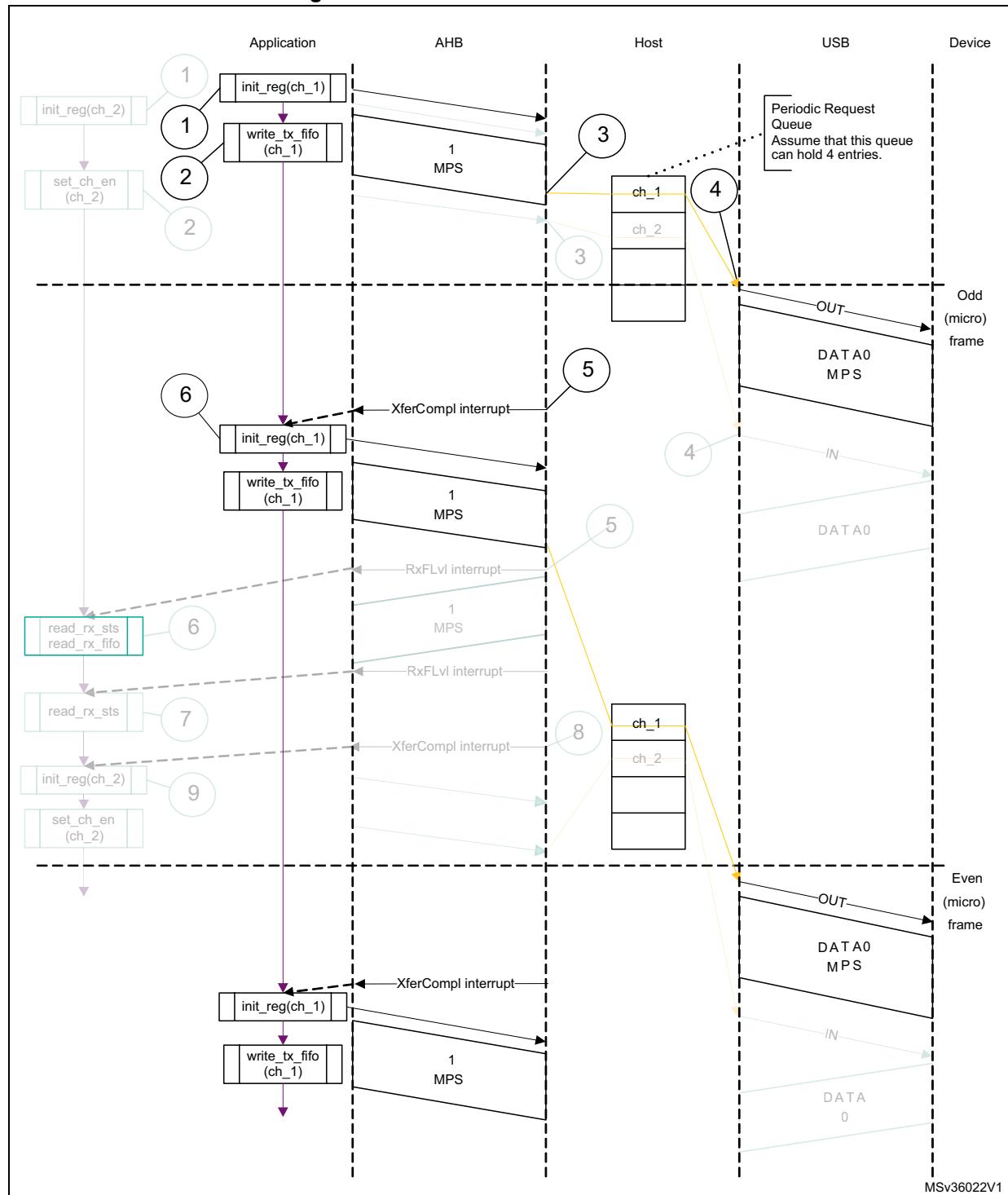
packet size), starting with an odd frame. (transfer size = 1 024 bytes).

- The periodic transmit FIFO can hold one packet (1 Kbyte).
- 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 470. Isochronous OUT transactions



1. The grayed elements are not relevant in the context of this figure.

- **Interrupt service routine for isochronous OUT/IN transactions**

Code sample: isochronous OUT

**Unmask (FMRMOR/XFRC)**

**if (XFRC)**

```
{  
    De-allocate Channel  
}  
  
else  
if (FRMOR)  
{  
    Unmask CHH  
    Disable Channel  
}  
  
else  
if (CHH)  
{  
    Mask CHH  
    De-allocate Channel  
}  
  
Code sample: Isochronous IN  
Unmask (TXERR/XFRC/FRMOR/BBERR)  
if (XFRC or FMRM)  
{  
    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**

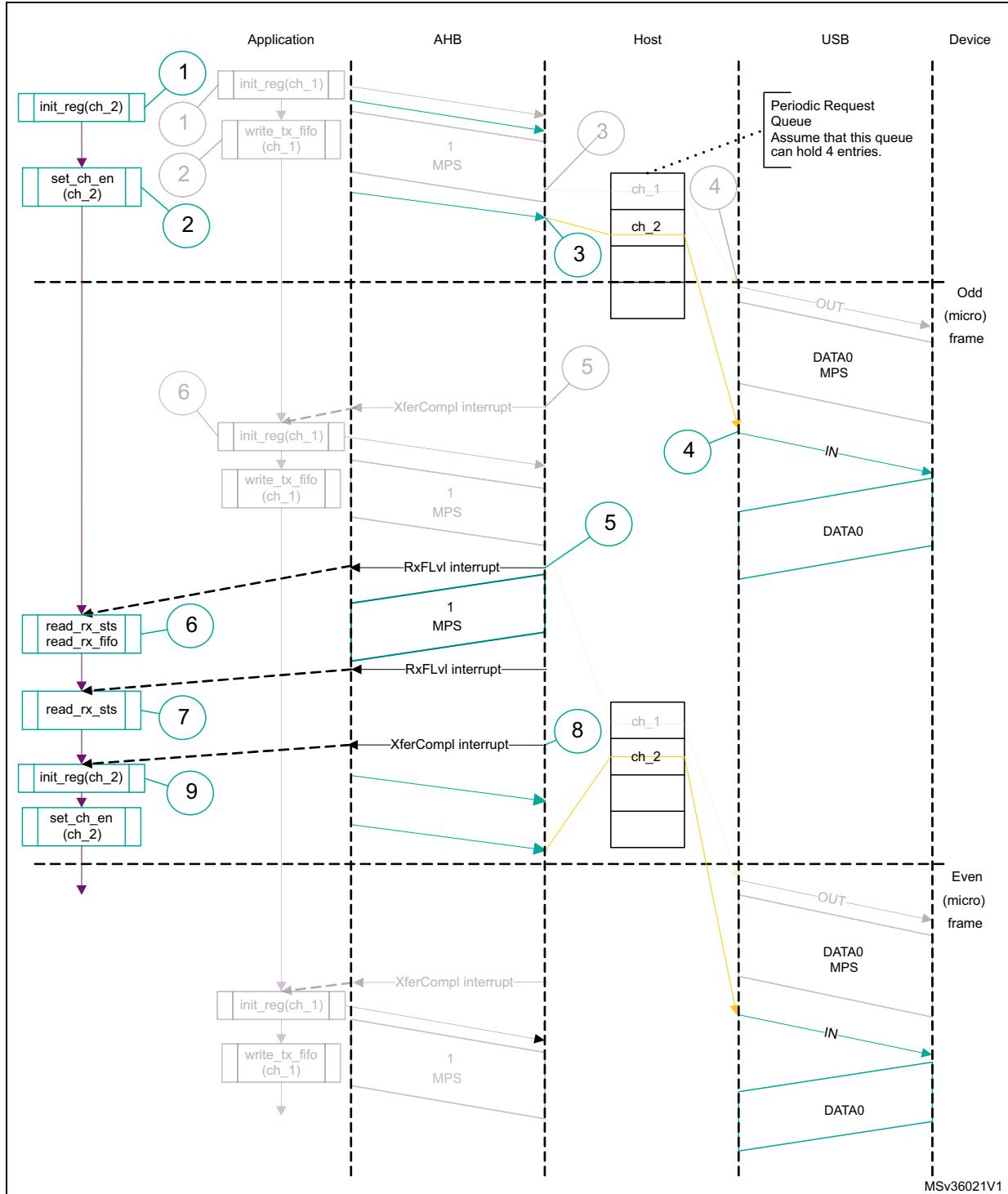
The assumptions are:

- 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 471. Isochronous IN transactions



1. The grayed elements are not relevant in the context of this figure.

- **Selecting the queue depth**

Choose the periodic and non-periodic request queue depths carefully to match the number of periodic/non-periodic endpoints accessed.

The non-periodic request queue depth affects the performance of non-periodic

transfers. The deeper the queue (along with sufficient FIFO size), the more often the core is able to pipeline non-periodic transfers. If the queue size is small, the core is able to put in new requests only when the queue space is freed up.

The core's periodic request queue depth is critical to perform periodic transfers as scheduled. Select the periodic queue depth, based on the number of periodic transfers scheduled in a microframe. If the periodic request queue depth is smaller than the periodic transfers scheduled in a microframe, a frame overrun condition occurs.

- **Handling babble conditions**

OTG\_FS/OTG\_HS controller handles two cases of babble: packet babble and port babble. Packet babble occurs if the device sends more data than the maximum packet size for the channel. Port babble occurs if the core continues to receive data from the device at EOF2 (the end of frame 2, which is very close to SOF).

When OTG\_FS/OTG\_HS controller detects a packet babble, it stops writing data into the Rx buffer and waits for the end of packet (EOP). When it detects an EOP, it flushes already written data in the Rx buffer and generates a Babble interrupt to the application.

When OTG\_FS/OTG\_HS controller detects a port babble, it flushes the Rx FIFO and disables the port. The core then generates a port disabled interrupt (HPRTINT in OTG\_GINTSTS, PENCHNG in OTG\_HPRT). On receiving this interrupt, the application must determine that this is not due to an overcurrent condition (another cause of the port disabled interrupt) by checking POCA in OTG\_HPRT, then perform a soft reset. The core does not send any more tokens after it has detected a port babble condition.

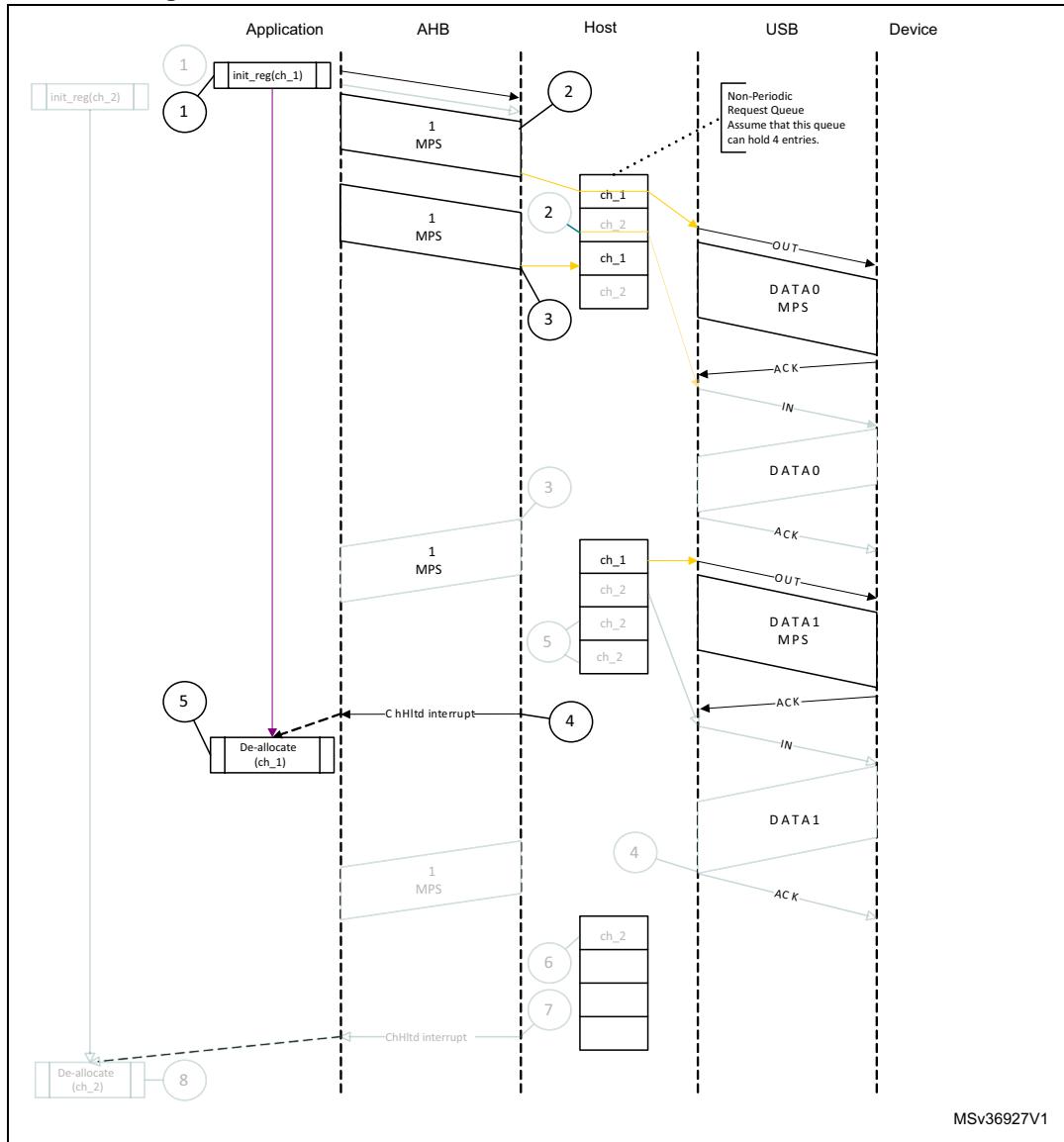
*Note:* The following paragraphs, ranging from here to the beginning of [Section 37.16](#), and covering DMA configurations, apply only to USB OTG HS.

- **Bulk and control OUT/SETUP transactions in DMA mode**

The sequence of operations is as follows:

1. Initialize and enable channel 1 as explained in [Section : Channel initialization](#).
2. The OTG\_HS host starts fetching the first packet as soon as the channel is enabled. For internal DMA mode, the OTG\_HS host uses the programmed DMA address to fetch the packet.
3. After fetching the last 32-bit word of the second (last) packet, the OTG\_HS host masks channel 1 internally for further arbitration.
4. The OTG\_HS host generates a CHH interrupt as soon as the last packet is sent.
5. In response to the CHH interrupt, de-allocate the channel for other transfers.

Figure 472. 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.

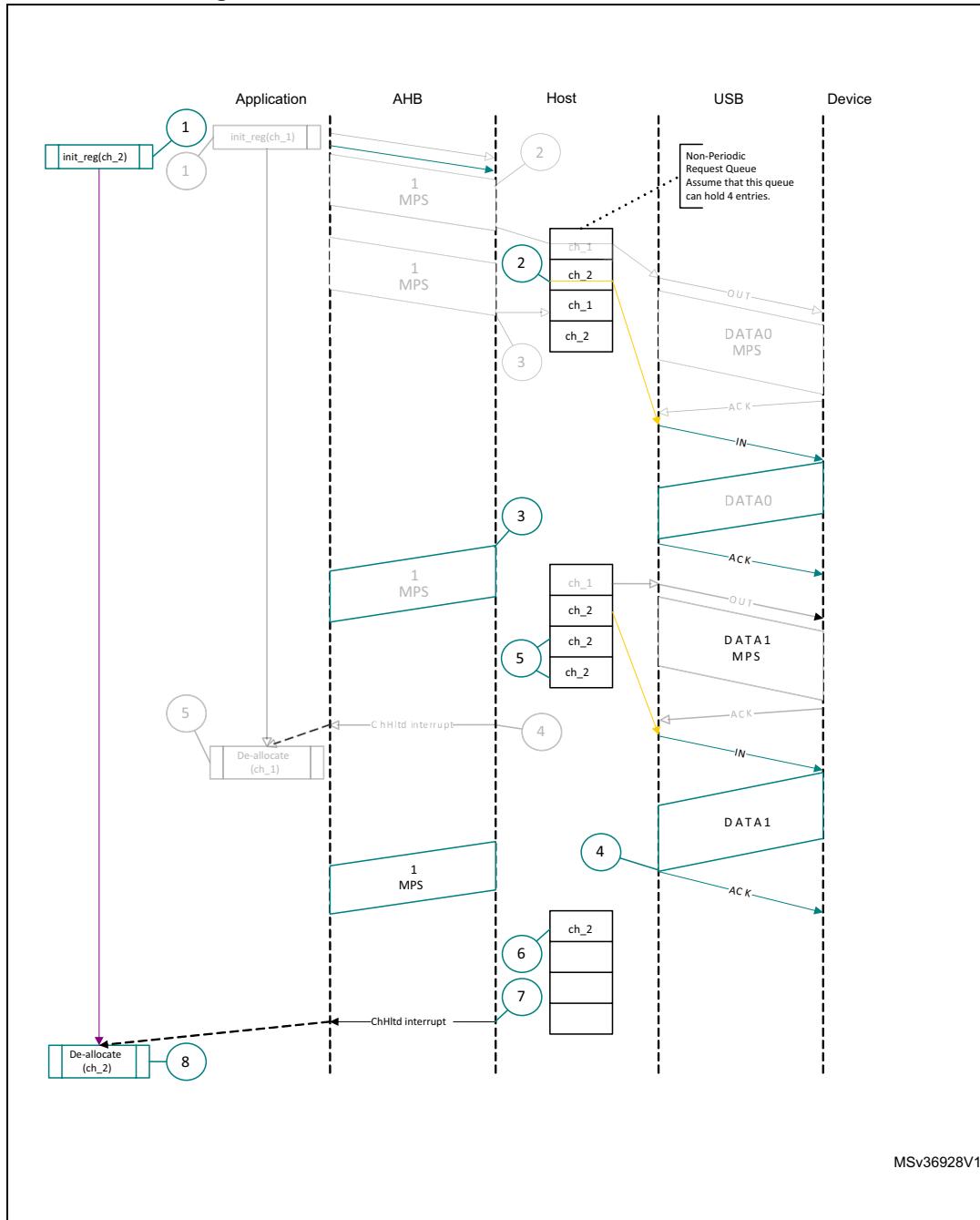
The core does not generate a separate interrupt when NAK or NYET is received by the host functionality.

- **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 473. Normal bulk/control IN transaction - DMA

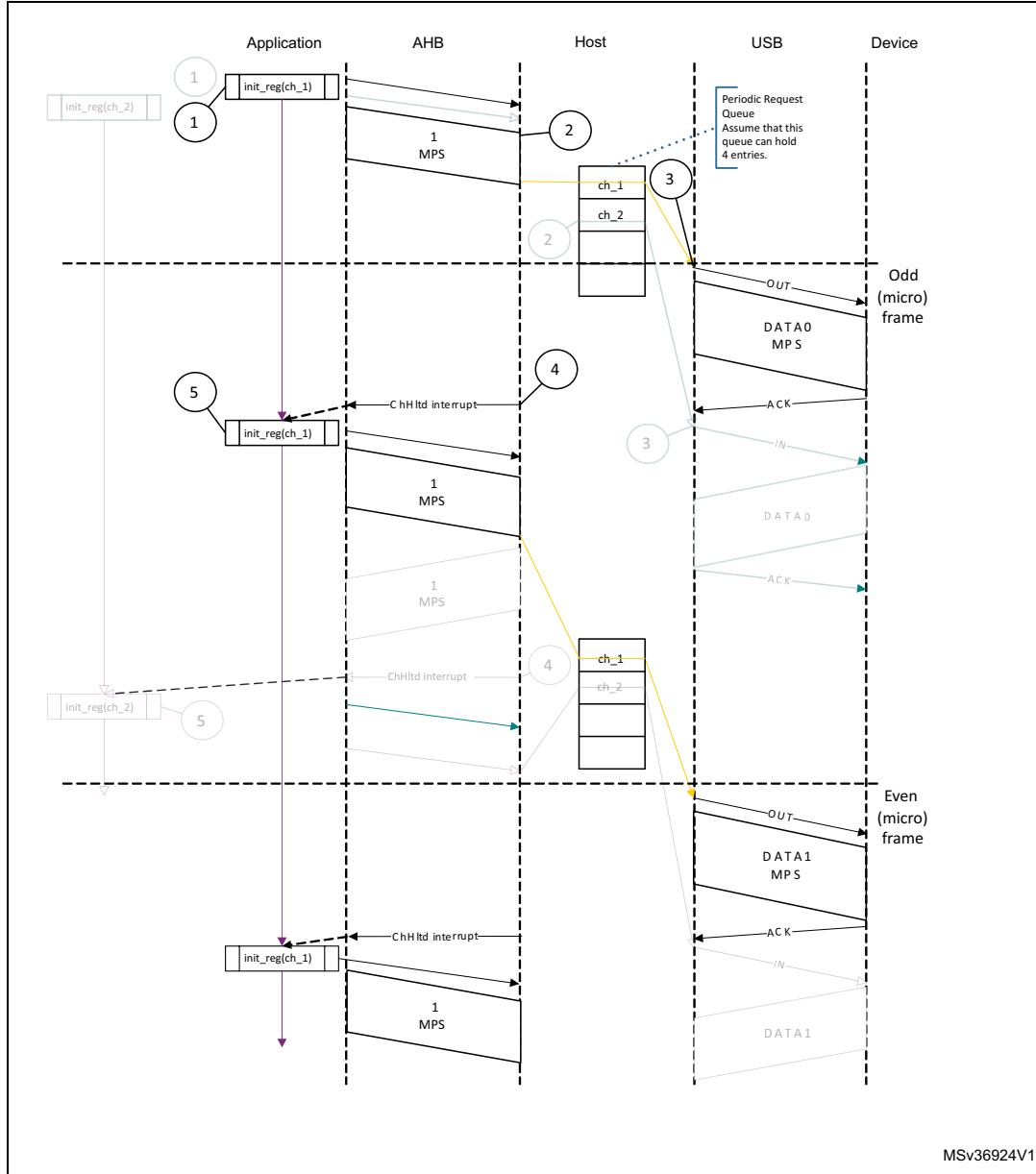


- **Interrupt OUT transactions in DMA mode**

1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG\_HS host starts fetching the first packet as soon the channel is enabled and writes the OUT request along with the last 32-bit word fetch. In high-bandwidth transfers, the OTG\_HS host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.
3. The OTG\_HS host attempts to send the OUT token at the beginning of the next odd frame/micro-frame.

4. After successfully transmitting the packet, the OTG\_HS host generates a CHH interrupt.
5. In response to the CHH interrupt, reinitialize the channel for the next transfer.

**Figure 474. Normal interrupt OUT transactions - DMA mode**

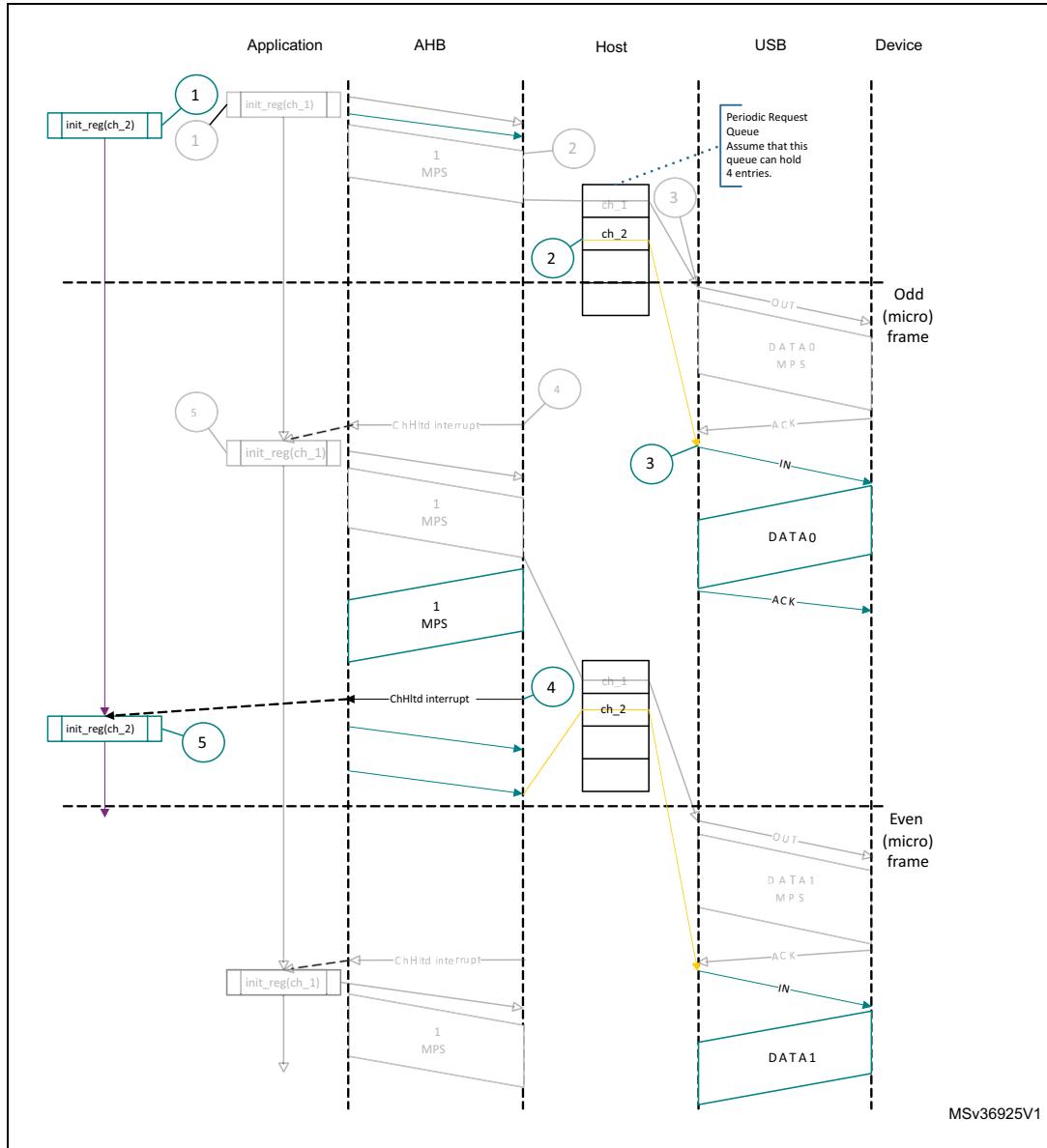


- **Interrupt IN transactions in DMA mode**

- 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 475. Normal interrupt IN transactions - DMA mode**



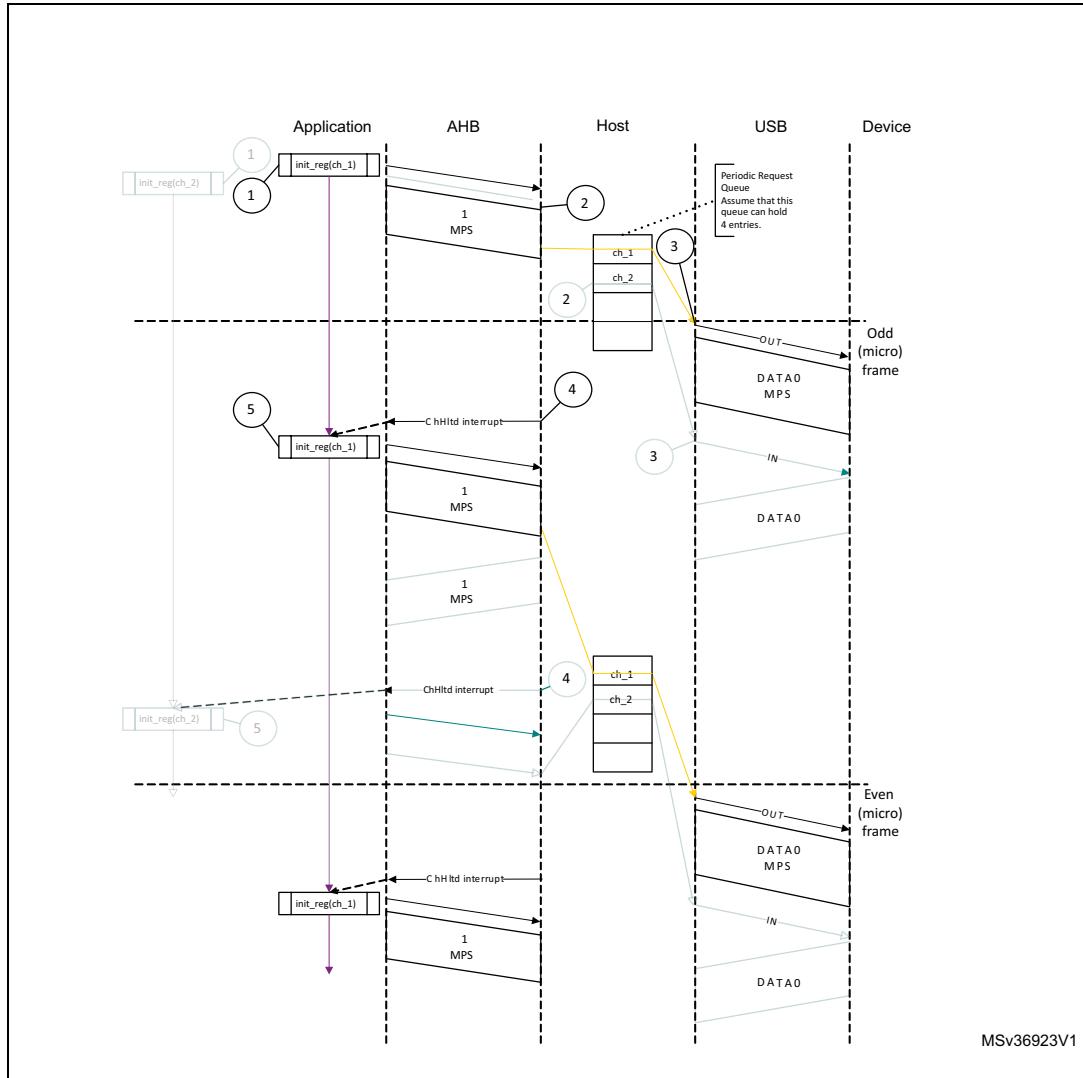
#### • Isochronous OUT transactions in DMA mode

1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG\_HS host starts fetching the first packet as soon as the channel is enabled, and writes the OUT request along with the last 32-bit word fetch. In high-bandwidth

transfers, the OTG\_HS host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.

3. The OTG\_HS host attempts to send an OUT token at the beginning of the next (odd) frame/micro-frame.
4. After successfully transmitting the packet, the OTG\_HS host generates a CHH interrupt.
5. In response to the CHH interrupt, reinitialize the channel for the next transfer.

**Figure 476. Normal isochronous OUT transaction - DMA mode**



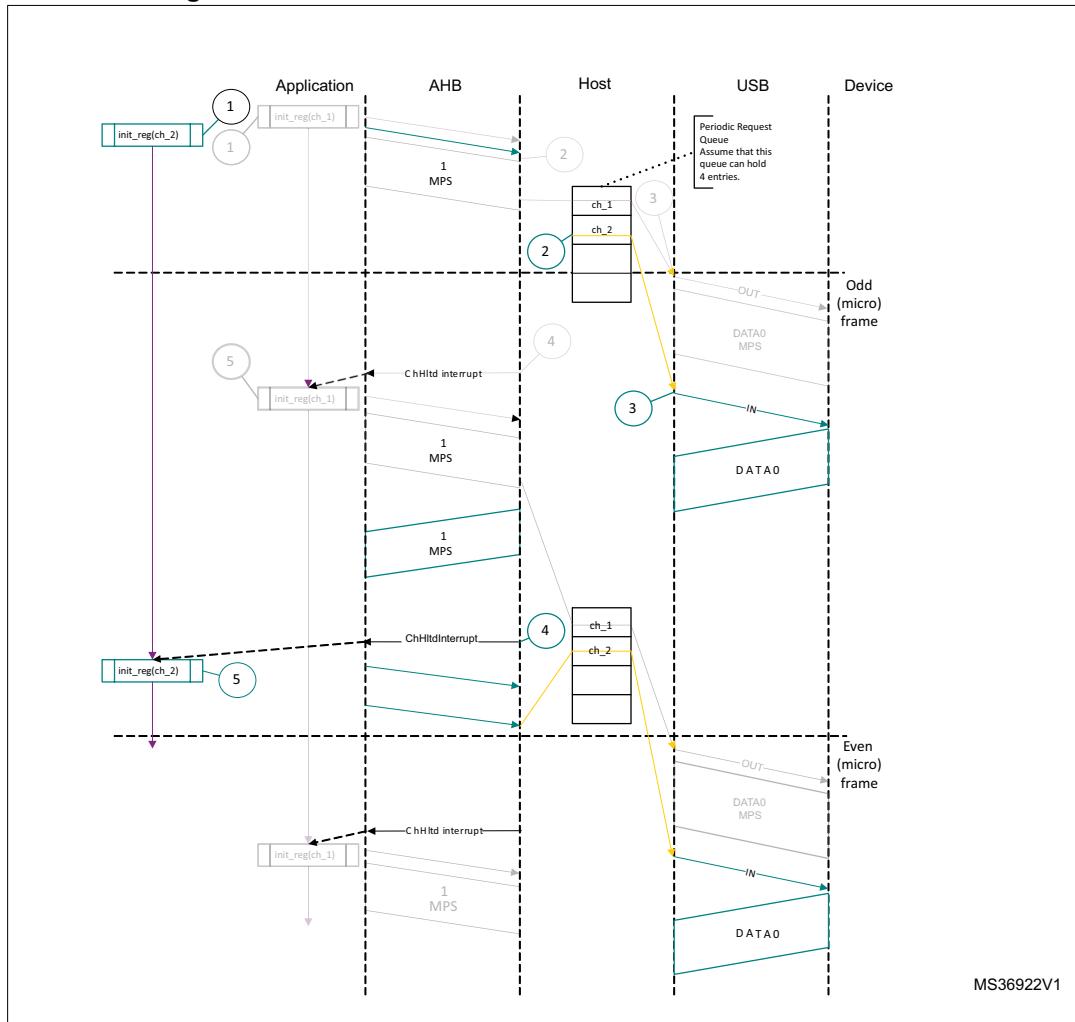
- **Isochronous IN transactions in DMA mode**

The sequence of operations ((channel x) is as follows:

1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG\_HS host writes an IN request to the request queue as soon as the channel x gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the OTG\_HS host performs consecutive write operations up to MC times.

3. The OTG\_HS host attempts to send an IN token at the beginning of the next (odd) frame/micro-frame.
4. As soon as the packet is received and written to the receive FIFO, the OTG\_HS host generates a CHH interrupt.
5. In response to the CHH interrupt, reinitialize the channel for the next transfer.

**Figure 477. Normal isochronous IN transactions - DMA mode**



- **Bulk and control OUT/SETUP split transactions in DMA mode**

The sequence of operations in (channel x) is as follows:

1. Initialize and enable channel x for start split as explained in [Section : Channel initialization](#).
2. The OTG\_HS host starts fetching the first packet as soon the channel is enabled and writes the OUT request along with the last 32-bit word fetch.
3. After successfully transmitting start split, the OTG\_HS host generates the CHH interrupt.
4. In response to the CHH interrupt, set the COMPLSPLT bit in OTG\_HCSPLT1 to send the complete split.

5. After successfully transmitting complete split, the OTG\_HS host generates the CHH interrupt.

6. In response to the CHH interrupt, de-allocate the channel.

- **Bulk/control IN split transactions in DMA mode**

The sequence of operations (channel x) is as follows:

1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG\_HS host writes the start split request to the nonperiodic request after getting the grant from the arbiter. The OTG\_HS host masks the channel x internally for the arbitration after writing the request.

3. As soon as the IN token is transmitted, the OTG\_HS host generates the CHH interrupt.

4. In response to the CHH interrupt, set the COMPLSPLT bit in OTG\_HCSPLT2 and re-enable the channel to send the complete split token. This unmasks channel x for arbitration.

5. The OTG\_HS host writes the complete split request to the nonperiodic request after receiving the grant from the arbiter.

6. The OTG\_HS host starts writing the packet to the system memory after receiving the packet successfully.

7. As soon as the received packet is written to the system memory, the OTG\_HS host generates a CHH interrupt.

8. In response to the CHH interrupt, de-allocate the channel.

- **Interrupt OUT split transactions in DMA mode**

The sequence of operations in (channel x) is as follows:

1. Initialize and enable channel 1 for start split as explained in [Section : Channel initialization](#). The application must set the ODDFRM bit in OTG\_HCCHAR1.

2. The OTG\_HS host starts reading the packet.

3. The OTG\_HS host attempts to send the start split transaction.

4. After successfully transmitting the start split, the OTG\_HS host generates the CHH interrupt.

5. In response to the CHH interrupt, set the COMPLSPLT bit in OTG\_HCSPLT1 to send the complete split.

6. After successfully completing the complete split transaction, the OTG\_HS host generates the CHH interrupt.

7. In response to CHH interrupt, de-allocate the channel.

- **Interrupt IN split transactions in DMA mode**

The sequence of operations in (channel x) is as follows:

1. Initialize and enable channel x for start split as explained in [Section : Channel initialization](#).

2. The OTG\_HS host writes an IN request to the request queue as soon as channel x receives the grant from the arbiter.

3. The OTG\_HS host attempts to send the start split IN token at the beginning of the next odd micro-frame.

4. The OTG\_HS host generates the CHH interrupt after successfully transmitting the start split IN token.

5. In response to the CHH interrupt, set the COMPLSPLT bit in OTG\_HCSPLT2 to send the complete split.

6. As soon as the packet is received successfully, the OTG\_HS host starts writing the data to the system memory.
7. The OTG\_HS host generates the CHH interrupt after transferring the received data to the system memory.
8. In response to the CHH interrupt, de-allocate or reinitialize the channel for the next start split.

- **Isochronous OUT split transactions in DMA mode**

The sequence of operations (channel x) is as follows:

1. Initialize and enable channel x for start split (begin) as explained in [Section : Channel initialization](#). The application must set the ODDFRM bit in OTG\_HCCHAR1. Program the MPS field.
2. The OTG\_HS host starts reading the packet.
3. After successfully transmitting the start split (begin), the OTG\_HS host generates the CHH interrupt.
4. In response to the CHH interrupt, reinitialize the registers to send the start split (end).
5. After successfully transmitting the start split (end), the OTG\_HS host generates a CHH interrupt.
6. In response to the CHH interrupt, de-allocate the channel.

- **Isochronous IN split transactions in DMA mode**

The sequence of operations (channel x) is as follows:

1. Initialize and enable channel x for start split as explained in [Section : Channel initialization](#).
2. The OTG\_HS host writes an IN request to the request queue as soon as channel x receives the grant from the arbiter.
3. The OTG\_HS host attempts to send the start split IN token at the beginning of the next odd micro-frame.
4. The OTG\_HS host generates the CHH interrupt after successfully transmitting the start split IN token.
5. In response to the CHH interrupt, set the COMPLSPLT bit in OTG\_HCSPLT2 to send the complete split.
6. As soon as the packet is received successfully, the OTG\_HS host starts writing the data to the system memory.

The OTG\_HS host generates the CHH interrupt after transferring the received data to the system memory. In response to the CHH interrupt, de-allocate the channel or reinitialize the channel for the next start split.

### 37.16.6 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.

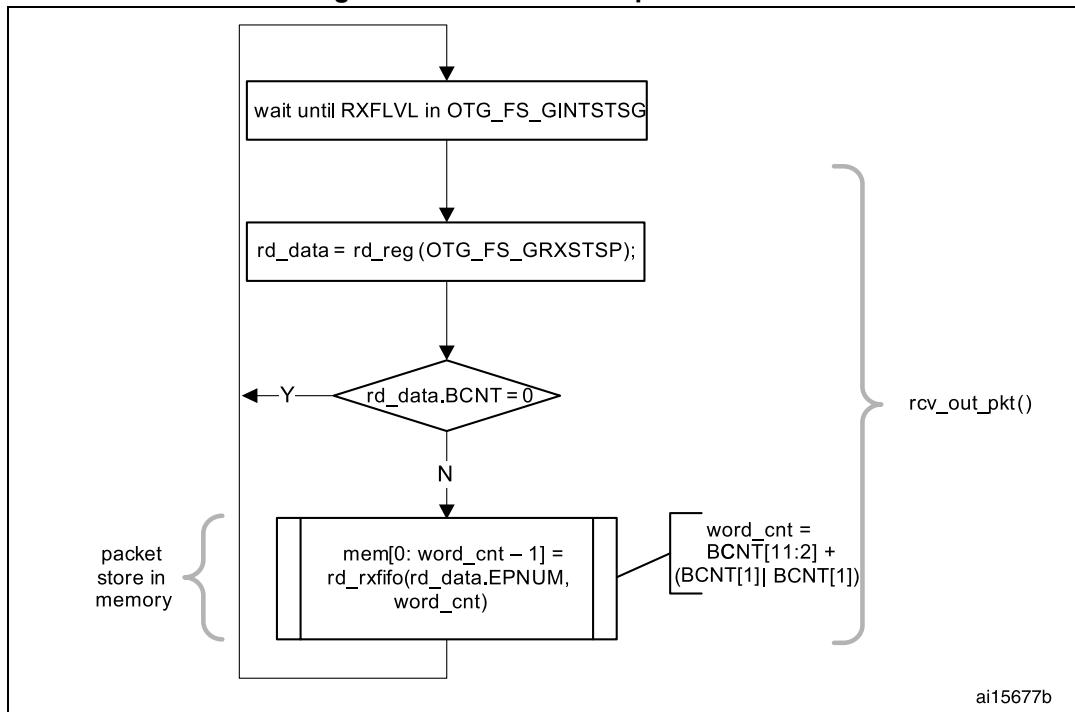
- **Packet read**

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 478* provides a flowchart of the above procedure.

**Figure 478. 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.

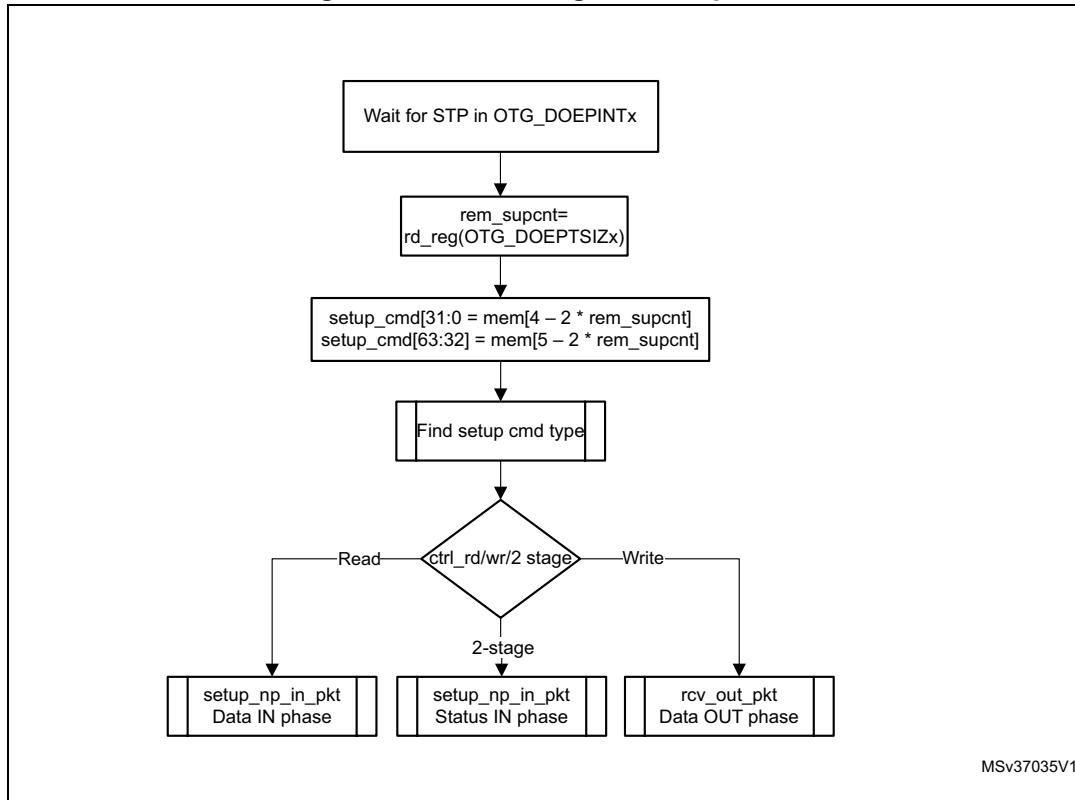
- **Internal data flow**

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 479. 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

- **Transfer Stop Programming for OUT endpoints**

The application must use the following programing sequence to stop any transfers (because of an interrupt from the host, typically a reset).

**Sequence of operations:**

1. Enable all OUT endpoints by setting
  - EPENA = 1 in all OTG\_DOEPCTLx registers.
2. Flush the RxFIFO as follows
  - Poll OTG\_GRSTCTL.AHBIDL until it is 1. This indicates that AHB master is idle.
  - Perform read modify write operation on OTG\_GRSTCTL.RXFFLSH =1
  - Poll OTG\_GRSTCTL.RXFFLSH until it is 0, but also using a timeout of less than 10 milli-seconds (corresponds to minimum reset signaling duration). If 0 is seen before the timeout, then the RxFIFO flush is successful. If at the moment the timeout occurs, there is still a 1, (this may be due to a packet on EP0 coming from the host) then go back (once only) to the previous step (“Perform read modify write operation”).
3. Before disabling any OUT endpoint, the application must enable Global OUT NAK mode in the core, according to the instructions in “[Setting the global OUT NAK on page 1492](#)”. This ensures that data in the RxFIFO is sent to the application successfully. Set SGONAK = 1 in OTG\_DCTL
4. Wait for the GONAKEFF interrupt (OTG\_GINTSTS)
5. Disable all active OUT endpoints by programming the following register bits:
  - EPDIS = 1 in registers OTG\_DOEPCTLx
  - SNAK = 1 in registers OTG\_DOEPCTLx
6. Wait for the EPDIS interrupt in OTG\_DOEPINTx for each OUT endpoint programmed in the previous step. The EPDIS interrupt in OTG\_DOEPINTx indicates that the corresponding OUT endpoint is completely disabled. When the EPDIS interrupt is asserted, the following bits are cleared:
  - EPENA = 0 in registers OTG\_DOEPCTLx
  - EPDIS = 0 in registers OTG\_DOEPCTLx
  - SNAK = 0 in registers OTG\_DOEPCTLx

- **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 may fail to detect the XFRC interrupt (OTG\_DOEPINTx) under the following circumstances:
  - When the receive FIFO cannot accommodate the complete ISO OUT data packet, the core drops the received ISO OUT data
  - When the isochronous OUT data packet is received with CRC errors
  - When the isochronous OUT token received by the core is corrupted
  - When the application is very slow in reading the data from the receive FIFO
2. When the core detects an end of periodic frame before transfer completion to all isochronous OUT endpoints, it asserts the incomplete isochronous OUT data interrupt (INCOMPISOOUT in OTG\_GINTSTS), indicating that an XFRC interrupt (in OTG\_DOEPINTx) is not asserted on at least one of the isochronous OUT endpoints. At

At this point, the endpoint with the incomplete transfer remains enabled, but no active transfers remain in progress on this endpoint on the USB.

Application programming sequence:

1. Asserting the INCOMPISOOUT interrupt (OTG\_GINTSTS) indicates that in the current frame, at least one isochronous OUT endpoint has an incomplete transfer.
2. If this occurs because isochronous OUT data is not completely emptied from the endpoint, the application must ensure that the application empties all isochronous OUT data (data and status) from the receive FIFO before proceeding.
  - When all data are emptied from the receive FIFO, the application can detect the XFRC interrupt (OTG\_DOEPINTx). In this case, the application must re-enable the endpoint to receive isochronous OUT data in the next frame.
3. When it receives an INCOMPISOOUT interrupt (in OTG\_GINTSTS), the application must read the control registers of all isochronous OUT endpoints (OTG\_DOEPCTLx) to determine which endpoints had an incomplete transfer in the current microframe. An endpoint transfer is incomplete if both the following conditions are met:
  - EONUM bit (in OTG\_DOEPCTLx) = FNSOF[0] (in OTG\_DSTS)
  - EPENA = 1 (in OTG\_DOEPCTLx)
4. The previous step must be performed before the SOF interrupt (in OTG\_GINTSTS) is detected, to ensure that the current frame number is not changed.
5. For isochronous OUT endpoints with incomplete transfers, the application must discard the data in the memory and disable the endpoint by setting the EPDIS bit in OTG\_DOEPCTLx.
6. Wait for the EPDISD interrupt (in OTG\_DOEPINTx) and enable the endpoint to receive new data in the next frame.
  - Because the core can take some time to disable the endpoint, the application may not be able to receive the data in the next frame after receiving bad isochronous data.

- **Stalling a non-isochronous OUT endpoint**

This section describes how the application can stall a non-isochronous endpoint.

1. Put the core in the Global OUT NAK mode.
2. Disable the required endpoint
  - When disabling the endpoint, instead of setting the SNAK bit in OTG\_DOEPCTL, set STALL = 1 (in OTG\_DOEPCTL).  
The STALL bit always takes precedence over the NAK bit.
3. When the application is ready to end the STALL handshake for the endpoint, the STALL bit (in OTG\_DOEPCTLx) must be cleared.
4. If the application is setting or clearing a STALL for an endpoint due to a SetFeature.Endpoint Halt or ClearFeature.Endpoint Halt command, the STALL bit must be set or cleared before the application sets up the status stage transfer on the control endpoint.

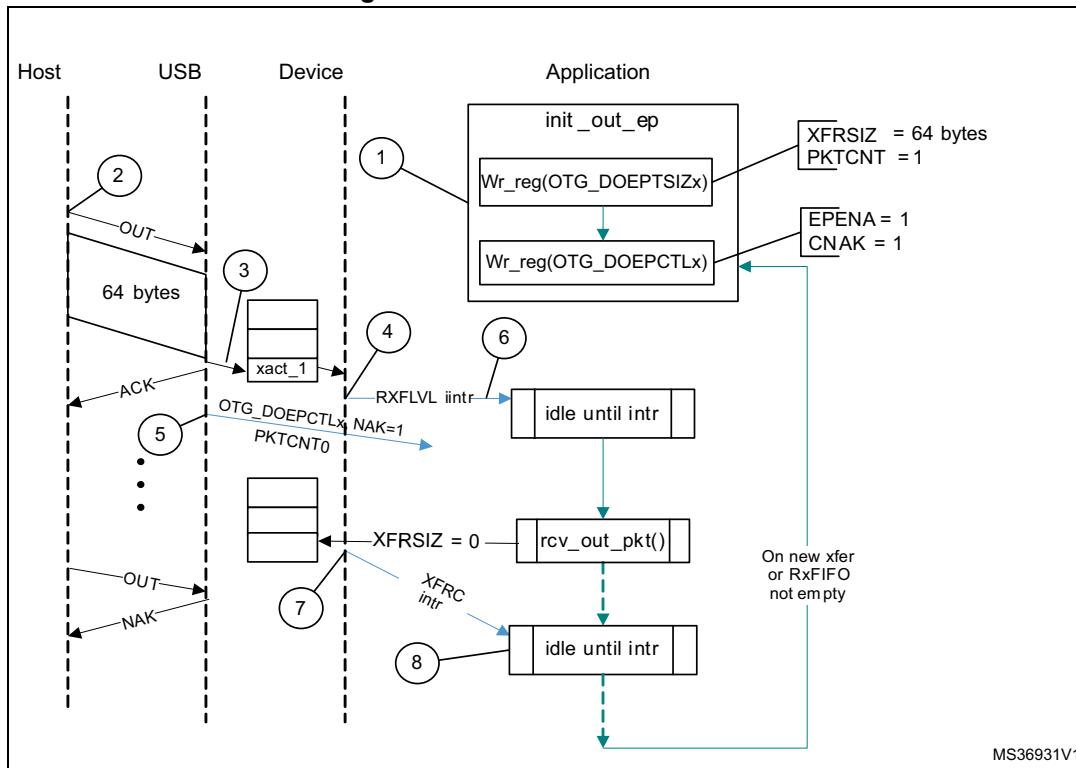
## Examples

This section describes and depicts some fundamental transfer types and scenarios.

- Bulk OUT transaction

*Figure 480* 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 480. 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.

- **Transfer Stop Programming for IN endpoints**

The application must use the following programing sequence to stop any transfers (because of an interrupt from the host, typically a reset).

**Sequence of operations:**

1. Disable the IN endpoint by setting:
  - EPDIS = 1 in all OTG\_DIEPCTLx registers
2. Wait for the EPDIS interrupt in OTG\_DIEPINTx, which indicates that the IN endpoint is completely disabled. When the EPDIS interrupt is asserted the following bits are cleared:
  - EPDIS = 0 in OTG\_DIEPCTLx
  - EPENA = 0 in OTG\_DIEPCTLx
3. Flush the TxFIFO by programming the following bits:
  - TXFFLSH = 1 in OTG\_GRSTCTL
  - TXFNUM = “FIFO number specific to endpoint” in OTG\_GRSTCTL
4. The application can start polling till TXFFLSH in OTG\_GRSTCTL is cleared. When this bit is cleared, it ensures that there is no data left in the Tx FIFO.

- **Generic non-periodic IN data transfers**

## Application requirements:

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer are part of a single buffer.
2. For IN transfers, the transfer size field in the endpoint transfer size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.
  - To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:  
 Transfer size[EPNUM] =  $x \times \text{MPSIZ}[EPNUM] + sp$   
 If (sp > 0), then packet count[EPNUM] =  $x + 1$ .  
 Otherwise, packet count[EPNUM] =  $x$
  - To transmit a single zero-length data packet:  
 Transfer size[EPNUM] = 0  
 Packet count[EPNUM] = 1
  - To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer into two parts. The first sends maximum-packet-size data packets and the second sends the zero-length data packet alone.  
 First transfer: transfer size[EPNUM] =  $x \times \text{MPSIZ}[epnum]$ ; packet count =  $n$ ;  
 Second transfer: transfer size[EPNUM] = 0; packet count = 1;
3. Once an endpoint is enabled for data transfers, the core updates the transfer size register. At the end of the IN transfer, the application must read the transfer size register to determine how much data posted in the transmit FIFO have already been sent on the USB.
4. Data fetched into transmit FIFO = Application-programmed initial transfer size – core-updated final transfer size
  - Data transmitted on USB = (application-programmed initial packet count – core updated final packet count)  $\times \text{MPSIZ}[EPNUM]$
  - 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 1502](#) 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\_DIEPTSIZx 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\_DIEPTSIZx 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.

### 37.16.7 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 may 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).

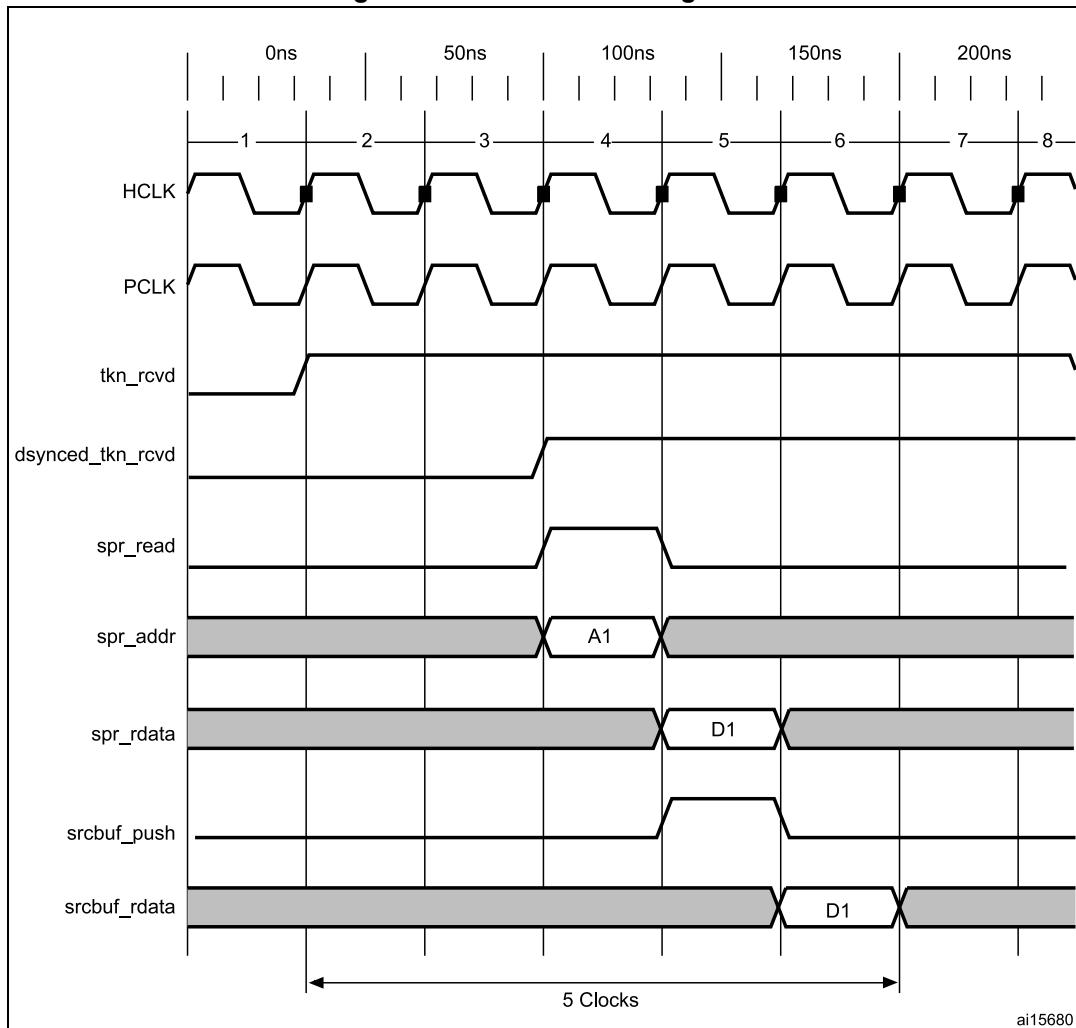
If the AHB is running at a higher frequency than the PHY, the application can use a smaller value for TRDT (in OTG\_GUSBCFG).

*Figure 481* 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 273: TRDT values \(FS\)](#) or [Table 274: TRDT values \(HS\)](#).

Figure 481. TRDT max timing case



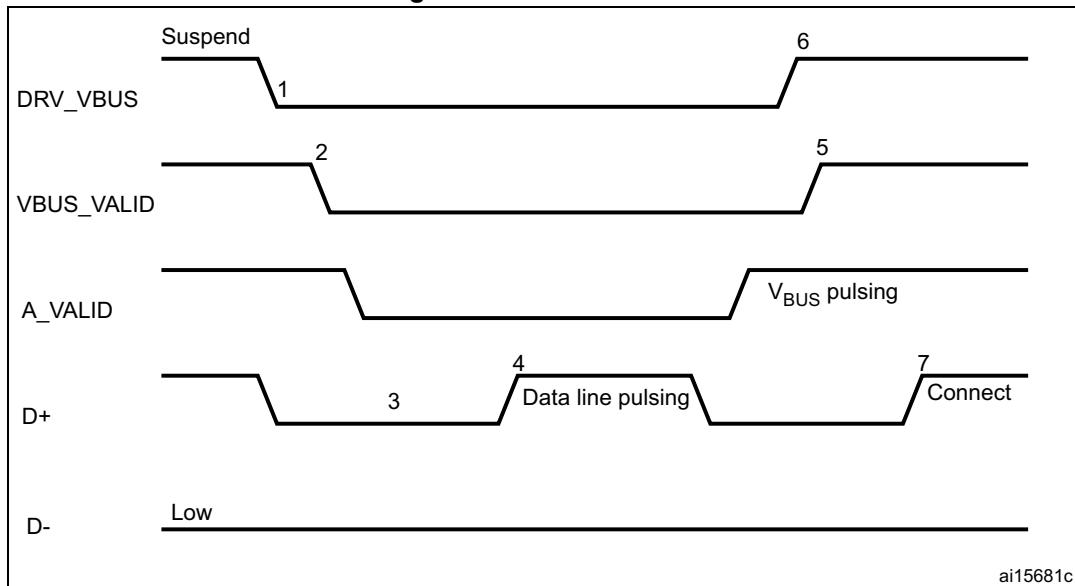
### 37.16.8 OTG programming model

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 482. A-device SRP



1.  $\text{DRV\_VBUS} = V_{\text{BUS}}$  drive signal to the PHY  
 $\text{VBUS\_VALID} = V_{\text{BUS}}$  valid signal from PHY  
 $\text{A\_VALID}$  = A-peripheral  $V_{\text{BUS}}$  level signal to PHY  
 $\text{D}^+$  = Data plus line  
 $\text{D}^-$  = Data minus line

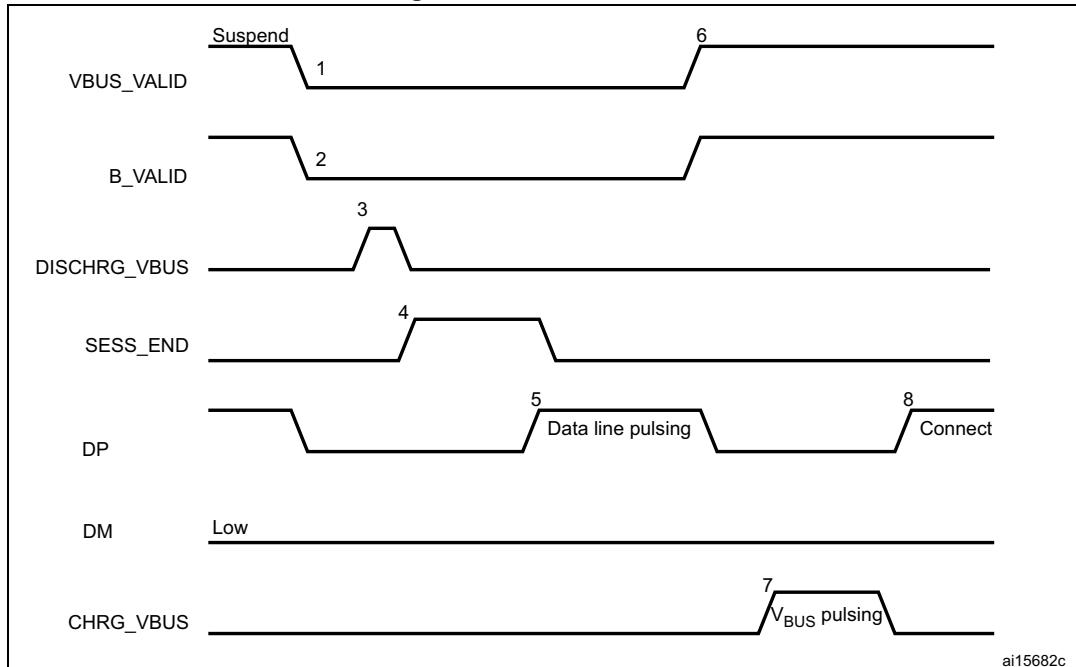
The following points refer and describe the signal numeration shown in the [Figure 482](#):

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 483. B-device SRP



1. VBUS\_VALID = V<sub>BUS</sub> 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<sub>BUS</sub> signal to PHY  
DP = Data plus line  
DM = Data minus line

The following points refer and describe the signal numeration shown in the [Figure 483](#):

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<sub>BUS</sub>.
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<sub>BUS</sub> 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<sub>BUS</sub> 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<sub>BUS</sub> pulsing.
5. The host detects SRP from either the data-line or V<sub>BUS</sub> pulsing, and turns on V<sub>BUS</sub>. The PHY indicates V<sub>BUS</sub> power-on to the device.
6. The OTG\_FS/OTG\_HS controller performs V<sub>BUS</sub> pulsing. The host starts a new session by turning on V<sub>BUS</sub>, indicating SRP success. The OTG\_FS/OTG\_HS controller interrupts the application by setting the session request

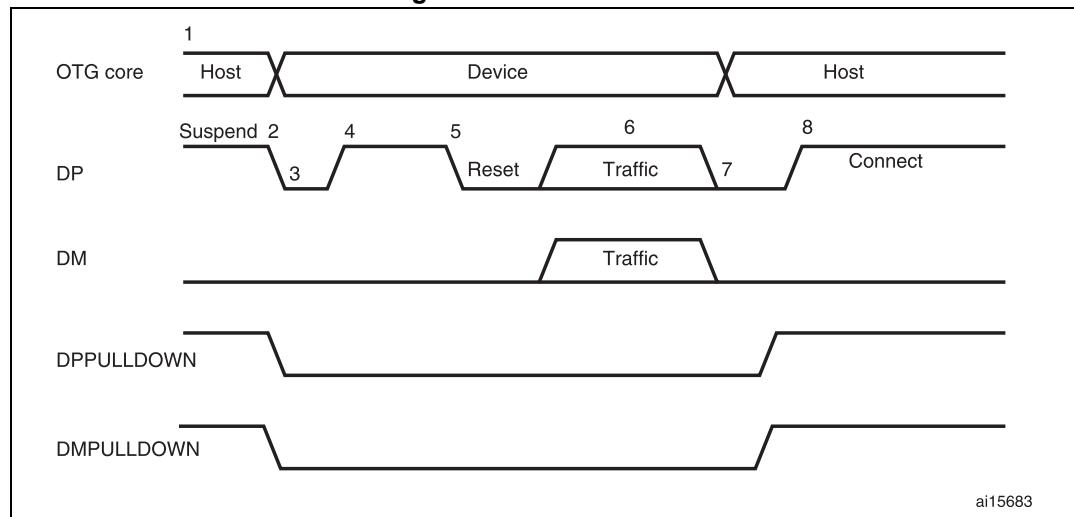
success status change bit in the OTG interrupt status register. The application reads the session request success bit in the OTG control and status register.

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 484. 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 484](#):

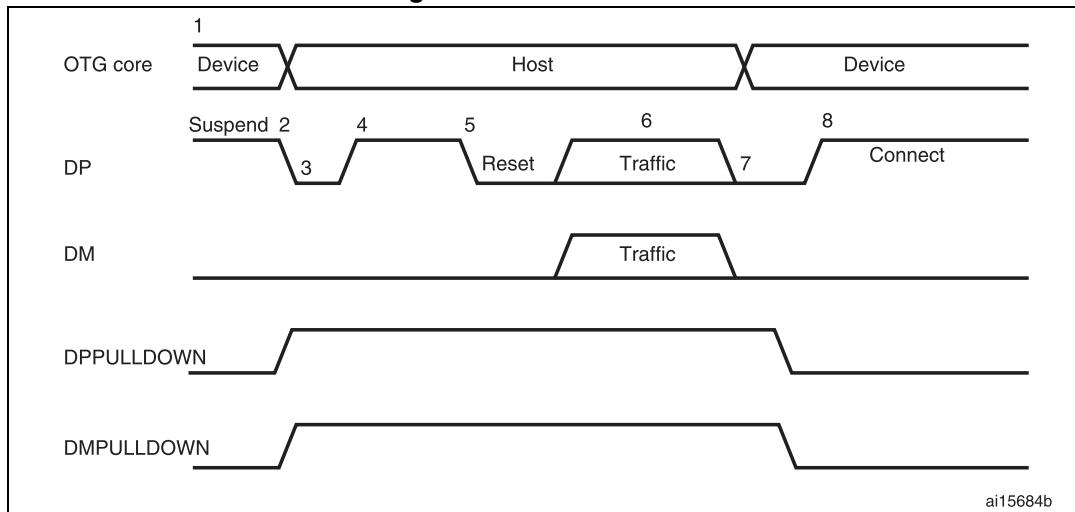
1. The OTG\_FS/OTG\_HS controller sends the B-device a SetFeature b\_hnp\_enable descriptor to enable HNP support. The B-device's ACK response indicates that the B-device supports HNP. The application must set host Set HNP enable bit in the OTG

- 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 485. B-device HNP



1. DPPULLDOWN = signal from core to PHY to enable/disable the pull-down on the DP line inside the PHY.  
DMPULLDOWN = signal from core to PHY to enable/disable the pull-down on the DM line inside the PHY.

The following points refer and describe the signal numeration shown in the [Figure 485](#):

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.

## 38 Ethernet (ETH): media access control (MAC) with DMA controller

### 38.1 Ethernet introduction

Portions Copyright (c) 2004, 2005 Synopsys, Inc. All rights reserved. Used with permission.

The Ethernet peripheral enables the STM32F75xxx and STM32F74xxx to transmit and receive data over Ethernet in compliance with the IEEE 802.3-2002 standard.

The Ethernet provides a configurable, flexible peripheral to meet the needs of various applications and customers. It supports two industry standard interfaces to the external physical layer (PHY): the default media independent interface (MII) defined in the IEEE 802.3 specifications and the reduced media independent interface (RMII). It can be used in number of applications such as switches, network interface cards, etc.

The Ethernet is compliant with the following standards:

- IEEE 802.3-2002 for Ethernet MAC
- IEEE 1588-2008 standard for precision networked clock synchronization
- AMBA 2.0 for AHB Master/Slave ports
- RMII specification from RMII consortium

### 38.2 Ethernet main features

The Ethernet (ETH) peripheral includes the following features, listed by category:

### 38.2.1 MAC core features

- Supports 10/100 Mbit/s data transfer rates with external PHY interfaces
- IEEE 802.3-compliant MII interface to communicate with an external Fast Ethernet PHY
- Supports both full-duplex and half-duplex operations
  - Supports CSMA/CD Protocol for half-duplex operation
  - Supports IEEE 802.3x flow control for full-duplex operation
  - Optional forwarding of received pause control frames to the user application in full-duplex operation
  - Back-pressure support for half-duplex operation
  - Automatic transmission of zero-quanta pause frame on deassertion of flow control input in full-duplex operation
- Preamble and start-of-frame data (SFD) insertion in Transmit, and deletion in Receive paths
- Automatic CRC and pad generation controllable on a per-frame basis
- Options for automatic pad/CRC stripping on receive frames
- Programmable frame length to support Standard frames with sizes up to 16 KB
- Programmable interframe gap (40-96 bit times in steps of 8)
- Supports a variety of flexible address filtering modes:
  - Up to four 48-bit perfect (DA) address filters with masks for each byte
  - Up to three 48-bit SA address comparison check with masks for each byte
  - 64-bit Hash filter (optional) for multicast and unicast (DA) addresses
  - Option to pass all multicast addressed frames
  - Promiscuous mode support to pass all frames without any filtering for network monitoring
  - Passes all incoming packets (as per filter) with a status report
- Separate 32-bit status returned for transmission and reception packets
- Supports IEEE 802.1Q VLAN tag detection for reception frames
- Separate transmission, reception, and control interfaces to the Application
- Supports mandatory network statistics with RMON/MIB counters (RFC2819/RFC2665)
- MDIO interface for PHY device configuration and management
- Detection of LAN wakeup frames and AMD Magic Packet™ frames
- Receive feature for checksum off-load for received IPv4 and TCP packets encapsulated by the Ethernet frame
- Enhanced receive feature for checking IPv4 header checksum and TCP, UDP, or ICMP checksum encapsulated in IPv4 or IPv6 datagrams
- Support Ethernet frame time stamping as described in IEEE 1588-2008. Sixty-four-bit time stamps are given in each frame's transmit or receive status
- Two sets of FIFOs: a 2-KB Transmit FIFO with programmable threshold capability, and a 2-KB Receive FIFO with a configurable threshold (default of 64 bytes)
- Receive Status vectors inserted into the Receive FIFO after the EOF transfer enables multiple-frame storage in the Receive FIFO without requiring another FIFO to store those frames' Receive Status
- Option to filter all error frames on reception and not forward them to the application in

Store-and-Forward mode

- Option to forward under-sized good frames
- Supports statistics by generating pulses for frames dropped or corrupted (due to overflow) in the Receive FIFO
- Supports Store and Forward mechanism for transmission to the MAC core
- Automatic generation of PAUSE frame control or back pressure signal to the MAC core based on Receive FIFO-fill (threshold configurable) level
- Handles automatic retransmission of Collision frames for transmission
- Discards frames on late collision, excessive collisions, excessive deferral and underrun conditions
- Software control to flush Tx FIFO
- Calculates and inserts IPv4 header checksum and TCP, UDP, or ICMP checksum in frames transmitted in Store-and-Forward mode
- Supports internal loopback on the MII for debugging

### 38.2.2 DMA features

- Supports all AHB burst types in the AHB Slave Interface
- Software can select the type of AHB burst (fixed or indefinite burst) in the AHB Master interface.
- Option to select address-aligned bursts from AHB master port
- Optimization for packet-oriented DMA transfers with frame delimiters
- Byte-aligned addressing for data buffer support
- Dual-buffer (ring) or linked-list (chained) descriptor chaining
- Descriptor architecture, allowing large blocks of data transfer with minimum CPU intervention;
- each descriptor can transfer up to 8 KB of data
- Comprehensive status reporting for normal operation and transfers with errors
- Individual programmable burst size for Transmit and Receive DMA Engines for optimal host bus utilization
- Programmable interrupt options for different operational conditions
- Per-frame Transmit/Receive complete interrupt control
- Round-robin or fixed-priority arbitration between Receive and Transmit engines
- Start/Stop modes
- Current Tx/Rx Buffer pointer as status registers
- Current Tx/Rx Descriptor pointer as status registers

### 38.2.3 PTP features

- Received and transmitted frames time stamping
- Coarse and fine correction methods
- Trigger interrupt when system time becomes greater than target time
- Pulse per second output (product alternate function output)

### 38.3 Ethernet pins

The table below shows the MAC signals and the corresponding MII/RMII signal mapping. All MAC signals are mapped onto AF11, some signals are mapped onto different I/O pins, and must be configured in Alternate function mode (for more details, refer to [Section 6.3.2: I/O pin alternate function multiplexer and mapping](#) and the corresponding datasheet).

**Table 277. Alternate function mapping**

AF11
<b>ETH</b>
ETH_MII_CRS
ETH_MII_RX_CLK / ETH_RMII_REF_CLK
ETH_MDIO
ETH_MII_COL
ETH_MII_RX_DV / ETH_RMII_CRS_DV
ETH_MII_RXD2
ETH_MII_RXD3
ETH_PPS_OUT
ETH_MII_TXD3
ETH_MII_RX_ER
ETH_MII_TX_EN / ETH_RMII_TX_EN
ETH_MII_TXD0 / ETH_RMII_TXD0
ETH_MII_TXD1 / ETH_RMII_TXD1
ETH_MDC
ETH_MII_TXD2
ETH_MII_TX_CLK
ETH_MII_RXD0 / ETH_RMII_RXD0
ETH_MII_RXD1 / ETH_RMII_RXD1
ETH_MII_TXD3
ETH_PPS_OUT
ETH_MII_TX_EN / ETH_RMII_TX_EN
ETH_MII_TXD0 / ETH_RMII_TXD0
ETH_MII_TXD1 / ETH_RMII_TXD1
ETH_MII_CRS
ETH_MII_COL
ETH_MII_RXD2
ETH_MII_RXD3
ETH_MII_RX_ER

## 38.4 Ethernet functional description: SMI, MII and RMII

The Ethernet peripheral consists of a MAC 802.3 (media access control) with a dedicated DMA controller. It supports both default media-independent interface (MII) and reduced media-independent interface (RMII) through one selection bit (refer to SYSCFG\_PMC register).

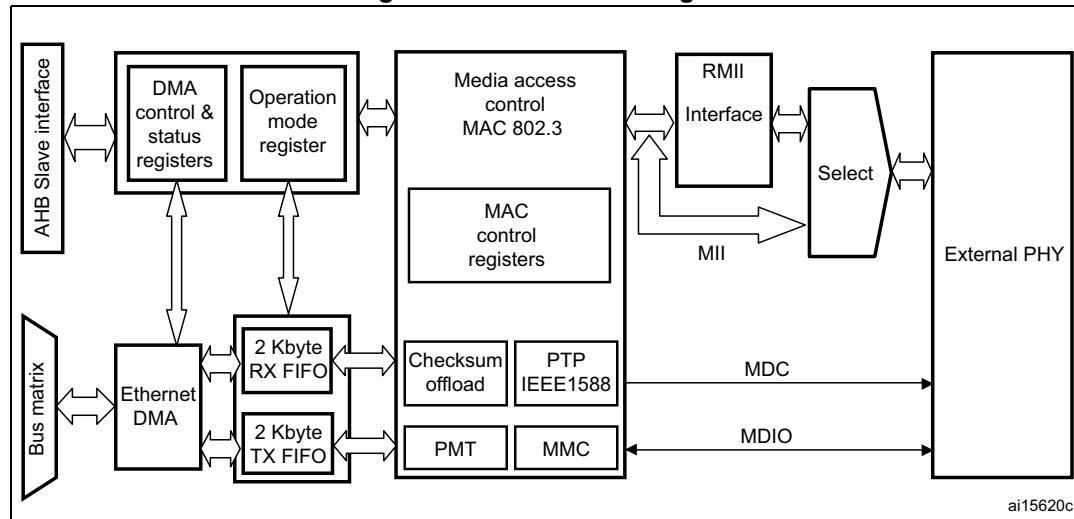
The DMA controller interfaces with the Core and memories through the AHB Master and Slave interfaces. The AHB Master Interface controls data transfers while the AHB Slave interface accesses Control and Status Registers (CSR) space.

The Transmit FIFO (Tx FIFO) buffers data read from system memory by the DMA before transmission by the MAC Core. Similarly, the Receive FIFO (Rx FIFO) stores the Ethernet frames received from the line until they are transferred to system memory by the DMA.

The Ethernet peripheral also includes an SMI to communicate with external PHY. A set of configuration registers permit the user to select the desired mode and features for the MAC and the DMA controller.

*Note:* The AHB clock frequency must be at least 25 MHz when the Ethernet is used.

Figure 486. ETH block diagram



1. For AHB connections refer to [Figure 1: System architecture for STM32F75xxx and STM32F74xxx devices](#).

### 38.4.1 Station management interface: SMI

The station management interface (SMI) allows the application to access any PHY registers through a 2-wire clock and data lines. The interface supports accessing up to 32 PHYs.

The application can select one of the 32 PHYs and one of the 32 registers within any PHY and send control data or receive status information. Only one register in one PHY can be addressed at any given time.

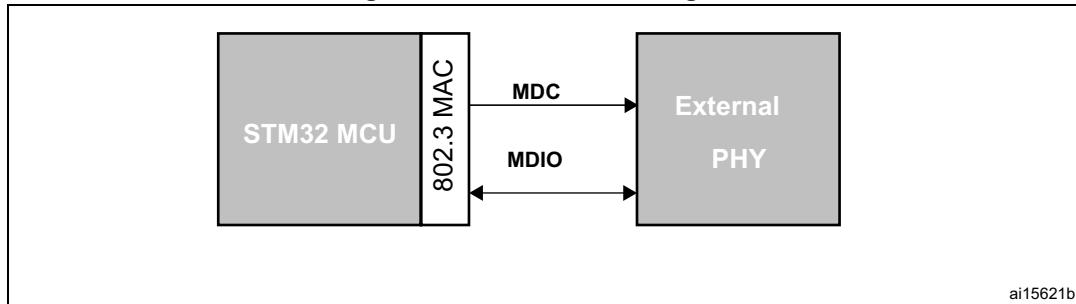
Both the MDC clock line and the MDIO data line are implemented as alternate function I/O in the microcontroller:

- MDC: a periodic clock that provides the timing reference for the data transfer at the maximum frequency of 2.5 MHz. The minimum high and low times for MDC must be

160 ns each, and the minimum period for MDC must be 400 ns. In idle state the SMI management interface drives the MDC clock signal low.

- MDIO: data input/output bitstream to transfer status information to/from the PHY device synchronously with the MDC clock signal

**Figure 487. SMI interface signals**



### SMI frame format

The frame structure related to a read or write operation is shown in [Table 278](#), the order of bit transmission must be from left to right.

**Table 278. Management frame format**

-	Management frame fields							
	Preamble (32 bits)	Start	Operation	PADDR	RADDR	TA	Data (16 bits)	Idle
Read	1... 1	01	10	ppppp	rrrrr	Z0	ddddddddddddd	Z
Write	1... 1	01	01	ppppp	rrrrr	10	ddddd	Z

The management frame consists of eight fields:

- **Preamble:** each transaction (read or write) can be initiated with the preamble field that corresponds to 32 contiguous logic one bits on the MDIO line with 32 corresponding cycles on MDC. This field is used to establish synchronization with the PHY device.
- **Start:** the start of frame is defined by a <01> pattern to verify transitions on the line from the default logic one state to zero and back to one.
- **Operation:** defines the type of transaction (read or write) in progress.
- **PADDR:** the PHY address is 5 bits, allowing 32 unique PHY addresses. The MSB bit of the address is the first transmitted and received.
- **RADDR:** the register address is 5 bits, allowing 32 individual registers to be addressed within the selected PHY device. The MSB bit of the address is the first transmitted and received.
- **TA:** the turn-around field defines a 2-bit pattern between the RADDR and DATA fields to avoid contention during a read transaction. For a read transaction the MAC controller drives high-impedance on the MDIO line for the 2 bits of TA. The PHY device must drive a high-impedance state on the first bit of TA, a zero bit on the second one.

For a write transaction, the MAC controller drives a <10> pattern during the TA field. The PHY device must drive a high-impedance state for the 2 bits of TA.

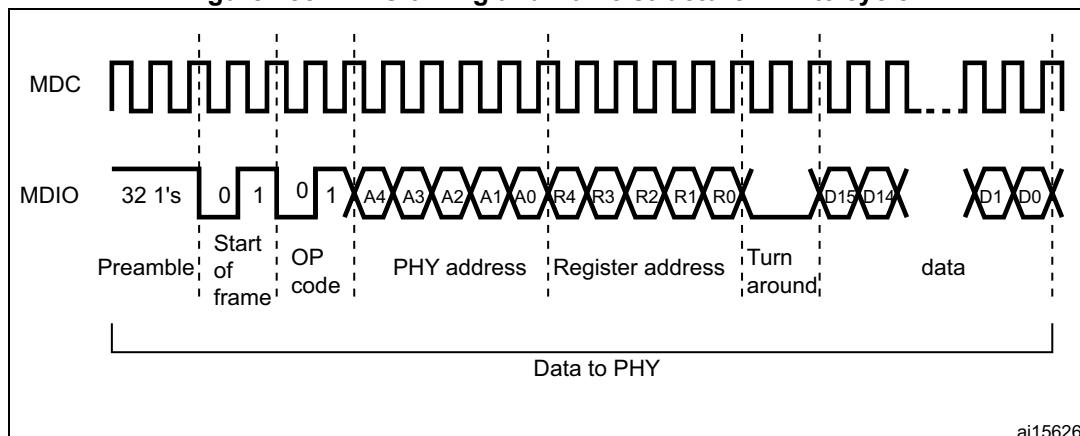
- **Data:** the data field is 16-bit. The first bit transmitted and received must be bit 15 of the ETH\_MIID register.
- **Idle:** the MDIO line is driven in high-impedance state. All three-state drivers must be disabled and the PHY's pull-up resistor keeps the line at logic one.

### SMI write operation

When the application sets the MII Write and Busy bits (in *Ethernet MAC MII address register (ETH\_MACMIIAR)*), the SMI initiates a write operation into the PHY registers by transferring the PHY address, the register address in PHY, and the write data (in *Ethernet MAC MII data register (ETH\_MACMIDR)*). The application should not change the MII Address register contents or the MII Data register while the transaction is ongoing. Write operations to the MII Address register or the MII Data Register during this period are ignored (the Busy bit is high), and the transaction is completed without any error. After the Write operation has completed, the SMI indicates this by resetting the Busy bit.

*Figure 488* shows the frame format for the write operation.

**Figure 488. MDIO timing and frame structure - Write cycle**

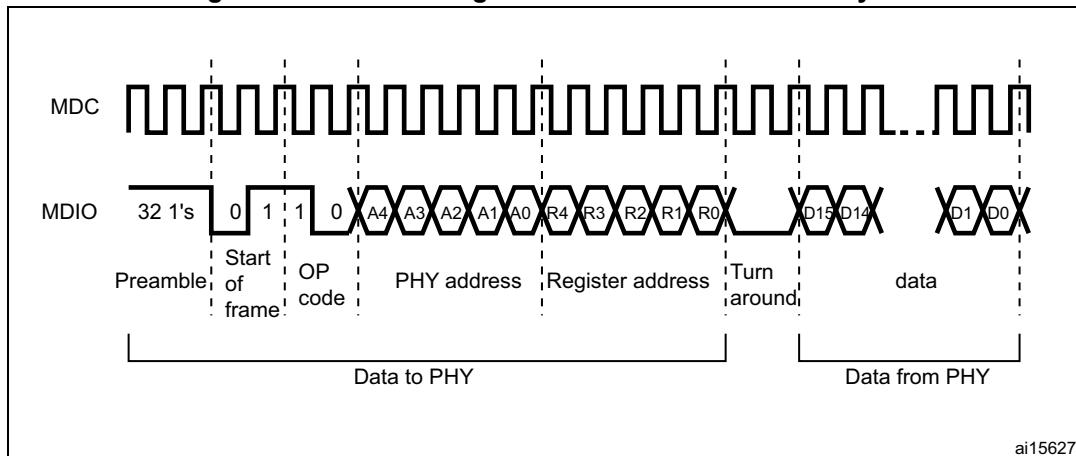


### SMI read operation

When the user sets the MII Busy bit in the Ethernet MAC MII address register (ETH\_MACMIIAR) with the MII Write bit at 0, the SMI initiates a read operation in the PHY registers by transferring the PHY address and the register address in PHY. The application should not change the MII Address register contents or the MII Data register while the transaction is ongoing. Write operations to the MII Address register or MII Data Register during this period are ignored (the Busy bit is high) and the transaction is completed without any error. After the read operation has completed, the SMI resets the Busy bit and then updates the MII Data register with the data read from the PHY.

*Figure 489* shows the frame format for the read operation.

Figure 489. MDIO timing and frame structure - Read cycle



### SMI clock selection

The MAC initiates the Management Write/Read operation. The SMI clock is a divided clock whose source is the application clock (AHB clock). The divide factor depends on the clock range setting in the MII Address register.

[Table 279](#) shows how to set the clock ranges.

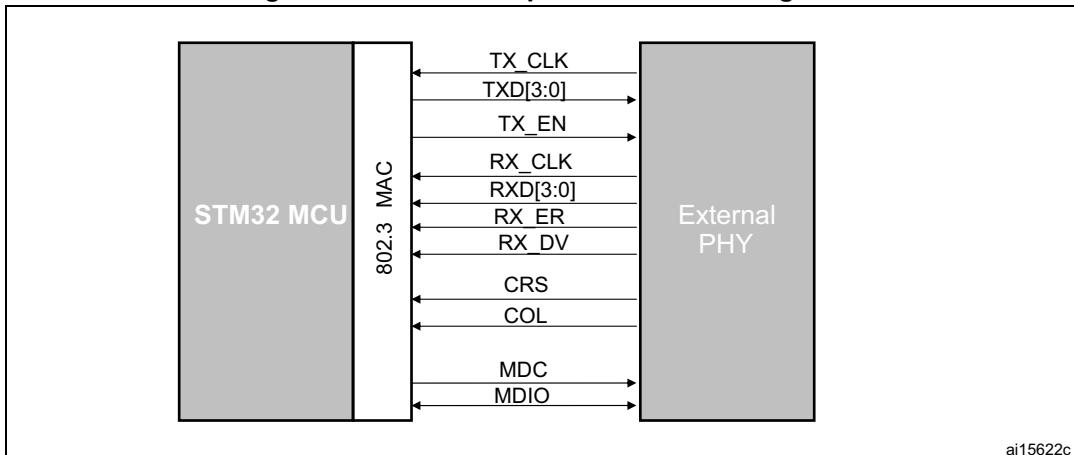
Table 279. Clock range

Selection	HCLK clock	MDC clock
000	60-100 MHz	AHB clock / 42
001	100-150 MHz	AHB clock / 62
010	20-35 MHz	AHB clock / 16
011	35-60 MHz	AHB clock / 26
100	150-216 MHz	AHB clock / 102
101, 110, 111	Reserved	-

### 38.4.2 Media-independent interface: MII

The media-independent interface (MII) defines the interconnection between the MAC sublayer and the PHY for data transfer at 10 Mbit/s and 100 Mbit/s.

**Figure 490. Media independent interface signals**



- MII\_TX\_CLK: continuous clock that provides the timing reference for the TX data transfer. The nominal frequency is: 2.5 MHz at 10 Mbit/s speed; 25 MHz at 100 Mbit/s speed.
- MII\_RX\_CLK: continuous clock that provides the timing reference for the RX data transfer. The nominal frequency is: 2.5 MHz at 10 Mbit/s speed; 25 MHz at 100 Mbit/s speed.
- MII\_TX\_EN: transmission enable indicates that the MAC is presenting nibbles on the MII for transmission. It must be asserted synchronously (MII\_TX\_CLK) with the first nibble of the preamble and must remain asserted while all nibbles to be transmitted are presented to the MII.
- MII\_TXD[3:0]: transmit data is a bundle of 4 data signals driven synchronously by the MAC sublayer and qualified (valid data) on the assertion of the MII\_TX\_EN signal. MII\_TXD[0] is the least significant bit, MII\_TXD[3] is the most significant bit. While MII\_TX\_EN is deasserted the transmit data must have no effect upon the PHY.
- MII\_CRS: carrier sense is asserted by the PHY when either the transmit or receive medium is non idle. It shall be deasserted by the PHY when both the transmit and receive media are idle. The PHY must ensure that the MII\_CS signal remains asserted throughout the duration of a collision condition. This signal is not required to transition synchronously with respect to the TX and RX clocks. In full duplex mode the state of this signal is don't care for the MAC sublayer.
- MII\_COL: collision detection must be asserted by the PHY upon detection of a collision on the medium and must remain asserted while the collision condition persists. This signal is not required to transition synchronously with respect to the TX and RX clocks. In full duplex mode the state of this signal is don't care for the MAC sublayer.
- MII\_RXD[3:0]: reception data is a bundle of 4 data signals driven synchronously by the PHY and qualified (valid data) on the assertion of the MII\_RX\_DV signal. MII\_RXD[0] is the least significant bit, MII\_RXD[3] is the most significant bit. While MII\_RX\_EN is

deasserted and MII\_RX\_ER is asserted, a specific MII\_RXD[3:0] value is used to transfer specific information from the PHY (see [Table 281](#)).

- MII\_RX\_DV: receive data valid indicates that the PHY is presenting recovered and decoded nibbles on the MII for reception. It must be asserted synchronously (MII\_RX\_CLK) with the first recovered nibble of the frame and must remain asserted through the final recovered nibble. It must be deasserted prior to the first clock cycle that follows the final nibble. In order to receive the frame correctly, the MII\_RX\_DV signal must encompass the frame, starting no later than the SFD field.
- MII\_RX\_ER: receive error must be asserted for one or more clock periods (MII\_RX\_CLK) to indicate to the MAC sublayer that an error was detected somewhere in the frame. This error condition must be qualified by MII\_RX\_DV assertion as described in [Table 281](#).

**Table 280. TX interface signal encoding**

MII_TX_EN	MII_TXD[3:0]	Description
0	0000 through 1111	Normal inter-frame
1	0000 through 1111	Normal data transmission

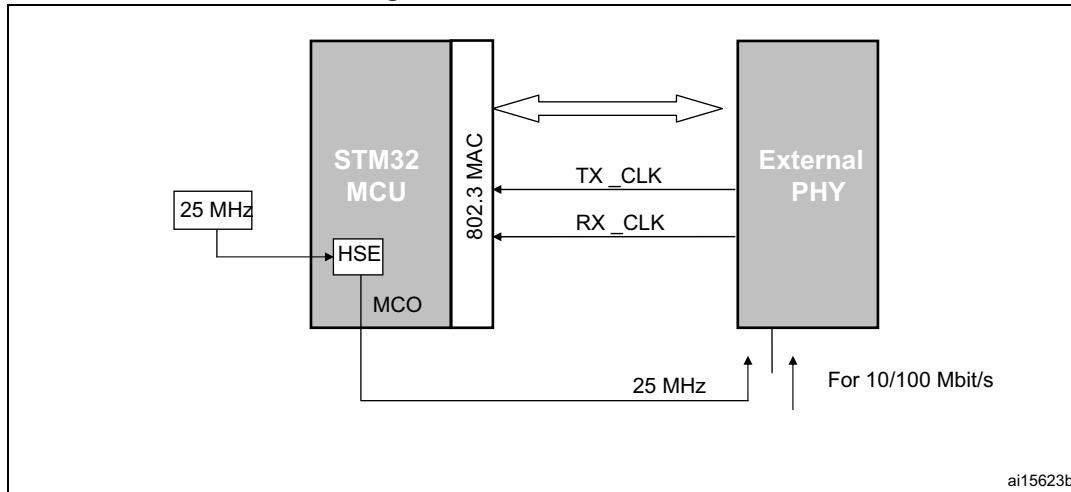
**Table 281. RX interface signal encoding**

MII_RX_DV	MII_RX_ERR	MII_RXD[3:0]	Description
0	0	0000 through 1111	Normal inter-frame
0	1	0000	Normal inter-frame
0	1	0001 through 1101	Reserved
0	1	1110	False carrier indication
0	1	1111	Reserved
1	0	0000 through 1111	Normal data reception
1	1	0000 through 1111	Data reception with errors

### MII clock sources

To generate both TX\_CLK and RX\_CLK clock signals, the external PHY must be clocked with an external 25 MHz as shown in [Figure 491](#). Instead of using an external 25 MHz quartz to provide this clock, the STM32F75xxx and STM32F74xxx microcontrollers can output this signal on its MCO pin. In this case, the PLL multiplier has to be configured so as to get the desired frequency on the MCO pin, from the 25 MHz external quartz.

Figure 491. MII clock sources



ai15623b

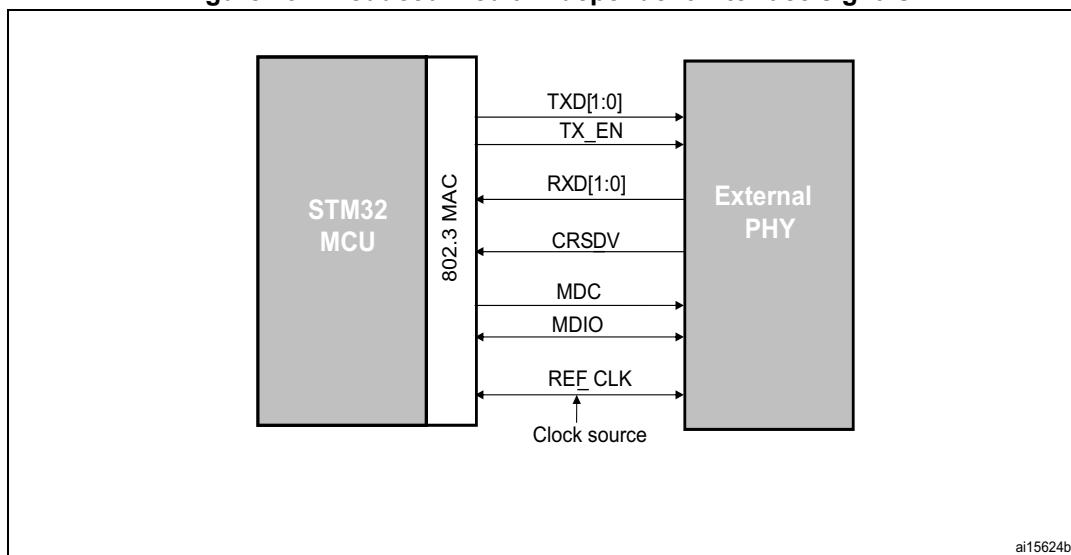
### 38.4.3 Reduced media-independent interface: RMII

The reduced media-independent interface (RMII) specification reduces the pin count between the microcontroller Ethernet peripheral and the external Ethernet in 10/100 Mbit/s. According to the IEEE 802.3u standard, an MII contains 16 pins for data and control. The RMII specification is dedicated to reduce the pin count to 7 pins (a 62.5% decrease in pin count).

The RMII is instantiated between the MAC and the PHY. This helps translation of the MAC's MII into the RMII. The RMII block has the following characteristics:

- It supports 10-Mbit/s and 100-Mbit/s operating rates
- The clock reference must be doubled to 50 MHz
- The same clock reference must be sourced externally to both MAC and external Ethernet PHY
- It provides independent 2-bit wide (dabit) transmit and receive data paths

Figure 492. Reduced media-independent interface signals

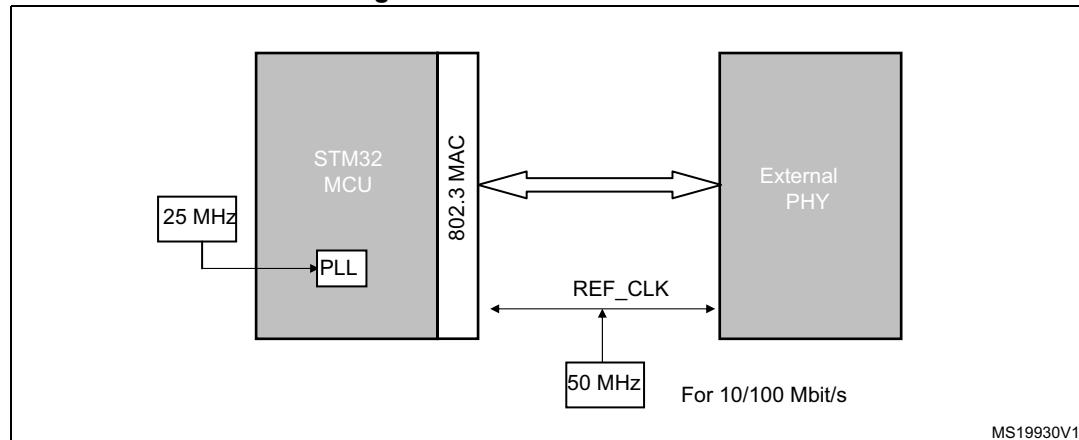


ai15624b

### RMII clock sources

Either clock the PHY from an external 50 MHz clock or use a PHY with an embedded PLL to generate the 50 MHz frequency.

**Figure 493. RMII clock sources-**



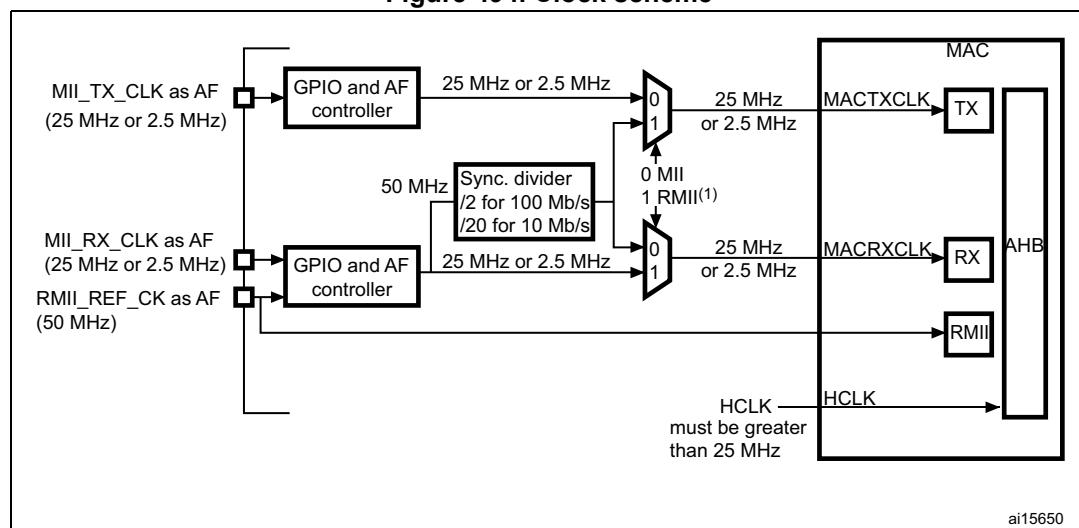
### 38.4.4 MII/RMII selection

The mode, MII or RMII, is selected using the configuration bit 23, MII\_RMII\_SEL, in the SYSCFG\_PMC register. The application has to set the MII/RMII mode while the Ethernet controller is under reset or before enabling the clocks.

#### MII/RMII internal clock scheme

The clock scheme required to support both the MII and RMII, as well as 10 and 100 Mbit/s operations is described in [Figure 494](#).

**Figure 494. Clock scheme**



1. The MII/RMII selection is controlled through bit 23, MII\_RMII\_SEL, in the SYSCFG\_PMC register.
- To save a pin, the two input clock signals, RMII\_REF\_CK and MII\_RX\_CLK, are multiplexed on the same GPIO pin.

## 38.5 Ethernet functional description: MAC 802.3

The IEEE 802.3 International Standard for local area networks (LANs) employs the CSMA/CD (carrier sense multiple access with collision detection) as the access method.

The Ethernet peripheral consists of a MAC 802.3 (media access control) controller with media independent interface (MII) and a dedicated DMA controller.

The MAC block implements the LAN CSMA/CD sublayer for the following families of systems: 10 Mbit/s and 100 Mbit/s of data rates for baseband and broadband systems. Half- and full-duplex operation modes are supported. The collision detection access method is applied only to the half-duplex operation mode. The MAC control frame sublayer is supported.

The MAC sublayer performs the following functions associated with a data link control procedure:

- Data encapsulation (transmit and receive)
  - Framing (frame boundary delimitation, frame synchronization)
  - Addressing (handling of source and destination addresses)
  - Error detection
- Media access management
  - Medium allocation (collision avoidance)
  - Contention resolution (collision handling)

Basically there are two operating modes of the MAC sublayer:

- Half-duplex mode: the stations contend for the use of the physical medium, using the CSMA/CD algorithms.
- Full duplex mode: simultaneous transmission and reception without contention resolution (CSMA/CD algorithm are unnecessary) when all the following conditions are met:
  - physical medium capability to support simultaneous transmission and reception
  - exactly 2 stations connected to the LAN
  - both stations configured for full-duplex operation

### 38.5.1 MAC 802.3 frame format

The MAC block implements the MAC sublayer and the optional MAC control sublayer (10/100 Mbit/s) as specified by the IEEE 802.3-2002 standard.

Two frame formats are specified for data communication systems using the CSMA/CD MAC:

- Basic MAC frame format
- Tagged MAC frame format (extension of the basic MAC frame format)

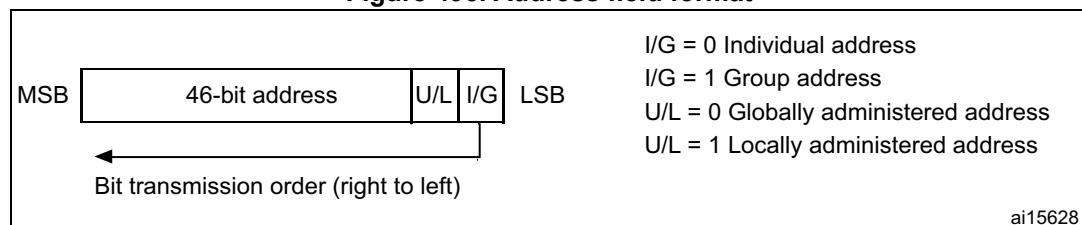
*Figure 496* and *Figure 497* describe the frame structure (untagged and tagged) that includes the following fields:

- Preamble: 7-byte field used for synchronization purposes (PLS circuitry)  
Hexadecimal value: 55-55-55-55-55-55-55  
Bit pattern: 01010101 01010101 01010101 01010101 01010101 01010101 01010101 (right-to-left bit transmission)
- Start frame delimiter (SFD): 1-byte field used to indicate the start of a frame.  
Hexadecimal value: D5  
Bit pattern: 11010101 (right-to-left bit transmission)
- Destination and Source Address fields: 6-byte fields to indicate the destination and source station addresses as follows (see *Figure 495*):
  - Each address is 48 bits in length
  - The first LSB bit (I/G) in the destination address field is used to indicate an individual (I/G = 0) or a group address (I/G = 1). A group address could identify none, one or more, or all the stations connected to the LAN. In the source address the first bit is reserved and reset to 0.
  - The second bit (U/L) distinguishes between locally (U/L = 1) or globally (U/L = 0) administered addresses. For broadcast addresses this bit is also 1.
  - Each byte of each address field must be transmitted least significant bit first.

The address designation is based on the following types:

- Individual address: this is the physical address associated with a particular station on the network.
- Group address. A multideestination address associated with one or more stations on a given network. There are two kinds of multicast address:
  - Multicast-group address: an address associated with a group of logically related stations.
  - Broadcast address: a distinguished, predefined multicast address (all 1's in the destination address field) that always denotes all the stations on a given LAN.

**Figure 495. Address field format**



- QTag Prefix: 4-byte field inserted between the Source address field and the MAC Client Length/Type field. This field is an extension of the basic frame (untagged) to obtain the tagged MAC frame. The untagged MAC frames do not include this field. The extensions for tagging are as follows:
  - 2-byte constant Length/Type field value consistent with the Type interpretation (greater than 0x0600) equal to the value of the 802.1Q Tag Protocol Type (0x8100)

hexadecimal). This constant field is used to distinguish tagged and untagged MAC frames.

- 2-byte field containing the Tag control information field subdivided as follows: a 3-bit user priority, a canonical format indicator (CFI) bit and a 12-bit VLAN Identifier. The length of the tagged MAC frame is extended by 4 bytes by the QTag Prefix.
- MAC client length/type: 2-byte field with different meaning (mutually exclusive), depending on its value:
  - If the value is less than or equal to maxValidFrame (0d1500) then this field indicates the number of MAC client data bytes contained in the subsequent data field of the 802.3 frame (length interpretation).
  - If the value is greater than or equal to MinTypeValue (0d1536 decimal, 0x0600) then this field indicates the nature of the MAC client protocol (Type interpretation) related to the Ethernet frame.

Regardless of the interpretation of the length/type field, if the length of the data field is less than the minimum required for proper operation of the protocol, a PAD field is added after the data field but prior to the FCS (frame check sequence) field. The length/type field is transmitted and received with the higher-order byte first.

For length/type field values in the range between maxValidLength and minTypeValue (boundaries excluded), the behavior of the MAC sublayer is not specified: they may or may not be passed by the MAC sublayer.

- Data and PAD fields: n-byte data field. Full data transparency is provided, it means that any arbitrary sequence of byte values may appear in the data field. The size of the PAD, if any, is determined by the size of the data field. Max and min length of the data and PAD field are:
  - Maximum length = 1500 bytes
  - Minimum length for untagged MAC frames = 46 bytes
  - Minimum length for tagged MAC frames = 42 bytes

When the data field length is less than the minimum required, the PAD field is added to match the minimum length (42 bytes for tagged frames, 46 bytes for untagged frames).

- Frame check sequence: 4-byte field that contains the cyclic redundancy check (CRC) value. The CRC computation is based on the following fields: source address, destination address, QTag prefix, length/type, LLC data and PAD (that is, all fields except the preamble, SFD). The generating polynomial is the following:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The CRC value of a frame is computed as follows:

- The first 2 bits of the frame are complemented
- The n-bits of the frame are the coefficients of a polynomial M(x) of degree (n – 1). The first bit of the destination address corresponds to the  $x^{n-1}$  term and the last bit of the data field corresponds to the  $x^0$  term
- M(x) is multiplied by  $x^{32}$  and divided by G(x), producing a remainder R(x) of degree  $\leq 31$
- The coefficients of R(x) are considered as a 32-bit sequence
- The bit sequence is complemented and the result is the CRC
- The 32-bits of the CRC value are placed in the frame check sequence. The  $x^{32}$  term is the first transmitted, the  $x^0$  term is the last one

Figure 496. MAC frame format

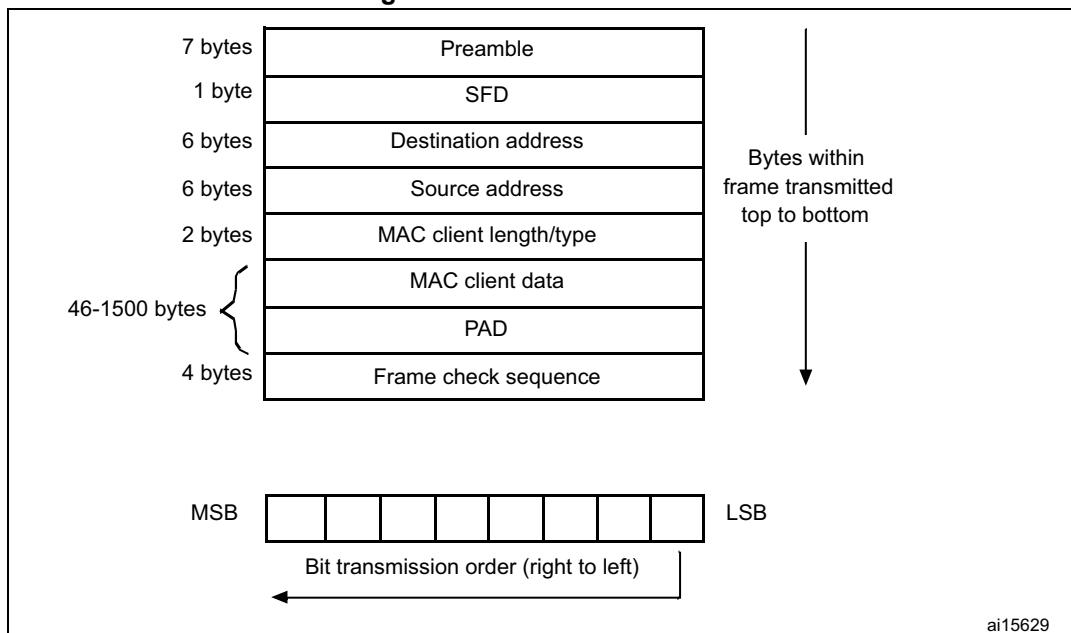
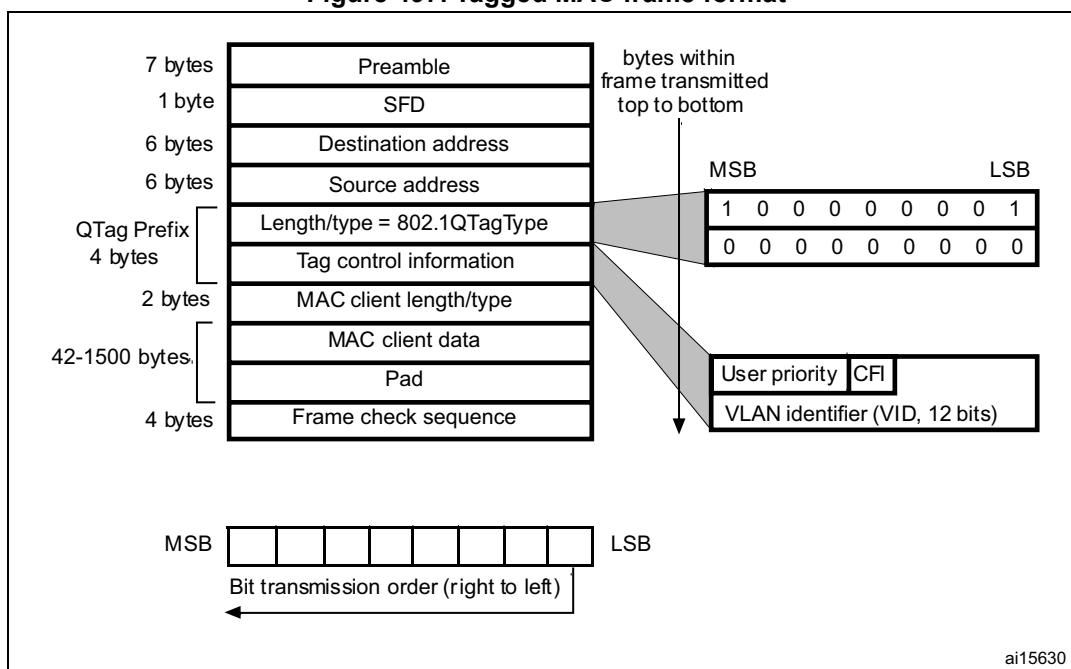


Figure 497. Tagged MAC frame format



Each byte of the MAC frame, except the FCS field, is transmitted low-order bit first.

An invalid MAC frame is defined by one of the following conditions:

- The frame length is inconsistent with the expected value as specified by the length/type field. If the length/type field contains a type value, then the frame length is assumed to be consistent with this field (no invalid frame)
- The frame length is not an integer number of bytes (extra bits)
- The CRC value computed on the incoming frame does not match the included FCS

### 38.5.2 MAC frame transmission

The DMA controls all transactions for the transmit path. Ethernet frames read from the system memory are pushed into the FIFO by the DMA. The frames are then popped out and transferred to the MAC core. When the end-of-frame is transferred, the status of the transmission is taken from the MAC core and transferred back to the DMA. The Transmit FIFO has a depth of 2 Kbyte. FIFO-fill level is indicated to the DMA so that it can initiate a data fetch in required bursts from the system memory, using the AHB interface. The data from the AHB Master interface is pushed into the FIFO.

When the SOF is detected, the MAC accepts the data and begins transmitting to the MII. The time required to transmit the frame data to the MII after the application initiates transmission is variable, depending on delay factors like IFG delay, time to transmit preamble/SFD, and any back-off delays for Half-duplex mode. After the EOF is transferred to the MAC core, the core completes normal transmission and then gives the status of transmission back to the DMA. If a normal collision (in Half-duplex mode) occurs during transmission, the MAC core makes the transmit status valid, then accepts and drops all further data until the next SOF is received. The same frame should be retransmitted from SOF on observing a Retry request (in the Status) from the MAC. The MAC issues an underflow status if the data are not provided continuously during the transmission. During the normal transfer of a frame, if the MAC receives an SOF without getting an EOF for the previous frame, then the SOF is ignored and the new frame is considered as the continuation of the previous frame.

There are two modes of operation for popping data towards the MAC core:

- In Threshold mode, as soon as the number of bytes in the FIFO crosses the configured threshold level (or when the end-of-frame is written before the threshold is crossed), the data is ready to be popped out and forwarded to the MAC core. The threshold level is configured using the TTC bits of ETH\_DMABMR.
- In Store-and-forward mode, only after a complete frame is stored in the FIFO, the frame is popped towards the MAC core. If the Tx FIFO size is smaller than the Ethernet frame to be transmitted, then the frame is popped towards the MAC core when the Tx FIFO becomes almost full.

The application can flush the Transmit FIFO of all contents by setting the FTF (ETH\_DMAOMR register [20]) bit. This bit is self-clearing and initializes the FIFO pointers to the default state. If the FTF bit is set during a frame transfer to the MAC core, then transfer is stopped as the FIFO is considered to be empty. Hence an underflow event occurs at the MAC transmitter and the corresponding Status word is forwarded to the DMA.

### Automatic CRC and pad generation

When the number of bytes received from the application falls below 60 (DA+SA+LT+Data), zeros are appended to the transmitting frame to make the data length exactly 46 bytes to meet the minimum data field requirement of IEEE 802.3. The MAC can be programmed not to append any padding. The cyclic redundancy check (CRC) for the frame check sequence (FCS) field is calculated and appended to the data being transmitted. When the MAC is programmed to not append the CRC value to the end of Ethernet frames, the computed CRC is not transmitted. An exception to this rule is that when the MAC is programmed to append pads for frames (DA+SA+LT+Data) less than 60 bytes, CRC will be appended at the end of the padded frames.

The CRC generator calculates the 32-bit CRC for the FCS field of the Ethernet frame. The encoding is defined by the following polynomial.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

### Transmit protocol

The MAC controls the operation of Ethernet frame transmission. It performs the following functions to meet the IEEE 802.3/802.3z specifications. It:

- generates the preamble and SFD
- generates the jam pattern in Half-duplex mode
- controls the Jabber timeout
- controls the flow for Half-duplex mode (back pressure)
- generates the transmit frame status
- contains time stamp snapshot logic in accordance with IEEE 1588

When a new frame transmission is requested, the MAC sends out the preamble and SFD, followed by the data. The preamble is defined as 7 bytes of 0b10101010 pattern, and the SFD is defined as 1 byte of 0b10101011 pattern. The collision window is defined as 1 slot time (512 bit times for 10/100 Mbit/s Ethernet). The jam pattern generation is applicable only to Half-duplex mode, not to Full-duplex mode.

In MII mode, if a collision occurs at any time from the beginning of the frame to the end of the CRC field, the MAC sends a 32-bit jam pattern of 0x5555 5555 on the MII to inform all other stations that a collision has occurred. If the collision is seen during the preamble transmission phase, the MAC completes the transmission of the preamble and SFD and then sends the jam pattern.

A jabber timer is maintained to cut off the transmission of Ethernet frames if more than 2048 (default) bytes have to be transferred. The MAC uses the deferral mechanism for flow control (back pressure) in Half-duplex mode. When the application requests to stop receiving frames, the MAC sends a JAM pattern of 32 bytes whenever it senses the reception of a frame, provided that transmit flow control is enabled. This results in a collision and the remote station backs off. The application requests flow control by setting the BPA bit (bit 0) in the ETH\_MACFCR register. If the application requests a frame to be transmitted, then it is scheduled and transmitted even when back pressure is activated. Note that if back pressure is kept activated for a long time (and more than 16 consecutive collision events occur) then the remote stations abort their transmissions due to excessive collisions. If IEEE 1588 time stamping is enabled for the transmit frame, this block takes a snapshot of the system time when the SFD is put onto the transmit MII bus.

### Transmit scheduler

The MAC is responsible for scheduling the frame transmission on the MII. It maintains the interframe gap between two transmitted frames and follows the truncated binary exponential backoff algorithm for Half-duplex mode. The MAC enables transmission after satisfying the IFG and backoff delays. It maintains an idle period of the configured interframe gap (IFG bits in the ETH\_MACCR register) between any two transmitted frames. If frames to be transmitted arrive sooner than the configured IFG time, the MII waits for the enable signal from the MAC before starting the transmission on it. The MAC starts its IFG counter as soon as the carrier signal of the MII goes inactive. At the end of the programmed IFG value, the MAC enables transmission in Full-duplex mode. In Half-duplex mode and when IFG is

configured for 96 bit times, the MAC follows the rule of deference specified in Section 4.2.3.2.1 of the IEEE 802.3 specification. The MAC resets its IFG counter if a carrier is detected during the first two-thirds (64-bit times for all IFG values) of the IFG interval. If the carrier is detected during the final one third of the IFG interval, the MAC continues the IFG count and enables the transmitter after the IFG interval. The MAC implements the truncated binary exponential backoff algorithm when it operates in Half-duplex mode.

### Transmit flow control

When the Transmit Flow Control Enable bit (TFE bit in ETH\_MACFCR) is set, the MAC generates Pause frames and transmits them as necessary, in Full-duplex mode. The Pause frame is appended with the calculated CRC, and is sent. Pause frame generation can be initiated in two ways.

A pause frame is sent either when the application sets the FCB bit in the ETH\_MACFCR register or when the receive FIFO is full (packet buffer).

- If the application has requested flow control by setting the FCB bit in ETH\_MACFCR, the MAC generates and transmits a single Pause frame. The value of the pause time in the generated frame contains the programmed pause time value in ETH\_MACFCR. To extend the pause or end the pause prior to the time specified in the previously transmitted Pause frame, the application must request another Pause frame transmission after programming the Pause Time value (PT in ETH\_MACFCR register) with the appropriate value.
- If the application has requested flow control when the receive FIFO is full, the MAC generates and transmits a Pause frame. The value of the pause time in the generated frame is the programmed pause time value in ETH\_MACFCR. If the receive FIFO remains full at a configurable number of slot-times (PLT bits in ETH\_MACFCR) before this Pause time runs out, a second Pause frame is transmitted. The process is repeated as long as the receive FIFO remains full. If this condition is no more satisfied prior to the sampling time, the MAC transmits a Pause frame with zero pause time to indicate to the remote end that the receive buffer is ready to receive new data frames.

### Single-packet transmit operation

The general sequence of events for a transmit operation is as follows:

1. If the system has data to be transferred, the DMA controller fetches them from the memory through the AHB Master interface and starts forwarding them to the FIFO. It continues to receive the data until the end of frame is transferred.
2. When the threshold level is crossed or a full packet of data is received into the FIFO, the frame data are popped and driven to the MAC core. The DMA continues to transfer data from the FIFO until a complete packet has been transferred to the MAC. Upon completion of the frame, the DMA controller is notified by the status coming from the MAC.

### Transmit operation—Two packets in the buffer

1. Because the DMA must update the descriptor status before releasing it to the Host, there can be at the most two frames inside a transmit FIFO. The second frame is fetched by the DMA and put into the FIFO only if the OSF (operate on second frame) bit is set. If this bit is not set, the next frame is fetched from the memory only after the MAC has completely processed the frame and the DMA has released the descriptors.
2. If the OSF bit is set, the DMA starts fetching the second frame immediately after completing the transfer of the first frame to the FIFO. It does not wait for the status to be updated. In the meantime, the second frame is received into the FIFO while the first

frame is being transmitted. As soon as the first frame has been transferred and the status is received from the MAC, it is pushed to the DMA. If the DMA has already completed sending the second packet to the FIFO, the second transmission must wait for the status of the first packet before proceeding to the next frame.

### Retransmission during collision

While a frame is being transferred to the MAC, a collision event may occur on the MAC line interface in Half-duplex mode. The MAC would then indicate a retry attempt by giving the status even before the end of frame is received. Then the retransmission is enabled and the frame is popped out again from the FIFO. After more than 96 bytes have been popped towards the MAC core, the FIFO controller frees up that space and makes it available to the DMA to push in more data. This means that the retransmission is not possible after this threshold is crossed or when the MAC core indicates a late collision event.

### Transmit FIFO flush operation

The MAC provides a control to the software to flush the Transmit FIFO through the use of Bit 20 in the Operation mode register. The Flush operation is immediate and the Tx FIFO and the corresponding pointers are cleared to the initial state even if the Tx FIFO is in the middle of transferring a frame to the MAC Core. This results in an underflow event in the MAC transmitter, and the frame transmission is aborted. The status of such a frame is marked with both underflow and frame flush events (TDES0 bits 13 and 1). No data are coming to the FIFO from the application (DMA) during the Flush operation. Transfer transmit status words are transferred to the application for the number of frames that is flushed (including partial frames). Frames that are completely flushed have the Frame flush status bit (TDES0 13) set. The Flush operation is completed when the application (DMA) has accepted all of the Status words for the frames that were flushed. The Transmit FIFO Flush control register bit is then cleared. At this point, new frames from the application (DMA) are accepted. All data presented for transmission after a Flush operation are discarded unless they start with an SOF marker.

### Transmit status word

At the end of the Ethernet frame transfer to the MAC core and after the core has completed the transmission of the frame, the transmit status is given to the application. The detailed description of the Transmit Status is the same as for bits [23:0] in TDES0. If IEEE 1588 time stamping is enabled, a specific frames' 64-bit time stamp is returned, along with the transmit status.

### Transmit checksum offload

Communication protocols such as TCP and UDP implement checksum fields, which helps determine the integrity of data transmitted over a network. Because the most widespread use of Ethernet is to encapsulate TCP and UDP over IP datagrams, the Ethernet controller has a transmit checksum offload feature that supports checksum calculation and insertion in the transmit path, and error detection in the receive path. This section explains the operation of the checksum offload feature for transmitted frames.

**Note:** *The checksum for TCP, UDP or ICMP is calculated over a complete frame, then inserted into its corresponding header field. Due to this requirement, this function is enabled only when the Transmit FIFO is configured for Store-and-forward mode (that is, when the TSF bit*

*(is set in the ETH\_ETH\_DMAOMR register). If the core is configured for Threshold (cut-through) mode, the Transmit checksum offload is bypassed.*

*The user must make sure the Transmit FIFO is deep enough to store a complete frame before that frame is transferred to the MAC Core transmitter. If the FIFO depth is less than the input Ethernet frame size, the payload (TCP/UDP/ICMP) checksum insertion function is bypassed and only the frame's IPv4 Header checksum is modified, even in Store-and-forward mode.*

The transmit checksum offload supports two types of checksum calculation and insertion. This checksum can be controlled for each frame by setting the CIC bits (bits 23:22 in TDES0, described in [TDES0: Transmit descriptor Word0 on page 1567](#)).

See IETF specifications RFC 791, RFC 793, RFC 768, RFC 792, RFC 2460 and RFC 4443 for IPv4, TCP, UDP, ICMP, IPv6 and ICMPv6 packet header specifications, respectively.

- IP header checksum

In IPv4 datagrams, the integrity of the header fields is indicated by the 16-bit header checksum field (the eleventh and twelfth bytes of the IPv4 datagram). The checksum offload detects an IPv4 datagram when the Ethernet frame's Type field has the value 0x0800 and the IP datagram's Version field has the value 0x4. The input frame's checksum field is ignored during calculation and replaced by the calculated value. IPv6 headers do not have a checksum field; thus, the checksum offload does not modify IPv6 header fields. The result of this IP header checksum calculation is indicated by the IP Header Error status bit in the Transmit status (Bit 16). This status bit is set whenever the values of the Ethernet Type field and the IP header's Version field are not consistent, or when the Ethernet frame does not have enough data, as indicated by the IP header Length field. In other words, this bit is set when an IP header error is asserted under the following circumstances:

- a) For IPv4 datagrams:

- The received Ethernet type is 0x0800, but the IP header's Version field does not equal 0x4
- The IPv4 Header Length field indicates a value less than 0x5 (20 bytes)
- The total frame length is less than the value given in the IPv4 Header Length field

- b) For IPv6 datagrams:

- The Ethernet type is 0x86DD but the IP header Version field does not equal 0x6
- The frame ends before the IPv6 header (40 bytes) or extension header (as given in the corresponding Header Length field in an extension header) has been completely received. Even when the checksum offload detects such an IP header error, it inserts an IPv4 header checksum if the Ethernet Type field indicates an IPv4 payload.

- TCP/UDP/ICMP checksum

The TCP/UDP/ICMP checksum processes the IPv4 or IPv6 header (including extension headers) and determines whether the encapsulated payload is TCP, UDP or ICMP.

Note that:

- a) For non-TCP, -UDP, or -ICMP/ICMPv6 payloads, this checksum is bypassed and nothing further is modified in the frame.
- b) Fragmented IP frames (IPv4 or IPv6), IP frames with security features (such as an authentication header or encapsulated security payload), and IPv6 frames with routing headers are bypassed and not processed by the checksum.

The checksum is calculated for the TCP, UDP, or ICMP payload and inserted into its corresponding field in the header. It can work in the following two modes:

- In the first mode, the TCP, UDP, or ICMPv6 pseudo-header is not included in the checksum calculation and is assumed to be present in the input frame's checksum field. The checksum field is included in the checksum calculation, and then replaced by the final calculated checksum.
- In the second mode, the checksum field is ignored, the TCP, UDP, or ICMPv6 pseudo-header data are included into the checksum calculation, and the checksum field is overwritten with the final calculated value.

Note that: for ICMP-over-IPv4 packets, the checksum field in the ICMP packet must always be 0x0000 in both modes, because pseudo-headers are not defined for such packets. If it does not equal 0x0000, an incorrect checksum may be inserted into the packet.

The result of this operation is indicated by the payload checksum error status bit in the Transmit Status vector (bit 12). The payload checksum error status bit is set when either of the following is detected:

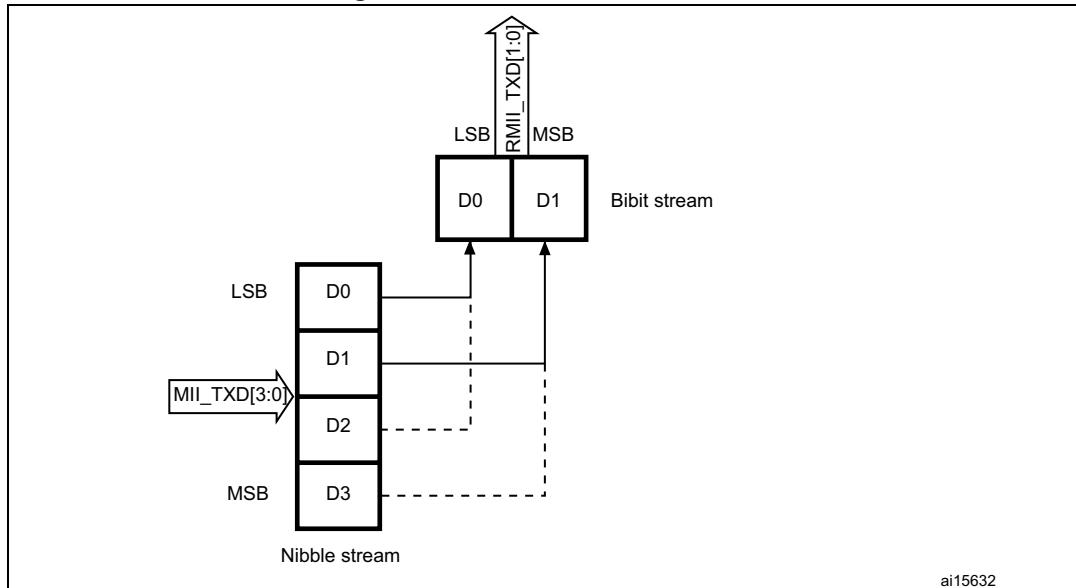
- the frame has been forwarded to the MAC transmitter in Store-and-forward mode without the end of frame being written to the FIFO
- the packet ends before the number of bytes indicated by the payload length field in the IP header is received.

When the packet is longer than the indicated payload length, the bytes are ignored as stuff bytes, and no error is reported. When the first type of error is detected, the TCP, UDP or ICMP header is not modified. For the second error type, still, the calculated checksum is inserted into the corresponding header field.

### MII/RMII transmit bit order

Each nibble from the MII is transmitted on the RMII a dabit at a time with the order of dabit transmission shown in [Figure 498](#). Lower order bits (D1 and D0) are transmitted first followed by higher order bits (D2 and D3).

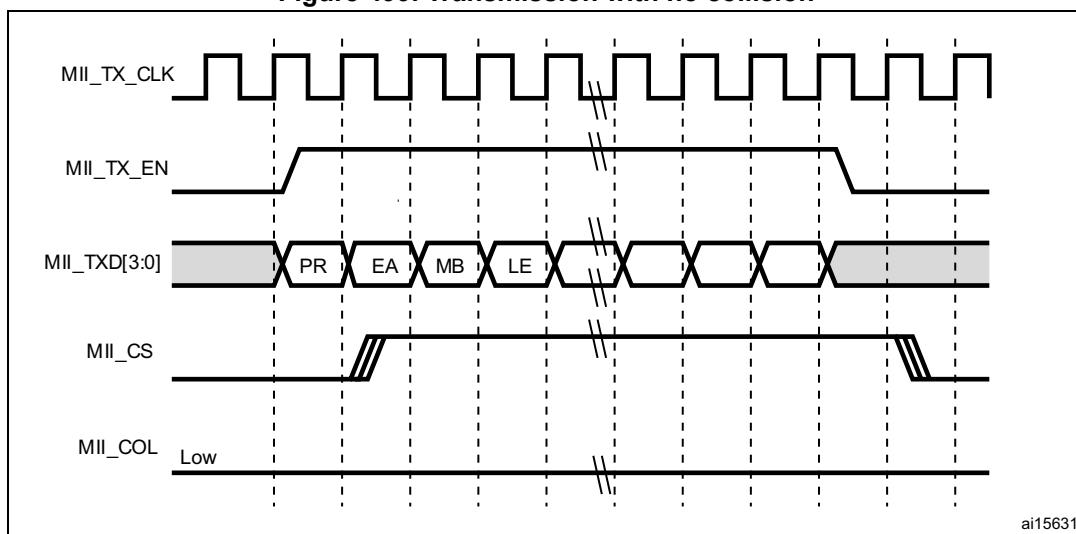
Figure 498. Transmission bit order



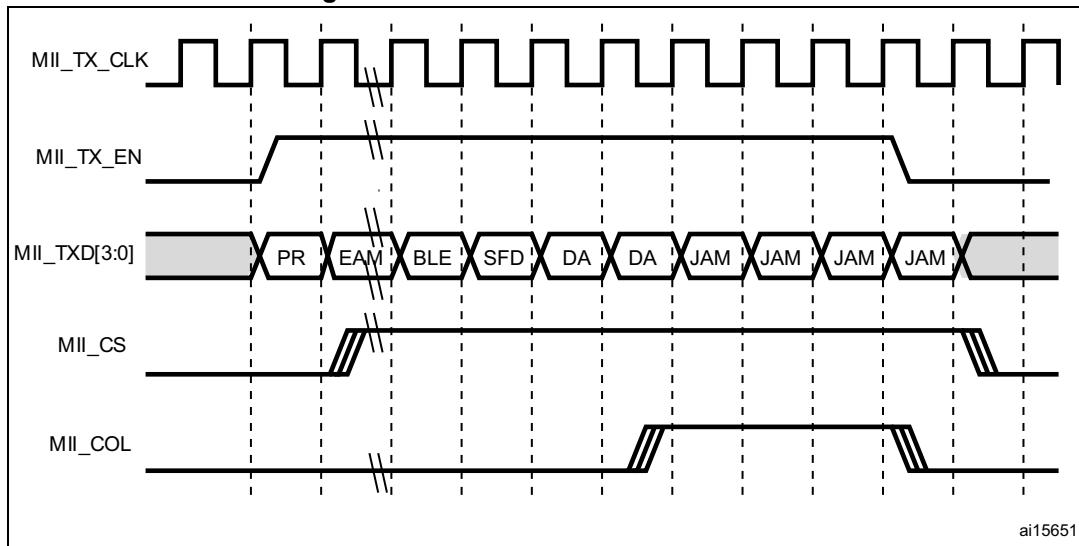
ai15632

**MII/RMII transmit timing diagrams**

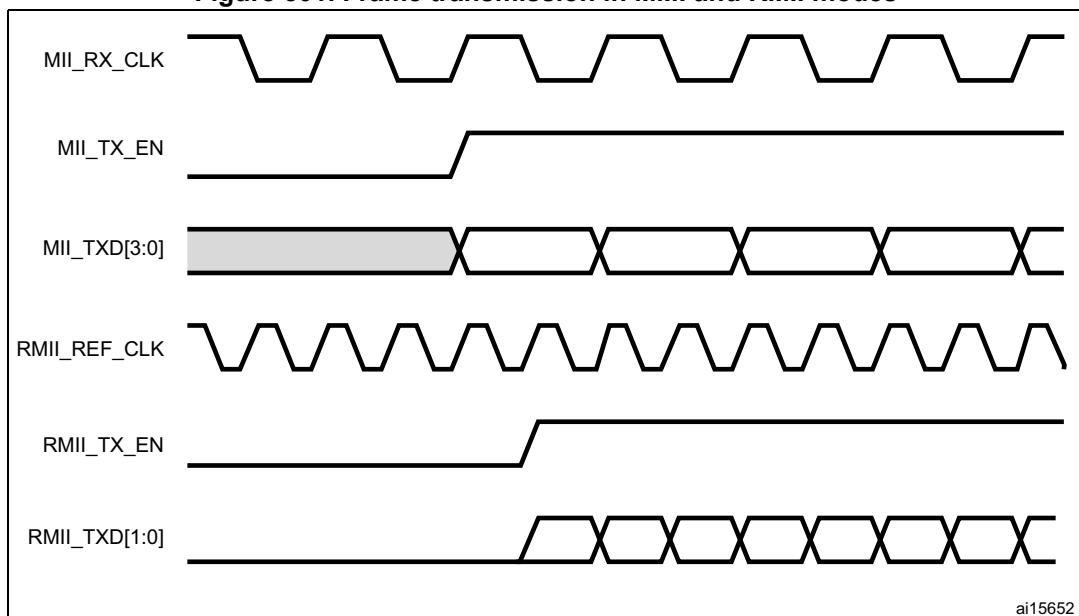
Figure 499. Transmission with no collision



ai15631

**Figure 500. Transmission with collision**

[Figure 501](#) shows a frame transmission in MII and RMII.

**Figure 501. Frame transmission in MII and RMII modes**

### 38.5.3 MAC frame reception

The MAC received frames are pushed into the Rx FIFO. The status (fill level) of this FIFO is indicated to the DMA once it crosses the configured receive threshold (RTC in the ETH\_DMAOMR register) so that the DMA can initiate pre-configured burst transfers towards the AHB interface.

In the default Cut-through mode, when 64 bytes (configured with the RTC bits in the ETH\_DMAOMR register) or a full packet of data are received into the FIFO, the data are popped out and the DMA is notified of its availability. Once the DMA has initiated the transfer to the AHB interface, the data transfer continues from the FIFO until a complete

packet has been transferred. Upon completion of the EOF frame transfer, the status word is popped out and sent to the DMA controller.

In Rx FIFO Store-and-forward mode (configured by the RSF bit in the ETH\_DMAOMR register), a frame is read out only after being written completely into the Receive FIFO. In this mode, all error frames are dropped (if the core is configured to do so) such that only valid frames are read out and forwarded to the application. In Cut-through mode, some error frames are not dropped, because the error status is received at the end of the frame, by which time the start of that frame has already been read out of the FIFO.

A receive operation is initiated when the MAC detects an SFD on the MII. The core strips the preamble and SFD before proceeding to process the frame. The header fields are checked for the filtering and the FCS field used to verify the CRC for the frame. The frame is dropped in the core if it fails the address filter.

### **Receive protocol**

The received frame preamble and SFD are stripped. Once the SFD has been detected, the MAC starts sending the Ethernet frame data to the receive FIFO, beginning with the first byte following the SFD (destination address). If IEEE 1588 time stamping is enabled, a snapshot of the system time is taken when any frame's SFD is detected on the MII. Unless the MAC filters out and drops the frame, this time stamp is passed on to the application.

If the received frame length/type field is less than 0x600 and if the MAC is programmed for the auto CRC/pad stripping option, the MAC sends the data of the frame to Rx FIFO up to the count specified in the length/type field, then starts dropping bytes (including the FCS field). If the Length/Type field is greater than or equal to 0x600, the MAC sends all received Ethernet frame data to Rx FIFO, regardless of the value on the programmed auto-CRC strip option. The MAC watchdog timer is enabled by default, that is, frames above 2048 bytes (DA + SA + LT + Data + pad + FCS) are cut off. This feature can be disabled by programming the watchdog disable (WD) bit in the MAC configuration register. However, even if the watchdog timer is disabled, frames greater than 16 KB in size are cut off and a watchdog timeout status is given.

### **Receive CRC: automatic CRC and pad stripping**

The MAC checks for any CRC error in the receiving frame. It calculates the 32-bit CRC for the received frame that includes the Destination address field through the FCS field. The encoding is defined by the following polynomial.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Regardless of the auto-pad/CRC strip, the MAC receives the entire frame to compute the CRC check for the received frame.

### **Receive checksum offload**

Both IPv4 and IPv6 frames in the received Ethernet frames are detected and processed for data integrity. The user can enable the receive checksum offload by setting the IPCO bit in the ETH\_MACCR register. The MAC receiver identifies IPv4 or IPv6 frames by checking for value 0x0800 or 0x86DD, respectively, in the received Ethernet frame Type field. This identification applies to VLAN-tagged frames as well. The receive checksum offload calculates IPv4 header checksums and checks that they match the received IPv4 header checksums. The IP Header Error bit is set for any mismatch between the indicated payload

type (Ethernet Type field) and the IP header version, or when the received frame does not have enough bytes, as indicated by the IPv4 header's Length field (or when fewer than 20 bytes are available in an IPv4 or IPv6 header). The receive checksum offload also identifies a TCP, UDP or ICMP payload in the received IP datagrams (IPv4 or IPv6) and calculates the checksum of such payloads properly, as defined in the TCP, UDP or ICMP specifications. It includes the TCP/UDP/ICMPv6 pseudo-header bytes for checksum calculation and checks whether the received checksum field matches the calculated value. The result of this operation is given as a Payload Checksum Error bit in the receive status word. This status bit is also set if the length of the TCP, UDP or ICMP payload does not match the expected payload length given in the IP header. As mentioned in [TCP/UDP/ICMP checksum on page 1537](#), the receive checksum offload bypasses the payload of fragmented IP datagrams, IP datagrams with security features, IPv6 routing headers, and payloads other than TCP, UDP or ICMP. This information (whether the checksum is bypassed or not) is given in the receive status, as described in the [RDES0: Receive descriptor Word0](#) section. In this configuration, the core does not append any payload checksum bytes to the received Ethernet frames.

As mentioned in [RDES0: Receive descriptor Word0 on page 1576](#), the meaning of certain register bits changes as shown in [Table 282](#).

**Table 282. Frame statuses**

Bit 18: Ethernet frame	Bit 27: Header checksum error	Bit 28: Payload checksum error	Frame status
0	0	0	The frame is an IEEE 802.3 frame (Length field value is less than 0x0600).
1	0	0	IPv4/IPv6 Type frame in which no checksum error is detected.
1	0	1	IPv4/IPv6 Type frame in which a payload checksum error (as described for PCE) is detected
1	1	0	IPv4/IPv6 Type frame in which IP header checksum error (as described for IPCO HCE) is detected.
1	1	1	IPv4/IPv6 Type frame in which both PCE and IPCO HCE are detected.
0	0	1	IPv4/IPv6 Type frame in which there is no IP HCE and the payload check is bypassed due to unsupported payload.
0	1	1	Type frame which is neither IPv4 or IPv6 (checksum offload bypasses the checksum check completely)
0	1	0	Reserved

### Receive frame controller

If the RA bit is reset in the MAC CSR frame filter register, the MAC performs frame filtering based on the destination/source address (the application still needs to perform another level of filtering if it decides not to receive any bad frames like runt, CRC error frames, etc.). On detecting a filter-fail, the frame is dropped and not transferred to the application. When the filtering parameters are changed dynamically, and in case of (DA-SA) filter-fail, the rest of

the frame is dropped and the Rx Status Word is immediately updated (with zero frame length, CRC error and Runt Error bits set), indicating the filter fail. In Ethernet power down mode, all received frames are dropped, and are not forwarded to the application.

### Receive flow control

The MAC detects the receiving Pause frame and pauses the frame transmission for the delay specified within the received Pause frame (only in Full-duplex mode). The Pause frame detection function can be enabled or disabled with the RFCE bit in ETH\_MACFCR. Once receive flow control has been enabled, the received frame destination address begins to be monitored for any match with the multicast address of the control frame (0x0180 C200 0001). If a match is detected (the destination address of the received frame matches the reserved control frame destination address), the MAC then decides whether or not to transfer the received control frame to the application, based on the level of the PCF bit in ETH\_MACFFR.

The MAC also decodes the type, opcode, and Pause Timer fields of the receiving control frame. If the byte count of the status indicates 64 bytes, and if there is no CRC error, the MAC transmitter pauses the transmission of any data frame for the duration of the decoded Pause time value, multiplied by the slot time (64 byte times for both 10/100 Mbit/s modes). Meanwhile, if another Pause frame is detected with a zero Pause time value, the MAC resets the Pause time and manages this new pause request.

If the received control frame matches neither the type field (0x8808), the opcode (0x00001), nor the byte length (64 bytes), or if there is a CRC error, the MAC does not generate a Pause.

In the case of a pause frame with a multicast destination address, the MAC filters the frame based on the address match.

For a pause frame with a unicast destination address, the MAC filtering depends on whether the DA matched the contents of the MAC address 0 register and whether the UPDF bit in ETH\_MACFCR is set (detecting a pause frame even with a unicast destination address). The PCF register bits (bits [7:6] in ETH\_MACFFR) control filtering for control frames in addition to address filtering.

### Receive operation multiframe handling

Since the status is available immediately following the data, the FIFO is capable of storing any number of frames into it, as long as it is not full.

### Error handling

If the Rx FIFO is full before it receives the EOF data from the MAC, an overflow is declared and the whole frame is dropped, and the overflow counter in the (ETH\_DMAMFBOCR register) is incremented. The status indicates a partial frame due to overflow. The Rx FIFO can filter error and undersized frames, if enabled (using the FEF and FUGF bits in ETH\_DMAOMR).

If the Receive FIFO is configured to operate in Store-and-forward mode, all error frames can be filtered and dropped.

In Cut-through mode, if a frame's status and length are available when that frame's SOF is read from the Rx FIFO, then the complete erroneous frame can be dropped. The DMA can flush the error frame being read from the FIFO, by enabling the receive frame flash bit. The data transfer to the application (DMA) is then stopped and the rest of the frame is internally read and dropped. The next frame transfer can then be started, if available.

### Receive status word

At the end of the Ethernet frame reception, the MAC outputs the receive status to the application (DMA). The detailed description of the receive status is the same as for bits[31:0] in RDES0, given in [RDES0: Receive descriptor Word0](#).

### Frame length interface

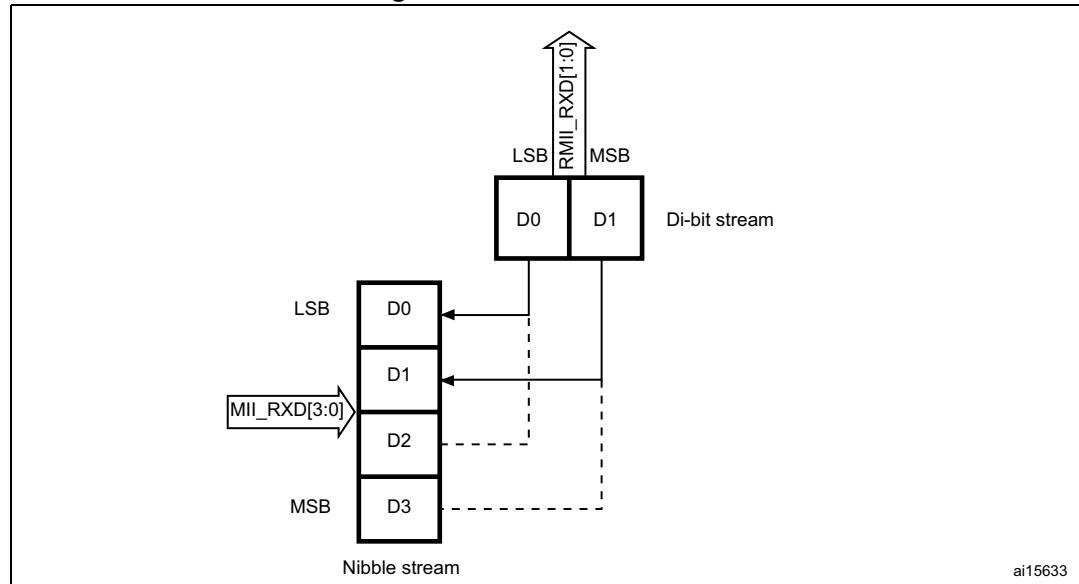
In case of switch applications, data transmission and reception between the application and MAC happen as complete frame transfers. The application layer should be aware of the length of the frames received from the ingress port in order to transfer the frame to the egress port. The MAC core provides the frame length of each received frame inside the status at the end of each frame reception.

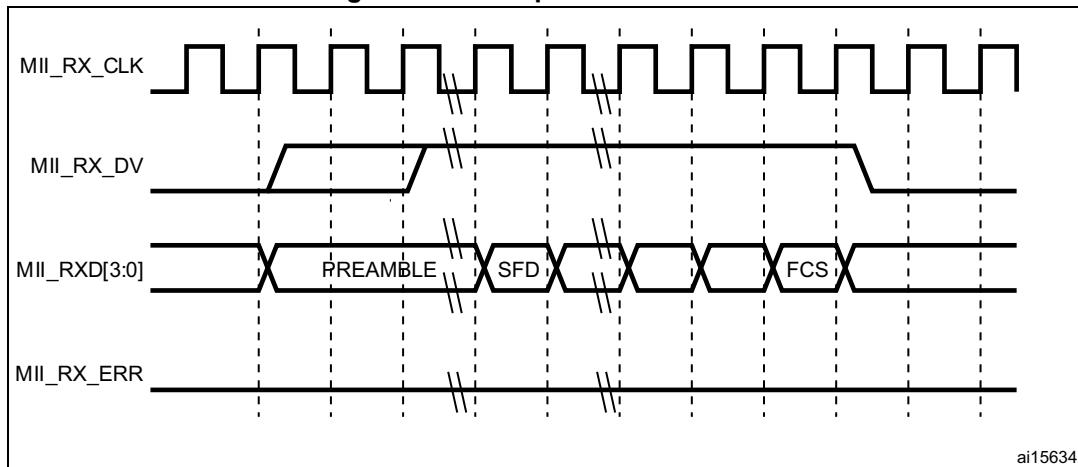
*Note:* A frame length value of 0 is given for partial frames written into the Rx FIFO due to overflow.

### MII/RMII receive bit order

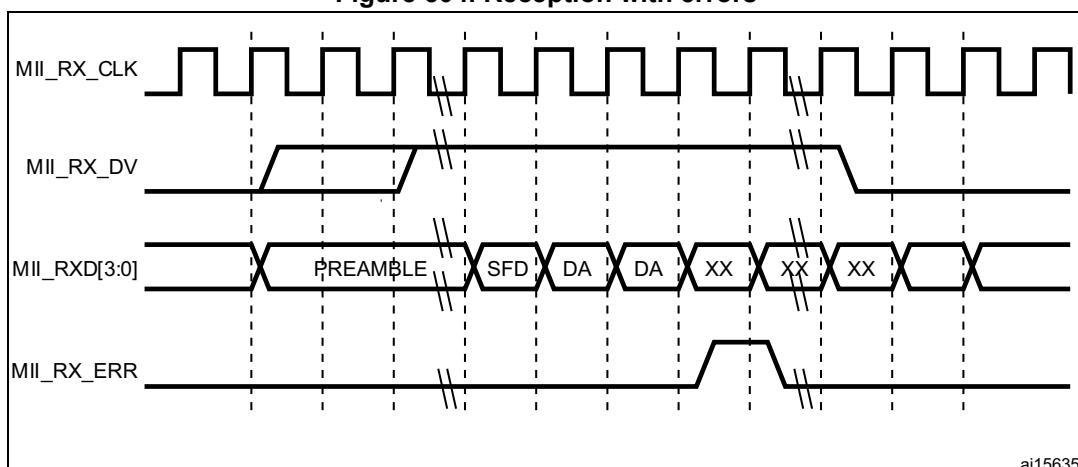
Each nibble is transmitted to the MII from the dabit received from the RMII in the nibble transmission order shown in [Figure 502](#). The lower-order bits (D0 and D1) are received first, followed by the higher-order bits (D2 and D3).

**Figure 502. Receive bit order**

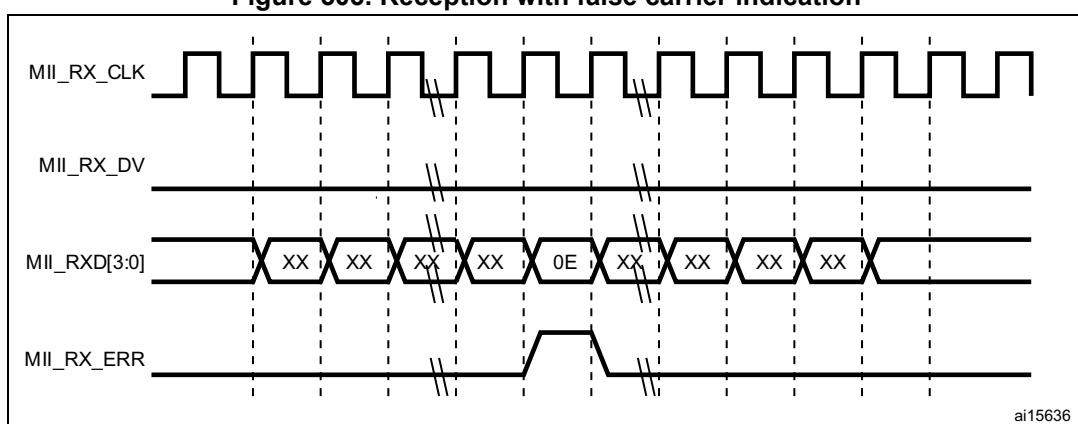


**Figure 503. Reception with no error**

ai15634

**Figure 504. Reception with errors**

ai15635

**Figure 505. Reception with false carrier indication**

ai15636

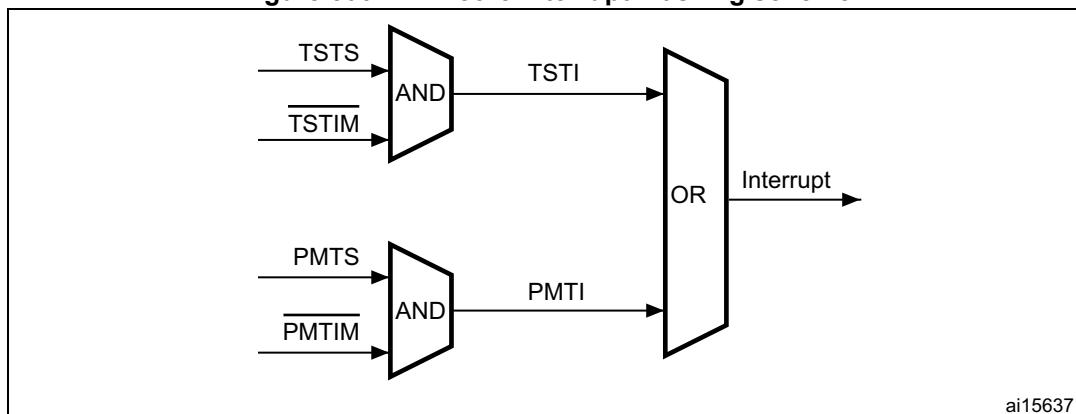
### 38.5.4 MAC interrupts

Interrupts can be generated from the MAC core as a result of various events.

The ETH\_MACSR register describes the events that can cause an interrupt from the MAC core. The user can prevent each event from asserting the interrupt by setting the corresponding mask bits in the Interrupt Mask register.

The interrupt register bits only indicate the block from which the event is reported. The user has to read the corresponding status registers and other registers to clear the interrupt. For example, bit 3 of the Interrupt register, set high, indicates that the Magic packet or Wake-on-LAN frame is received in Power-down mode. The user must read the ETH\_MACPMTCSR Register to clear this interrupt event.

**Figure 506. MAC core interrupt masking scheme**



### 38.5.5 MAC filtering

#### Address filtering

Address filtering checks the destination and source addresses on all received frames and the address filtering status is reported accordingly. Address checking is based on different parameters (Frame filter register) chosen by the application. The filtered frame can also be identified: multicast or broadcast frame.

Address filtering uses the station's physical (MAC) address and the Multicast Hash table for address checking purposes.

#### Unicast destination address filter

The MAC supports up to 4 MAC addresses for unicast perfect filtering. If perfect filtering is selected (HU bit in the Frame filter register is reset), the MAC compares all 48 bits of the received unicast address with the programmed MAC address for any match. Default MacAddr0 is always enabled, other addresses MacAddr1–MacAddr3 are selected with an individual enable bit. Each byte of these other addresses (MacAddr1–MacAddr3) can be masked during comparison with the corresponding received DA byte by setting the corresponding Mask Byte Control bit in the register. This helps group address filtering for the DA. In Hash filtering mode (when HU bit is set), the MAC performs imperfect filtering for unicast addresses using a 64-bit Hash table. For hash filtering, the MAC uses the 6 upper CRC (see note 1 below) bits of the received destination address to index the content of the Hash table. A value of 000000 selects bit 0 in the selected register, and a value of 111111 selects bit 63 in the Hash Table register. If the corresponding bit (indicated by the 6-bit CRC)

is set to 1, the unicast frame is said to have passed the Hash filter; otherwise, the frame has failed the Hash filter.

**Note:** *This CRC is a 32-bit value coded by the following polynomial (for more details refer to Section 38.5.3: MAC frame reception):*

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

### Multicast destination address filter

The MAC can be programmed to pass all multicast frames by setting the PAM bit in the Frame filter register. If the PAM bit is reset, the MAC performs the filtering for multicast addresses based on the HM bit in the Frame filter register. In Perfect filtering mode, the multicast address is compared with the programmed MAC destination address registers (1–3). Group address filtering is also supported. In Hash filtering mode, the MAC performs imperfect filtering using a 64-bit Hash table. For hash filtering, the MAC uses the 6 upper CRC (see note 1 below) bits of the received multicast address to index the content of the Hash table. A value of 000000 selects bit 0 in the selected register and a value of 111111 selects bit 63 in the Hash Table register. If the corresponding bit is set to 1, then the multicast frame is said to have passed the Hash filter; otherwise, the frame has failed the Hash filter.

**Note:** *This CRC is a 32-bit value coded by the following polynomial (for more details refer to Section 38.5.3: MAC frame reception):*

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

### Hash or perfect address filter

The DA filter can be configured to pass a frame when its DA matches either the Hash filter or the Perfect filter by setting the HPF bit in the Frame filter register and setting the corresponding HU or HM bits. This configuration applies to both unicast and multicast frames. If the HPF bit is reset, only one of the filters (Hash or Perfect) is applied to the received frame.

### Broadcast address filter

The MAC does not filter any broadcast frames in the default mode. However, if the MAC is programmed to reject all broadcast frames by setting the BFD bit in the Frame filter register, any broadcast frames are dropped.

### Unicast source address filter

The MAC can also perform perfect filtering based on the source address field of the received frames. By default, the MAC compares the SA field with the values programmed in the SA registers. The MAC address registers [1:3] can be configured to contain SA instead of DA for comparison, by setting bit 30 in the corresponding register. Group filtering with SA is also supported. The frames that fail the SA filter are dropped by the MAC if the SAF bit in the Frame filter register is set. Otherwise, the result of the SA filter is given as a status bit in the Receive Status word (see [RDES0: Receive descriptor Word0](#)).

When the SAF bit is set, the result of the SA and DA filters is AND'ed to decide whether the frame needs to be forwarded. This means that either of the filter fail result will drop the frame. Both filters have to pass the frame for the frame to be forwarded to the application.

### Inverse filtering operation

For both destination and source address filtering, there is an option to invert the filter-match result at the final output. These are controlled by the DAIF and SAIF bits in the Frame filter register, respectively. The DAIF bit is applicable for both Unicast and Multicast DA frames. The result of the unicast/multicast destination address filter is inverted in this mode. Similarly, when the SAIF bit is set, the result of the unicast SA filter is inverted. [Table 283](#) and [Table 284](#) summarize destination and source address filtering based on the type of frame received.

**Table 283. Destination address filtering**

Frame type	PM	HPF	HU	DAIF	HM	PAM	DB	DA filter operation
Broadcast	1	X	X	X	X	X	X	Pass
	0	X	X	X	X	X	0	Pass
	0	X	X	X	X	X	1	Fail
Unicast	1	X	X	X	X	X	X	Pass all frames
	0	X	0	0	X	X	X	Pass on perfect/group filter match
	0	X	0	1	X	X	X	Fail on perfect/Group filter match
	0	0	1	0	X	X	X	Pass on hash filter match
	0	0	1	1	X	X	X	Fail on hash filter match
	0	1	1	0	X	X	X	Pass on hash or perfect/Group filter match
	0	1	1	1	X	X	X	Fail on hash or perfect/Group filter match
Multicast	1	X	X	X	X	X	X	Pass all frames
	X	X	X	X	X	1	X	Pass all frames
	0	X	X	0	0	0	X	Pass on Perfect/Group filter match and drop PAUSE control frames if PCF = 0x
	0	0	X	0	1	0	X	Pass on hash filter match and drop PAUSE control frames if PCF = 0x
	0	1	X	0	1	0	X	Pass on hash or perfect/Group filter match and drop PAUSE control frames if PCF = 0x
	0	X	X	1	0	0	X	Fail on perfect/Group filter match and drop PAUSE control frames if PCF = 0x
	0	0	X	1	1	0	X	Fail on hash filter match and drop PAUSE control frames if PCF = 0x
	0	1	X	1	1	0	X	Fail on hash or perfect/Group filter match and drop PAUSE control frames if PCF = 0x

**Table 284. Source address filtering**

Frame type	PM	SAIF	SAF	SA filter operation
Unicast	1	X	X	Pass all frames
	0	0	0	Pass status on perfect/Group filter match but do not drop frames that fail
	0	1	0	Fail status on perfect/group filter match but do not drop frame
	0	0	1	Pass on perfect/group filter match and drop frames that fail
	0	1	1	Fail on perfect/group filter match and drop frames that fail

### 38.5.6 MAC loopback mode

The MAC supports loopback of transmitted frames onto its receiver. By default, the MAC loopback function is disabled, but this feature can be enabled by programming the Loopback bit in the MAC ETH\_MACCR register.

### 38.5.7 MAC management counters: MMC

The MAC management counters (MMC) maintain a set of registers for gathering statistics on the received and transmitted frames. These include a control register for controlling the behavior of the registers, two 32-bit registers containing generated interrupts (receive and transmit), and two 32-bit registers containing masks for the Interrupt register (receive and transmit). These registers are accessible from the application. Each register is 32 bits wide.

[Section 38.8: Ethernet registers](#) describes the various counters and lists the addresses of each of the statistics counters. This address is used for read/write accesses to the desired transmit/receive counter.

The Receive MMC counters are updated for frames that pass address filtering. Dropped frames statistics are not updated unless the dropped frames are runt frames of less than 6 bytes (DA bytes are not received fully).

#### Good transmitted and received frames

Transmitted frames are considered “good” if transmitted successfully. In other words, a transmitted frame is good if the frame transmission is not aborted due to any of the following errors:

- + Jabber Timeout
- + No Carrier/Loss of Carrier
- + Late Collision
- + Frame Underflow
- + Excessive Deferral
- + Excessive Collision

Received frames are considered “good” if none of the following errors exists:

- + CRC error
- + Runt Frame (shorter than 64 bytes)
- + Alignment error (in 10/ 100 Mbit/s only)
- + Length error (non-Type frames only)
- + Out of Range (non-Type frames only, longer than maximum size)
- + MII\_RXER Input error

The maximum frame size depends on the frame type, as follows:

- + Untagged frame maxsize = 1518
- + VLAN Frame maxsize = 1522

### 38.5.8 Power management: PMT

This section describes the power management (PMT) mechanisms supported by the MAC. PMT supports the reception of network (remote) wakeup frames and Magic Packet frames. PMT generates interrupts for wakeup frames and Magic Packets received by the MAC. The PMT block is enabled with remote wakeup frame enable and Magic Packet enable. These enable bits (WFE and MPE) are in the ETH\_MACPMTCSR register and are programmed by the application. When the power down mode is enabled in the PMT, then all received frames are dropped by the MAC and they are not forwarded to the application. The MAC comes out of the power down mode only when either a Magic Packet or a Remote wakeup frame is received and the corresponding detection is enabled.

#### Remote wakeup frame filter register

There are eight wakeup frame filter registers. To write on each of them, load the wakeup frame filter register value by value. The wanted values of the wakeup frame filter are loaded by sequentially loading eight times the wakeup frame filter register. The read operation is identical to the write operation. To read the eight values, the user has to read eight times the wakeup frame filter register to reach the last register. Each read/write points the wakeup frame filter register to the next filter register.

Figure 507. Wakeup frame filter register

The diagram illustrates the structure of the Wakeup frame filter register. It consists of eight rows, each representing a register from reg0 to reg7. Each row contains four fields: Filter i Byte Mask, Filter i Command, Filter i Offset, and Filter i CRC-16.

Wakeup frame filter reg0	Filter 0 Byte Mask							
Wakeup frame filter reg1	Filter 1 Byte Mask							
Wakeup frame filter reg2	Filter 2 Byte Mask							
Wakeup frame filter reg3	Filter 3 Byte Mask							
Wakeup frame filter reg4	RSVD	Filter 3 Command	RSVD	Filter 2 Command	RSVD	Filter 1 Command	RSVD	Filter 0 Command
Wakeup frame filter reg5	Filter 3 Offset		Filter 2 Offset		Filter 1 Offset		Filter 0 Offset	
Wakeup frame filter reg6	Filter 1 CRC - 16				Filter 0 CRC - 16			
Wakeup frame filter reg7	Filter 3 CRC - 16				Filter 2 CRC - 16			

ai15647

- Filter i Byte Mask

This register defines which bytes of the frame are examined by filter i (0, 1, 2, and 3) in order to determine whether or not the frame is a wakeup frame. The MSB (thirty-first bit) must be zero. Bit j [30:0] is the Byte Mask. If bit j (byte number) of the Byte Mask is set, then Filter i Offset + j of the incoming frame is processed by the CRC block; otherwise Filter i Offset + j is ignored.

- Filter i Command

This 4-bit command controls the filter i operation. Bit 3 specifies the address type, defining the pattern's destination address type. When the bit is set, the pattern applies to only multicast frames. When the bit is reset, the pattern applies only to unicast frames. Bit 2 and bit 1 are reserved. Bit 0 is the enable bit for filter i; if bit 0 is not set, filter i is disabled.

- Filter i Offset

This register defines the offset (within the frame) from which the frames are examined by filter i. This 8-bit pattern offset is the offset for the filter i first byte to be examined. The minimum allowed is 12, which refers to the 13th byte of the frame (offset value 0 refers to the first byte of the frame).

- Filter i CRC-16

This register contains the CRC\_16 value calculated from the pattern, as well as the byte mask programmed to the wakeup filter register block.

### Remote wakeup frame detection

When the MAC is in sleep mode and the remote wakeup bit is enabled in the ETH\_MACPMTCSR register, normal operation is resumed after receiving a remote wakeup frame. The application writes all eight wakeup filter registers, by performing a sequential write to the wakeup frame filter register address. The application enables remote wakeup by writing a 1 to bit 2 in the ETH\_MACPMTCSR register. PMT supports four programmable filters that provide different receive frame patterns. If the incoming frame passes the address filtering of Filter Command, and if Filter CRC-16 matches the incoming examined pattern, then the wakeup frame is received. Filter\_offset (minimum value 12, which refers to the 13th byte of the frame) determines the offset from which the frame is to be examined. Filter Byte Mask determines which bytes of the frame must be examined. The thirty-first bit of Byte Mask must be set to zero. The wakeup frame is checked only for length error, FCS error, dribble bit error, MII error, collision, and to ensure that it is not a runt frame. Even if the

wakeup frame is more than 512 bytes long, if the frame has a valid CRC value, it is considered valid. Wakeup frame detection is updated in the ETH\_MACPMTCSR register for every remote wakeup frame received. If enabled, a PMT interrupt is generated to indicate the reception of a remote wakeup frame.

### Magic packet detection

The Magic Packet frame is based on a method that uses Advanced Micro Device's Magic Packet technology to power up the sleeping device on the network. The MAC receives a specific packet of information, called a Magic Packet, addressed to the node on the network. Only Magic Packets that are addressed to the device or a broadcast address are checked to determine whether they meet the wakeup requirements. Magic Packets that pass address filtering (unicast or broadcast) are checked to determine whether they meet the remote Wake-on-LAN data format of 6 bytes of all ones followed by a MAC address appearing 16 times. The application enables Magic Packet wakeup by writing a 1 to bit 1 in the ETH\_MACPMTCSR register. The PMT block constantly monitors each frame addressed to the node for a specific Magic Packet pattern. Each received frame is checked for a 0xFFFF FFFF FFFF pattern following the destination and source address field. The PMT block then checks the frame for 16 repetitions of the MAC address without any breaks or interruptions. In case of a break in the 16 repetitions of the address, the 0xFFFF FFFF FFFF pattern is scanned for again in the incoming frame. The 16 repetitions can be anywhere in the frame, but must be preceded by the synchronization stream (0xFFFF FFFF FFFF). The device also accepts a multicast frame, as long as the 16 duplications of the MAC address are detected. If the MAC address of a node is 0x0011 2233 4455, then the MAC scans for the data sequence:

```
Destination address source address ..... FFFF FFFF FFFF  
0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455  
0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455  
0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455  
0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455  
...CRC
```

Magic Packet detection is updated in the ETH\_MACPMTCSR register for received Magic Packet. If enabled, a PMT interrupt is generated to indicate the reception of a Magic Packet.

### System consideration during power-down

The Ethernet PMT block is able to detect frames while the system is in the Stop mode, provided that the EXTI line 19 is enabled.

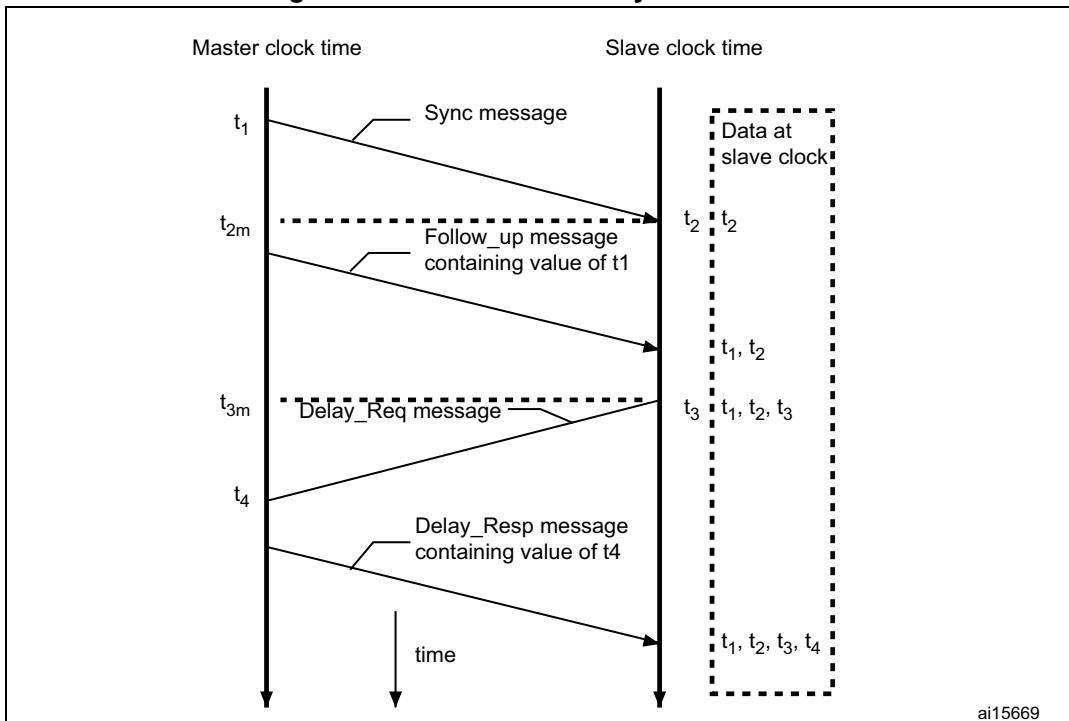
The MAC receiver state machine should remain enabled during the power-down mode. This means that the RE bit has to remain set in the ETH\_MACCR register because it is involved in magic packet/ wake-on-LAN frame detection. The transmit state machine should however be turned off during the power-down mode by clearing the TE bit in the ETH\_MACCR register. Moreover, the Ethernet DMA should be disabled during the power-down mode, because it is not necessary to copy the magic packet/wake-on-LAN frame into the SRAM. To disable the Ethernet DMA, clear the ST bit and the SR bit (for the transmit DMA and the receive DMA, respectively) in the ETH\_DMAOMR register.

The recommended power-down and wakeup sequences are as follows:

1. Disable the transmit DMA and wait for any previous frame transmissions to complete. These transmissions can be detected when the transmit interrupt ETH\_DMASR register[0] is received.
2. Disable the MAC transmitter and MAC receiver by clearing the RE and TE bits in the ETH\_MACCR configuration register.
3. Wait for the receive DMA to have emptied all the frames in the Rx FIFO.
4. Disable the receive DMA.
5. Configure and enable the EXTI line 19 to generate either an event or an interrupt.
6. If you configure the EXTI line 19 to generate an interrupt, you also have to correctly configure the ETH\_WKUP\_IRQ Handler function, which should clear the pending bit of the EXTI line 19.
7. Enable Magic packet/Wake-on-LAN frame detection by setting the MFE/ WFE bit in the ETH\_MACPMTCSR register.
8. Enable the MAC power-down mode, by setting the PD bit in the ETH\_MACPMTCSR register.
9. Enable the MAC Receiver by setting the RE bit in the ETH\_MACCR register.
10. Enter the system's Stop mode (for more details refer to [Section 4.3.5: Stop mode](#)):
11. On receiving a valid wakeup frame, the Ethernet peripheral exits the power-down mode.
12. Read the ETH\_MACPMTCSR to clear the power management event flag, enable the MAC transmitter state machine, and the receive and transmit DMA.
13. Configure the system clock: enable the HSE and set the clocks.

### 38.5.9 Precision time protocol (IEEE1588 PTP)

The IEEE 1588 standard defines a protocol that allows precise clock synchronization in measurement and control systems implemented with technologies such as network communication, local computing and distributed objects. The protocol applies to systems that communicate by local area networks supporting multicast messaging, including (but not limited to) Ethernet. This protocol is used to synchronize heterogeneous systems that include clocks of varying inherent precision, resolution and stability. The protocol supports system-wide synchronization accuracy in the submicrosecond range with minimum network and local clock computing resources. The message-based protocol, known as the precision time protocol (PTP), is transported over UDP/IP. The system or network is classified into Master and Slave nodes for distributing the timing/clock information. The protocol's technique for synchronizing a slave node to a master node by exchanging PTP messages is described in [Figure 508](#).

**Figure 508. Networked time synchronization**

1. The master broadcasts PTP Sync messages to all its nodes. The Sync message contains the master's reference time information. The time at which this message leaves the master's system is  $t_1$ . For Ethernet ports, this time has to be captured at the MII.
2. A slave receives the Sync message and also captures the exact time,  $t_2$ , using its timing reference.
3. The master then sends the slave a Follow\_up message, which contains the  $t_1$  information for later use.
4. The slave sends the master a Delay\_Req message, noting the exact time,  $t_3$ , at which this frame leaves the MII.
5. The master receives this message and captures the exact time,  $t_4$ , at which it enters its system.
6. The master sends the  $t_4$  information to the slave in the Delay\_Resp message.
7. The slave uses the four values of  $t_1$ ,  $t_2$ ,  $t_3$ , and  $t_4$  to synchronize its local timing reference to the master's timing reference.

Most of the protocol implementation occurs in the software, above the UDP layer. As described above, however, hardware support is required to capture the exact time when specific PTP packets enter or leave the Ethernet port at the MII. This timing information has to be captured and returned to the software for a proper, high-accuracy implementation of PTP.

### Reference timing source

To get a snapshot of the time, the core requires a reference time in 64-bit format (split into two 32-bit channels, with the upper 32 bits providing time in seconds, and the lower 32 bits indicating time in nanoseconds) as defined in the IEEE 1588 specification.

The PTP reference clock input is used to internally generate the reference time (also called the System Time) and to capture time stamps. The frequency of this reference clock must be greater than or equal to the resolution of time stamp counter. The synchronization accuracy target between the master node and the slaves is around 100 ns.

The generation, update and modification of the System Time are described in the [System Time correction methods](#).

The accuracy depends on the PTP reference clock input period, the characteristics of the oscillator (drift) and the frequency of the synchronization procedure.

Due to the synchronization from the Tx and Rx clock input domain to the PTP reference clock domain, the uncertainty on the time stamp latched value is 1 reference clock period. If we add the uncertainty due to resolution, we will add half the period for time stamping.

### Transmission of frames with the PTP feature

When a frame's SFD is output on the MII, a time stamp is captured. Frames for which time stamp capture is required are controllable on a per-frame basis. In other words, each transmitted frame can be marked to indicate whether a time stamp must be captured or not for that frame. The transmitted frames are not processed to identify PTP frames. Frame control is exercised through the control bits in the transmit descriptor. Captured time stamps are returned to the application in the same way as the status is provided for frames. The time stamp is sent back along with the Transmit status of the frame, inside the corresponding transmit descriptor, thus connecting the time stamp automatically to the specific PTP frame. The 64-bit time stamp information is written back to the TDES2 and TDES3 fields, with TDES2 holding the time stamp's 32 least significant bits.

### Reception of frames with the PTP feature

When the IEEE 1588 time stamping feature is enabled, the Ethernet MAC captures the time stamp of all frames received on the MII. The MAC provides the time stamp as soon as the frame reception is complete. Captured time stamps are returned to the application in the same way as the frame status is provided. The time stamp is sent back along with the Receive status of the frame, inside the corresponding receive descriptor. The 64-bit time stamp information is written back to the RDES2 and RDES3 fields, with RDES2 holding the time stamp's 32 least significant bits.

### System Time correction methods

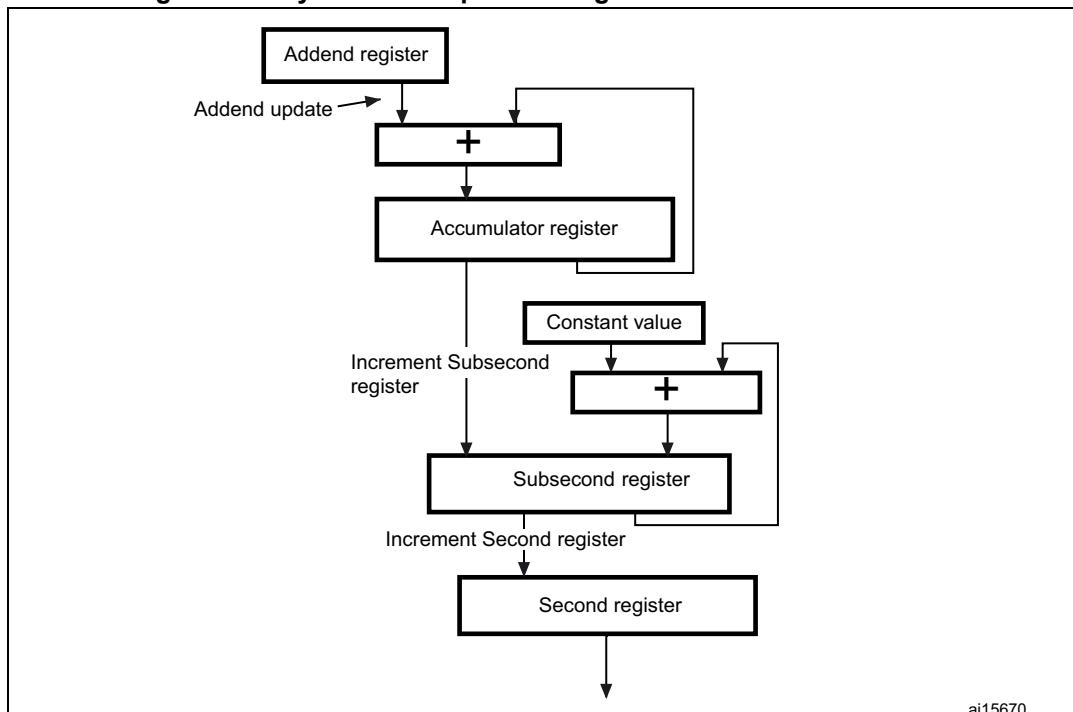
The 64-bit PTP time is updated using the PTP input reference clock, HCLK. This PTP time is used as a source to take snapshots (time stamps) of the Ethernet frames being transmitted or received at the MII. The System Time counter can be initialized or corrected using either the Coarse or the Fine correction method.

In the Coarse correction method, the initial value or the offset value is written to the Time stamp update register (refer to [Section 38.8.3: IEEE 1588 time stamp registers on page 1611](#)). For initialization, the System Time counter is written with the value in the Time stamp update registers, whereas for system time correction, the offset value (Time stamp update register) is added to or subtracted from the system time.

In the Fine correction method, the slave clock (reference clock) frequency drift with respect to the master clock (as defined in IEEE 1588) is corrected over a period of time, unlike in the Coarse correction method where it is corrected in a single clock cycle. The longer correction time helps maintain linear time and does not introduce drastic changes (or a large jitter) in the reference time between PTP Sync message intervals. In this method, an accumulator

sums up the contents of the Addend register as shown in [Figure 509](#). The arithmetic carry that the accumulator generates is used as a pulse to increment the system time counter. The accumulator and the addend are 32-bit registers. Here, the accumulator acts as a high-precision frequency multiplier or divider. [Figure 509](#) shows this algorithm.

**Figure 509. System time update using the Fine correction method**



The system time update logic requires a 50 MHz clock frequency to achieve 20 ns accuracy. The frequency division is the ratio of the reference clock frequency to the required clock frequency. Hence, if the reference clock (HCLK) is, let us say, 66 MHz, the ratio is calculated as  $66\text{ MHz}/50\text{ MHz} = 1.32$ . Hence, the default addend value to be set in the register is  $2^{32}/1.32$ , which is equal to 0xC1F0 7C1F.

If the reference clock drifts lower, to 65 MHz for example, the ratio is  $65/50$  or 1.3 and the value to set in the addend register is  $2^{32}/1.30$  equal to 0xC4EC 4EC4. If the clock drifts higher, to 67 MHz for example, the addend register must be set to 0xBF0 B7672. When the clock drift is zero, the default addend value of 0xC1F0 7C1F ( $2^{32}/1.32$ ) should be programmed.

In [Figure 509](#), the constant value used to increment the subsecond register is 0d43. This makes an accuracy of 20 ns in the system time (in other words, it is incremented by 20 ns steps).

The software has to calculate the drift in frequency based on the Sync messages, and to update the Addend register accordingly. Initially, the slave clock is set with FreqCompensationValue0 in the Addend register. This value is as follows:

$$\text{FreqCompensationValue0} = 2^{32} / \text{FreqDivisionRatio}$$

If MasterToSlaveDelay is initially assumed to be the same for consecutive Sync messages, the algorithm described below must be applied. After a few Sync cycles, frequency lock occurs. The slave clock can then determine a precise MasterToSlaveDelay value and resynchronize with the master using the new value.

The algorithm is as follows:

- At time MasterSyncTime (n) the master sends the slave clock a Sync message. The slave receives this message when its local clock is SlaveClockTime (n) and computes MasterClockTime (n) as:  

$$\text{MasterClockTime (n)} = \text{MasterSyncTime (n)} + \text{MasterToSlaveDelay (n)}$$
- The master clock count for current Sync cycle, MasterClockCount (n) is given by:  

$$\text{MasterClockCount (n)} = \text{MasterClockTime (n)} - \text{MasterClockTime (n - 1)}$$
 (assuming that MasterToSlaveDelay is the same for Sync cycles n and n - 1)
- The slave clock count for current Sync cycle, SlaveClockCount (n) is given by:  

$$\text{SlaveClockCount (n)} = \text{SlaveClockTime (n)} - \text{SlaveClockTime (n - 1)}$$
- The difference between master and slave clock counts for current Sync cycle, ClockDiffCount (n) is given by:  

$$\text{ClockDiffCount (n)} = \text{MasterClockCount (n)} - \text{SlaveClockCount (n)}$$
- The frequency-scaling factor for slave clock, FreqScaleFactor (n) is given by:  

$$\text{FreqScaleFactor (n)} = (\text{MasterClockCount (n)} + \text{ClockDiffCount (n)}) / \text{SlaveClockCount (n)}$$
- The frequency compensation value for Addend register, FreqCompensationValue (n) is given by:  

$$\text{FreqCompensationValue (n)} = \text{FreqScaleFactor (n)} \times \text{FreqCompensationValue (n - 1)}$$

In theory, this algorithm achieves lock in one Sync cycle; however, it may take several cycles, due to changing network propagation delays and operating conditions.

This algorithm is self-correcting: if for any reason the slave clock is initially set to a value from the master that is incorrect, the algorithm corrects it at the cost of more Sync cycles.

### Programming steps for system time generation initialization

The time stamping feature can be enabled by setting bit 0 in the Time stamp control register (ETH\_\_PTPTSCR). However, it is essential to initialize the time stamp counter after this bit is set to start time stamp operation. The proper sequence is the following:

1. Mask the Time stamp trigger interrupt by setting bit 9 in the MACIMR register.
2. Program Time stamp register bit 0 to enable time stamping.
3. Program the Subsecond increment register based on the PTP clock frequency.
4. If you are using the Fine correction method, program the Time stamp addend register and set Time stamp control register bit 5 (addend register update).
5. Poll the Time stamp control register until bit 5 is cleared.
6. To select the Fine correction method (if required), program Time stamp control register bit 1.
7. Program the Time stamp high update and Time stamp low update registers with the appropriate time value.
8. Set Time stamp control register bit 2 (Time stamp init).
9. The Time stamp counter starts operation as soon as it is initialized with the value written in the Time stamp update register.
10. Enable the MAC receiver and transmitter for proper time stamping.

**Note:** *If time stamp operation is disabled by clearing bit 0 in the ETH\_PTPTSCR register, the above steps must be repeated to restart the time stamp operation.*

### Programming steps for system time update in the Coarse correction method

To synchronize or update the system time in one process (coarse correction method), perform the following steps:

1. Write the offset (positive or negative) in the Time stamp update high and low registers.
2. Set bit 3 (TSSTU) in the Time stamp control register.
3. The value in the Time stamp update registers is added to or subtracted from the system time when the TSSTU bit is cleared.

### Programming steps for system time update in the Fine correction method

To synchronize or update the system time to reduce system-time jitter (fine correction method), perform the following steps:

1. With the help of the algorithm explained in [System Time correction methods](#), calculate the rate by which you want to speed up or slow down the system time increments.
2. Update the time stamp.
3. Wait the time you want the new value of the Addend register to be active. You can do this by activating the Time stamp trigger interrupt after the system time reaches the target value.
4. Program the required target time in the Target time high and low registers. Unmask the Time stamp interrupt by clearing bit 9 in the ETH\_MACIMR register.
5. Set Time stamp control register bit 4 (TSARU).
6. When this trigger causes an interrupt, read the ETH\_MACSR register.
7. Reprogram the Time stamp addend register with the old value and set ETH\_TPTSCR bit 5 again.

### PTP trigger internal connection with TIM2

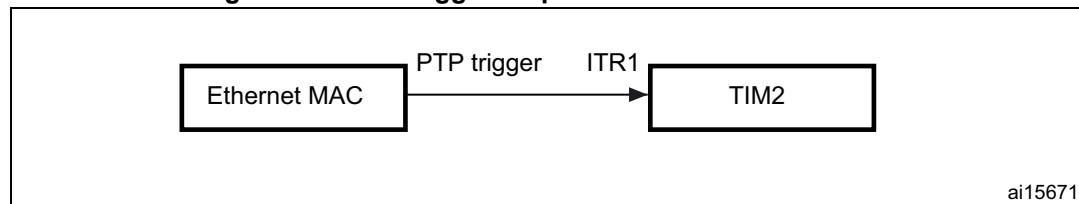
The MAC provides a trigger interrupt when the system time becomes greater than the target time. Using an interrupt introduces a known latency plus an uncertainty in the command execution time.

In order to avoid this uncertainty, a PTP trigger output signal is set high when the system time is greater than the target time. It is internally connected to the TIM2 input trigger. With this signal, the input capture feature, the output compare feature and the waveforms of the timer can be used, triggered by the synchronized PTP system time. No uncertainty is introduced since the clock of the timer (PCLK1: TIM2 APB1 clock) and PTP reference clock (HCLK) are synchronous.

This PTP trigger signal is connected to the TIM2 ITR1 input selectable by software. The connection is enabled through bits 11 and 10 in the TIM2 option register (TIM2\_OR).

[Figure 510](#) shows the connection.

**Figure 510. PTP trigger output to TIM2 ITR1 connection**



### PTP pulse-per-second output signal

This PTP pulse output is used to check the synchronization between all nodes in the network. To be able to test the difference between the local slave clock and the master reference clock, both clocks were given a pulse-per-second (PPS) output signal that may be connected to an oscilloscope if necessary. The deviation between the two signals can therefore be measured. The pulse width of the PPS output is 125 ms.

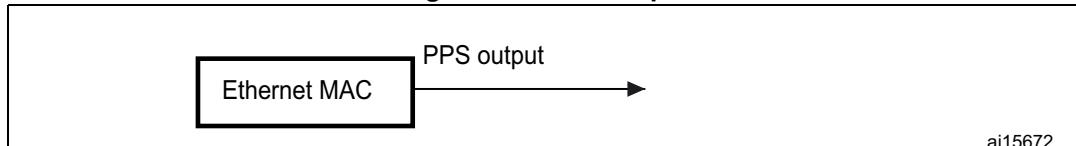
The PPS output is enabled through bits 11 and 10 in the TIM2 option register (TIM2\_OR).

The default frequency of the PPS output is 1 Hz. PPSFREQ[3:0] (in ETH\_PTPPPSCR) can be used to set the frequency of the PPS output to  $2^{PPSFREQ}$  Hz.

When set to 1 Hz, the PPS pulse width is 125 ms with binary rollover (TSSSR=0, bit 9 in ETH\_PTPTSCR) and 100 ms with digital rollover (TSSSR=1). When set to 2 Hz and higher, the duty cycle of the PPS output is 50% with binary rollover.

With digital rollover (TSSSR=1), it is recommended not to use the PPS output with a frequency other than 1 Hz as it would have irregular waveforms (though its average frequency would always be correct during any one-second window).

**Figure 511. PPS output**



ai15672

## 38.6 Ethernet functional description: DMA controller operation

The DMA has independent transmit and receive engines, and a CSR space. The transmit engine transfers data from system memory into the Tx FIFO while the receive engine transfers data from the Rx FIFO into system memory. The controller utilizes descriptors to efficiently move data from source to destination with minimum CPU intervention. The DMA is designed for packet-oriented data transfers such as frames in Ethernet. The controller can be programmed to interrupt the CPU in cases such as frame transmit and receive transfer completion, and other normal/error conditions. The DMA and the STM32F75xxx and STM32F74xxx communicate through two data structures:

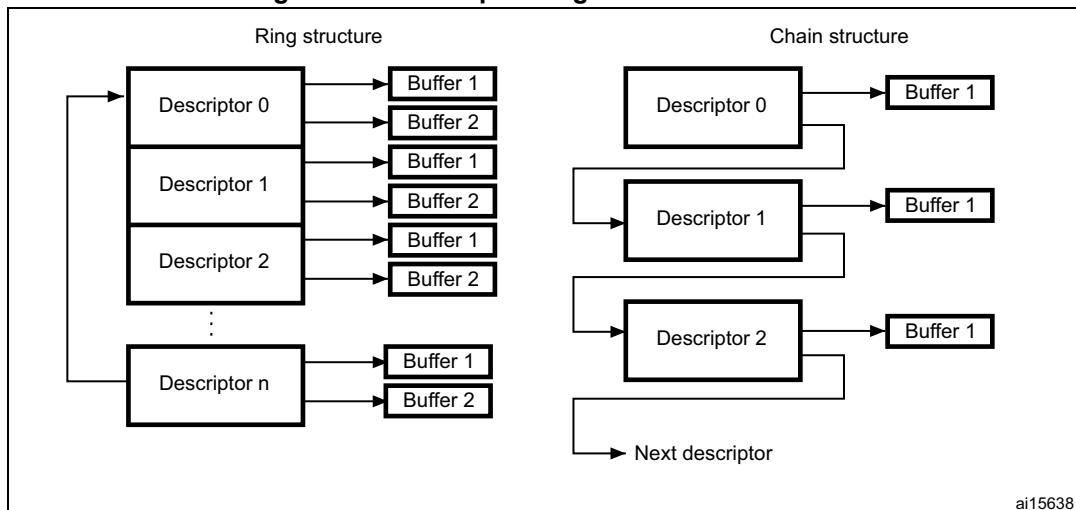
- Control and status registers (CSR)
- Descriptor lists and data buffers.

Control and status registers are described in detail in [Section 38.8: Ethernet registers](#). Descriptors are described in detail in [Normal Tx DMA descriptors](#).

The DMA transfers the received data frames to the receive buffer in the STM32F75xxx and STM32F74xxx memory, and transmits data frames from the transmit buffer in the STM32F75xxx and STM32F74xxx memory. Descriptors that reside in the STM32F75xxx and STM32F74xxx memory act as pointers to these buffers. There are two descriptor lists: one for reception, and one for transmission. The base address of each list is written into DMA Registers 3 and 4, respectively. A descriptor list is forward-linked (either implicitly or explicitly). The last descriptor may point back to the first entry to create a ring structure. Explicit chaining of descriptors is accomplished by configuring the second address chained in both the receive and transmit descriptors (RDES1[14] and TDES0[20]). The descriptor lists reside in the Host's physical memory space. Each descriptor can point to a maximum of

two buffers. This enables the use of two physically addressed buffers, instead of two contiguous buffers in memory. A data buffer resides in the Host's physical memory space, and consists of an entire frame or part of a frame, but cannot exceed a single frame. Buffers contain only data. The buffer status is maintained in the descriptor. Data chaining refers to frames that span multiple data buffers. However, a single descriptor cannot span multiple frames. The DMA skips to the next frame buffer when the end of frame is detected. Data chaining can be enabled or disabled. The descriptor ring and chain structure is shown in [Figure 512](#).

**Figure 512. Descriptor ring and chain structure**



ai15638

### 38.6.1 Initialization of a transfer using DMA

Initialization for the MAC is as follows:

1. Write to ETH\_DMABMR to set STM32F75xxx and STM32F74xxx bus access parameters.
2. Write to the ETH\_DMAIER register to mask unnecessary interrupt causes.
3. The software driver creates the transmit and receive descriptor lists. Then it writes to both the ETH\_DMARDLAR and ETH\_DMATDLAR registers, providing the DMA with the start address of each list.
4. Write to MAC Registers 1, 2, and 3 to choose the desired filtering options.
5. Write to the MAC ETH\_MACCR register to configure and enable the transmit and receive operating modes. The PS and DM bits are set based on the auto-negotiation result (read from the PHY).
6. Write to the ETH\_DMAOMR register to set bits 13 and 1 and start transmission and reception.
7. The transmit and receive engines enter the running state and attempt to acquire descriptors from the respective descriptor lists. The receive and transmit engines then begin processing receive and transmit operations. The transmit and receive processes are independent of each other and can be started or stopped separately.

### 38.6.2 Host bus burst access

The DMA attempts to execute fixed-length burst transfers on the AHB master interface if configured to do so (FB bit in ETH\_DMABMR). The maximum burst length is indicated and

limited by the PBL field (ETH\_DMABMR [13:8]). The receive and transmit descriptors are always accessed in the maximum possible burst size (limited by PBL) for the 16 bytes to be read.

The Transmit DMA initiates a data transfer only when there is sufficient space in the Transmit FIFO to accommodate the configured burst or the number of bytes until the end of frame (when it is less than the configured burst length). The DMA indicates the start address and the number of transfers required to the AHB Master Interface. When the AHB Interface is configured for fixed-length burst, then it transfers data using the best combination of INCR4, INCR8, INCR16 and SINGLE transactions. Otherwise (no fixed-length burst), it transfers data using INCR (undefined length) and SINGLE transactions.

The Receive DMA initiates a data transfer only when sufficient data for the configured burst is available in Receive FIFO or when the end of frame (when it is less than the configured burst length) is detected in the Receive FIFO. The DMA indicates the start address and the number of transfers required to the AHB master interface. When the AHB interface is configured for fixed-length burst, then it transfers data using the best combination of INCR4, INCR8, INCR16 and SINGLE transactions. If the end of frame is reached before the fixed-burst ends on the AHB interface, then dummy transfers are performed in order to complete the fixed-length burst. Otherwise (FB bit in ETH\_DMABMR is reset), it transfers data using INCR (undefined length) and SINGLE transactions.

When the AHB interface is configured for address-aligned beats, both DMA engines ensure that the first burst transfer the AHB initiates is less than or equal to the size of the configured PBL. Thus, all subsequent beats start at an address that is aligned to the configured PBL. The DMA can only align the address for beats up to size 16 (for PBL > 16), because the AHB interface does not support more than INCR16.

### 38.6.3 Host data buffer alignment

The transmit and receive data buffers do not have any restrictions on start address alignment. In our system with 32-bit memory, the start address for the buffers can be aligned to any of the four bytes. However, the DMA always initiates transfers with address aligned to the bus width with dummy data for the byte lanes not required. This typically happens during the transfer of the beginning or end of an Ethernet frame.

- Example of buffer read:

If the Transmit buffer address is 0x0000 0FF2, and 15 bytes need to be transferred, then the DMA will read five full words from address 0x0000 0FF0, but when transferring data to the Transmit FIFO, the extra bytes (the first two bytes) will be dropped or ignored. Similarly, the last 3 bytes of the last transfer will also be ignored. The DMA always ensures it transfers a full 32-bit data items to the Transmit FIFO, unless it is the end of frame.

- Example of buffer write:

If the Receive buffer address is 0x0000 0FF2, and 16 bytes of a received frame need to be transferred, then the DMA will write five full 32-bit data items from address 0x0000 0FF0. But the first 2 bytes of the first transfer and the last 2 bytes of the third transfer will have dummy data.

### 38.6.4 Buffer size calculations

The DMA does not update the size fields in the transmit and receive descriptors. The DMA updates only the status fields (xDES0) of the descriptors. The driver has to calculate the sizes. The transmit DMA transfers the exact number of bytes (indicated by buffer size field in

TDES1) towards the MAC core. If a descriptor is marked as first (FS bit in TDES0 is set), then the DMA marks the first transfer from the buffer as the start of frame. If a descriptor is marked as last (LS bit in TDES0), then the DMA marks the last transfer from that data buffer as the end of frame. The receive DMA transfers data to a buffer until the buffer is full or the end of frame is received. If a descriptor is not marked as last (LS bit in RDES0), then the buffer(s) that correspond to the descriptor are full and the amount of valid data in a buffer is accurately indicated by the buffer size field minus the data buffer pointer offset when the descriptor's FS bit is set. The offset is zero when the data buffer pointer is aligned to the databus width. If a descriptor is marked as last, then the buffer may not be full (as indicated by the buffer size in RDES1). To compute the amount of valid data in this final buffer, the driver must read the frame length (FL bits in RDES0[29:16]) and subtract the sum of the buffer sizes of the preceding buffers in this frame. The receive DMA always transfers the start of next frame with a new descriptor.

**Note:** *Even when the start address of a receive buffer is not aligned to the system databus width the system should allocate a receive buffer of a size aligned to the system bus width. For example, if the system allocates a 1024 byte (1 KB) receive buffer starting from address 0x1000, the software can program the buffer start address in the receive descriptor to have a 0x1002 offset. The receive DMA writes the frame to this buffer with dummy data in the first two locations (0x1000 and 0x1001). The actual frame is written from location 0x1002. Thus, the actual useful space in this buffer is 1022 bytes, even though the buffer size is programmed as 1024 bytes, due to the start address offset.*

### 38.6.5 DMA arbiter

The arbiter inside the DMA takes care of the arbitration between transmit and receive channel accesses to the AHB master interface. Two types of arbitrations are possible: round-robin, and fixed-priority. When round-robin arbitration is selected (DA bit in ETH\_DMABMR is reset), the arbiter allocates the databus in the ratio set by the PM bits in ETH\_DMABMR, when both transmit and receive DMAs request access simultaneously. When the DA bit is set, the receive DMA always gets priority over the transmit DMA for data access.

### 38.6.6 Error response to DMA

For any data transfer initiated by a DMA channel, if the slave replies with an error response, that DMA stops all operations and updates the error bits and the fatal bus error bit in the Status register (ETH\_DMASR register). That DMA controller can resume operation only after soft- or hard-resetting the peripheral and re-initializing the DMA.

### 38.6.7 Tx DMA configuration

#### TxDMA operation: default (non-OSF) mode

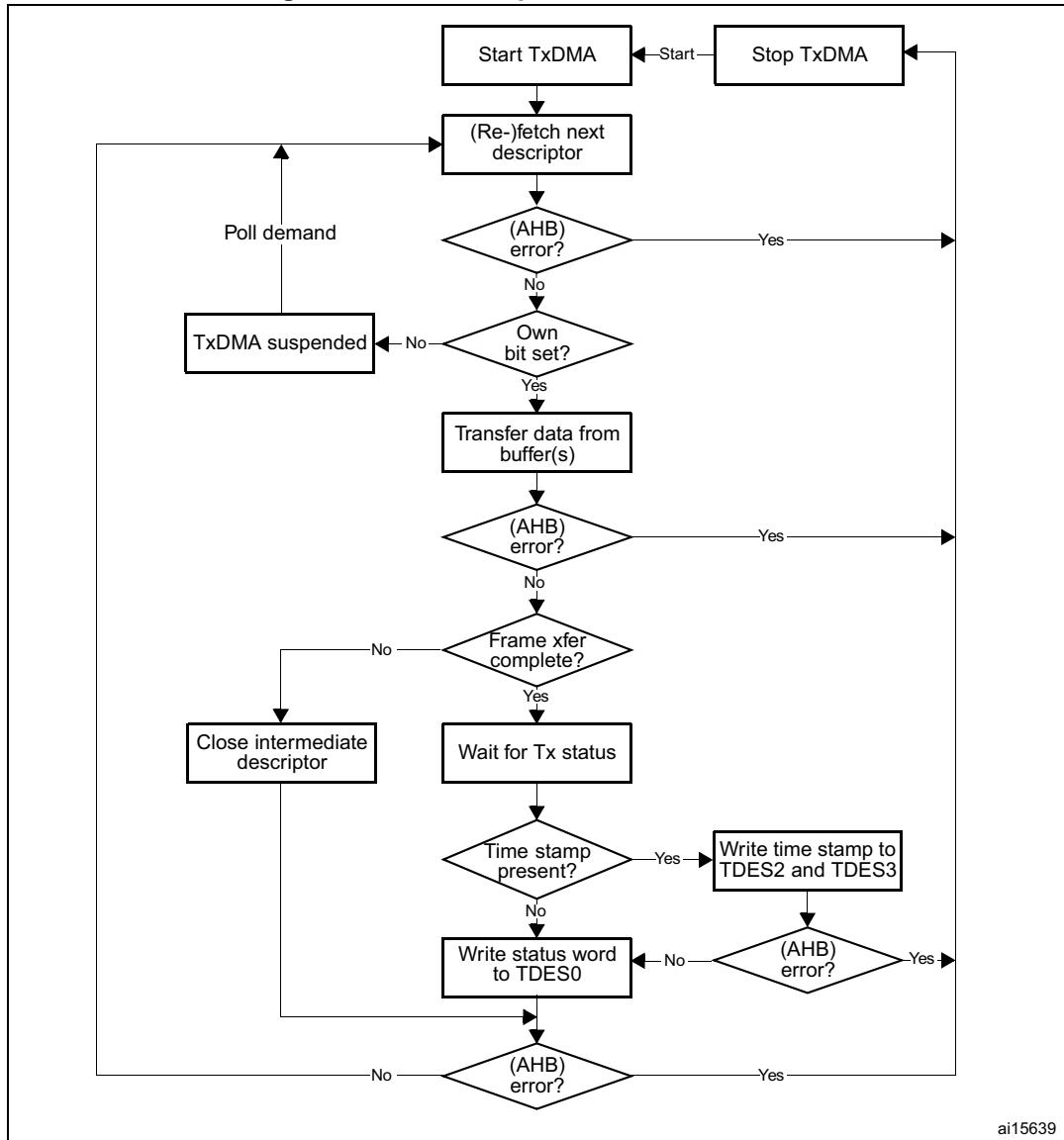
The transmit DMA engine in default mode proceeds as follows:

1. The user sets up the transmit descriptor (TDES0-TDES3) and sets the OWN bit (TDES0[31]) after setting up the corresponding data buffer(s) with Ethernet frame data.
2. Once the ST bit (ETH\_DMAOMR register[13]) is set, the DMA enters the Run state.
3. While in the Run state, the DMA polls the transmit descriptor list for frames requiring transmission. After polling starts, it continues in either sequential descriptor ring order or chained order. If the DMA detects a descriptor flagged as owned by the CPU, or if an error condition occurs, transmission is suspended and both the Transmit Buffer

- Unavailable (ETH\_DMASR register[2]) and Normal Interrupt Summary (ETH\_DMASR register[16]) bits are set. The transmit engine proceeds to Step 9.
4. If the acquired descriptor is flagged as owned by DMA (TDES0[31] is set), the DMA decodes the transmit data buffer address from the acquired descriptor.
  5. The DMA fetches the transmit data from the STM32F75xxx and STM32F74xxx memory and transfers the data.
  6. If an Ethernet frame is stored over data buffers in multiple descriptors, the DMA closes the intermediate descriptor and fetches the next descriptor. Steps 3, 4, and 5 are repeated until the end of Ethernet frame data is transferred.
  7. When frame transmission is complete, if IEEE 1588 time stamping was enabled for the frame (as indicated in the transmit status) the time stamp value is written to the transmit descriptor (TDES2 and TDES3) that contains the end-of-frame buffer. The status information is then written to this transmit descriptor (TDES0). Because the OWN bit is cleared during this step, the CPU now owns this descriptor. If time stamping was not enabled for this frame, the DMA does not alter the contents of TDES2 and TDES3.
  8. Transmit Interrupt (ETH\_DMASR register [0]) is set after completing the transmission of a frame that has Interrupt on Completion (TDES1[31]) set in its last descriptor. The DMA engine then returns to Step 3.
  9. In the Suspend state, the DMA tries to re-acquire the descriptor (and thereby returns to Step 3) when it receives a transmit poll demand, and the Underflow Interrupt Status bit is cleared.

*Figure 513* shows the TxDMA transmission flow in default mode.

Figure 513. TxDMA operation in default mode



ai15639

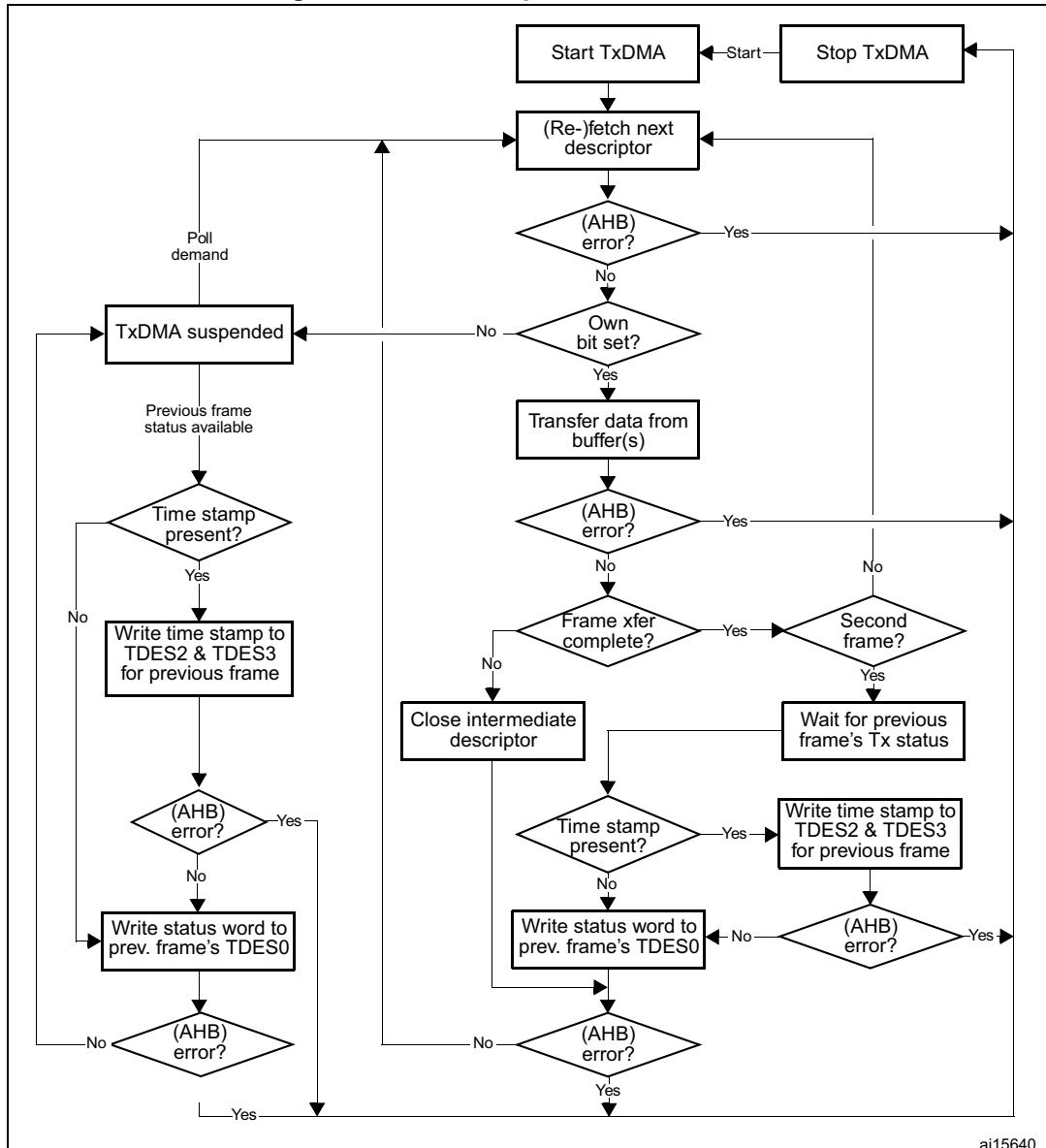
### TxDMA operation: OSF mode

While in the Run state, the transmit process can simultaneously acquire two frames without closing the Status descriptor of the first (if the OSF bit is set in ETH\_DMAOMR register[2]). As the transmit process finishes transferring the first frame, it immediately polls the transmit descriptor list for the second frame. If the second frame is valid, the transmit process transfers this frame before writing the first frame's status information. In OSF mode, the Run-state transmit DMA operates according to the following sequence:

1. The DMA operates as described in steps 1–6 of the TxDMA (default mode).
2. Without closing the previous frame's last descriptor, the DMA fetches the next descriptor.
3. If the DMA owns the acquired descriptor, the DMA decodes the transmit buffer address in this descriptor. If the DMA does not own the descriptor, the DMA goes into Suspend mode and skips to Step 7.
4. The DMA fetches the Transmit frame from the STM32F75xxx and STM32F74xxx memory and transfers the frame until the end of frame data are transferred, closing the intermediate descriptors if this frame is split across multiple descriptors.
5. The DMA waits for the transmission status and time stamp of the previous frame. When the status is available, the DMA writes the time stamp to TDES2 and TDES3, if such time stamp was captured (as indicated by a status bit). The DMA then writes the status, with a cleared OWN bit, to the corresponding TDES0, thus closing the descriptor. If time stamping was not enabled for the previous frame, the DMA does not alter the contents of TDES2 and TDES3.
6. If enabled, the Transmit interrupt is set, the DMA fetches the next descriptor, then proceeds to Step 3 (when Status is normal). If the previous transmission status shows an underflow error, the DMA goes into Suspend mode (Step 7).
7. In Suspend mode, if a pending status and time stamp are received by the DMA, it writes the time stamp (if enabled for the current frame) to TDES2 and TDES3, then writes the status to the corresponding TDES0. It then sets relevant interrupts and returns to Suspend mode.
8. The DMA can exit Suspend mode and enter the Run state (go to Step 1 or Step 2 depending on pending status) only after receiving a Transmit Poll demand (ETH\_DMATPDR register).

*Figure 514* shows the basic flowchart in OSF mode.

**Figure 514. TxDMA operation in OSF mode**



## Transmit frame processing

The transmit DMA expects that the data buffers contain complete Ethernet frames, excluding preamble, pad bytes, and FCS fields. The DA, SA, and Type/Len fields contain valid data. If the transmit descriptor indicates that the MAC core must disable CRC or pad insertion, the buffer must have complete Ethernet frames (excluding preamble), including the CRC bytes. Frames can be data-chained and span over several buffers. Frames have to be delimited by the first descriptor (TDES0[28]) and the last descriptor (TDES0[29]). As the transmission starts, TDES0[28] has to be set in the first descriptor. When this occurs, the frame data are transferred from the memory buffer to the Transmit FIFO. Concurrently, if the last descriptor (TDES0[29]) of the current frame is cleared, the transmit process attempts to acquire the next descriptor. The transmit process expects TDES0[28] to be cleared in this descriptor. If TDES0[29] is cleared, it indicates an intermediary buffer. If TDES0[29] is set, it indicates the last buffer of the frame. After the last buffer of the frame has been transmitted,

the DMA writes back the final status information to the transmit descriptor 0 (TDES0) word of the descriptor that has the last segment set in transmit descriptor 0 (TDES0[29]). At this time, if Interrupt on Completion (TDES0[30]) is set, Transmit Interrupt (in ETH\_DMASR register [0]) is set, the next descriptor is fetched, and the process repeats. Actual frame transmission begins after the Transmit FIFO has reached either a programmable transmit threshold (ETH\_DMAOMR register[16:14]), or a full frame is contained in the FIFO. There is also an option for the Store and forward mode (ETH\_DMAOMR register[21]). Descriptors are released (OWN bit TDES0[31] is cleared) when the DMA finishes transferring the frame.

### Transmit polling suspended

Transmit polling can be suspended by either of the following conditions:

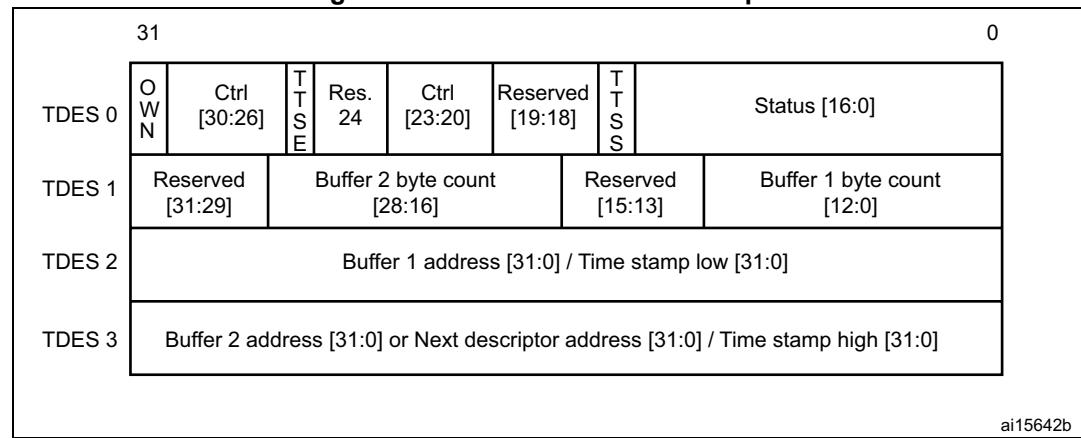
- The DMA detects a descriptor owned by the CPU (TDES0[31]=0) and the Transmit buffer unavailable flag is set (ETH\_DMASR register[2]). To resume, the driver must give descriptor ownership to the DMA and then issue a Poll Demand command.
- A frame transmission is aborted when a transmit error due to underflow is detected. The appropriate Transmit Descriptor 0 (TDES0) bit is set. If the second condition occurs, both the Abnormal Interrupt Summary (in ETH\_DMASR register [15]) and Transmit Underflow bits (in ETH\_DMASR register[5]) are set, and the information is written to Transmit Descriptor 0, causing the suspension. If the DMA goes into Suspend state due to the first condition, then both the Normal Interrupt Summary (ETH\_DMASR register [16]) and Transmit Buffer Unavailable (ETH\_DMASR register[2]) bits are set. In both cases, the position in the transmit list is retained. The retained position is that of the descriptor following the last descriptor closed by the DMA. The driver must explicitly issue a Transmit Poll Demand command after rectifying the suspension cause.

### Normal Tx DMA descriptors

The normal transmit descriptor structure consists of four 32-bit words as shown in [Figure 515](#). The bit descriptions of TDES0, TDES1, TDES2 and TDES3 are given below.

Note that enhanced descriptors must be used if time stamping is activated (ETH\_PTPTSCR bit 0, TSE=1) or if IPv4 checksum offload is activated (ETH\_MACCR bit 10, IPCO=1).

**Figure 515. Normal transmit descriptor**



- **TDES0: Transmit descriptor Word0**

The application software has to program the control bits [30:26]+[23:20] plus the OWN bit [31] during descriptor initialization. When the DMA updates the descriptor (or writes it back), it resets all the control bits plus the OWN bit, and reports only the status bits.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OWN	IC	LS	FS	DC	DP	TTSE	Res.	CIC	TER	TCH	Res.	Res.	TTSS	IHE	
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
ES	JT	FF	IPE	LCA	NC	LCO	EC	VF	CC	CC	ED	UF	DB		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **OWN:** Own bit

When set, this bit indicates that the descriptor is owned by the DMA. When this bit is reset, it indicates that the descriptor is owned by the CPU. The DMA clears this bit either when it completes the frame transmission or when the buffers allocated in the descriptor are read completely. The ownership bit of the frame's first descriptor must be set after all subsequent descriptors belonging to the same frame have been set.

Bit 30 **IC:** Interrupt on completion

When set, this bit sets the Transmit Interrupt (ETH\_DMASR[0]) after the present frame has been transmitted.

Bit 29 **LS:** Last segment

When set, this bit indicates that the buffer contains the last segment of the frame.

Bit 28 **FS:** First segment

When set, this bit indicates that the buffer contains the first segment of a frame.

Bit 27 **DC:** Disable CRC

When this bit is set, the MAC does not append a cyclic redundancy check (CRC) to the end of the transmitted frame. This is valid only when the first segment (TDES0[28]) is set.

Bit 26 **DP:** Disable pad

When set, the MAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes, and the CRC field is added despite the state of the DC (TDES0[27]) bit. This is valid only when the first segment (TDES0[28]) is set.

Bit 25 **TTSE:** Transmit time stamp enable

When TTSE is set and when TSE is set (ETH\_PTPTSCR bit 0), IEEE1588 hardware time stamping is activated for the transmit frame described by the descriptor. This field is only valid when the First segment control bit (TDES0[28]) is set.

Bit 24 Reserved, must be kept at reset value.

Bits 23:22 **CIC:** Checksum insertion control

These bits control the checksum calculation and insertion. Bit encoding is as shown below:

00: Checksum Insertion disabled

01: Only IP header checksum calculation and insertion are enabled

10: IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware

11: IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware.

Bit 21 **TER:** Transmit end of ring

When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.

Bit 20 **TCH:** Second address chained

When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When TDES0[20] is set, TBS2 (TDES1[28:16]) is a “don’t care” value. TDES0[21] takes precedence over TDES0[20].

Bits 19:18 Reserved, must be kept at reset value.

Bit 17 **TTSS:** Transmit time stamp status

This field is used as a status bit to indicate that a time stamp was captured for the described transmit frame. When this bit is set, TDES2 and TDES3 have a time stamp value captured for the transmit frame. This field is only valid when the descriptor’s Last segment control bit (TDES0[29]) is set.

Note that when enhanced descriptors are enabled (EDFE=1 in ETH\_DMABMR), TTSS=1 indicates that TDES6 and TDES7 have the time stamp value.

Bit 16 **IHE:** IP header error

When set, this bit indicates that the MAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application and indicates an error status if there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet length/type field value for an IPv4 or IPv6 frame must match the IP header version received with the packet. For IPv4 frames, an error status is also indicated if the Header Length field has a value less than 0x5.

Bit 15 **ES:** Error summary

Indicates the logical OR of the following bits:

- TDES0[14]: Jabber timeout
- TDES0[13]: Frame flush
- TDES0[11]: Loss of carrier
- TDES0[10]: No carrier
- TDES0[9]: Late collision
- TDES0[8]: Excessive collision
- TDES0[2]: Excessive deferral
- TDES0[1]: Underflow error
- TDES0[16]: IP header error
- TDES0[12]: IP payload error

Bit 14 **JT:** Jabber timeout

When set, this bit indicates the MAC transmitter has experienced a jabber timeout. This bit is only set when the MAC configuration register’s JD bit is not set.

Bit 13 **FF:** Frame flushed

When set, this bit indicates that the DMA/MTL flushed the frame due to a software Flush command given by the CPU.

Bit 12 **IPE:** IP payload error

When set, this bit indicates that MAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload. The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP or ICMP packet bytes received from the application and issues an error status in case of a mismatch.

Bit 11 **LCA:** Loss of carrier

When set, this bit indicates that a loss of carrier occurred during frame transmission (that is, the MII\_CRS signal was inactive for one or more transmit clock periods during frame transmission). This is valid only for the frames transmitted without collision when the MAC operates in Half-duplex mode.

Bit 10 **NC:** No carrier

When set, this bit indicates that the Carrier Sense signal from the PHY was not asserted during transmission.

Bit 9 **LCO:** Late collision

When set, this bit indicates that frame transmission was aborted due to a collision occurring after the collision window (64 byte times, including preamble, in MII mode). This bit is not valid if the Underflow Error bit is set.

Bit 8 **EC:** Excessive collision

When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the RD (Disable retry) bit in the MAC Configuration register is set, this bit is set after the first collision, and the transmission of the frame is aborted.

Bit 7 **VF:** VLAN frame

When set, this bit indicates that the transmitted frame was a VLAN-type frame.

Bits 6:3 **CC:** Collision count

This 4-bit counter value indicates the number of collisions occurring before the frame was transmitted. The count is not valid when the Excessive collisions bit (TDES0[8]) is set.

Bit 2 **ED:** Excessive deferral

When set, this bit indicates that the transmission has ended because of excessive deferral of over 24 288 bit times if the Deferral check (DC) bit in the MAC Control register is set high.

Bit 1 **UF:** Underflow error

When set, this bit indicates that the MAC aborted the frame because data arrived late from the RAM memory. Underflow error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the Suspended state and sets both Transmit underflow (ETH\_DMASR[5]) and Transmit interrupt (ETH\_DMASR[0]).

Bit 0 **DB:** Deferred bit

When set, this bit indicates that the MAC defers before transmission because of the presence of the carrier. This bit is valid only in Half-duplex mode.

- **TDES1: Transmit descriptor Word1**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	TBS2													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	TBS1													
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31:29 Reserved, must be kept at reset value.

28:16 **TBS2:** Transmit buffer 2 size

These bits indicate the second data buffer size in bytes. This field is not valid if TDES0[20] is set.

15:13 Reserved, must be kept at reset value.

- 12:0 **TBS1: Transmit buffer 1 size**

These bits indicate the first data buffer byte size, in bytes. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor, depending on the value of TCH (TDES0[20]).

- **TDES2: Transmit descriptor Word2**

TDES2 contains the address pointer to the first buffer of the descriptor or it contains time stamp data.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TBAP1/TBAP/TTS1															
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
TBAP1/TBAP/TTS1															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **TBAP1: Transmit buffer 1 address pointer / Transmit frame time stamp low**

These bits have two different functions: they indicate to the DMA the location of data in memory, and after all data are transferred, the DMA can then use these bits to pass back time stamp data.

**TBAP:** When the software makes this descriptor available to the DMA (at the moment that the OWN bit is set to 1 in TDES0), these bits indicate the physical address of Buffer 1. There is no limitation on the buffer address alignment. See [Host data buffer alignment on page 1560](#) for further details on buffer address alignment.

**TTSL:** Before it clears the OWN bit in TDES0, the DMA updates this field with the 32 least significant bits of the time stamp captured for the corresponding transmit frame (overwriting the value for TBAP1). This field has the time stamp only if time stamping is activated for this frame (see TTSE, TDES0 bit 25) and if the Last segment control bit (LS) in the descriptor is set.

- **TDES3: Transmit descriptor Word3**

TDES3 contains the address pointer either to the second buffer of the descriptor or the next descriptor, or it contains time stamp data.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TBAP2/TBAP2/TTS2															
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
TBAP2/TBAP2/TTS2															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **TBAP2:** Transmit buffer 2 address pointer (Next descriptor address) / Transmit frame time stamp high

These bits have two different functions: they indicate to the DMA the location of data in memory, and after all data are transferred, the DMA can then use these bits to pass back time stamp data.

**TBAP2:** When the software makes this descriptor available to the DMA (at the moment when the OWN bit is set to 1 in TDES0), these bits indicate the physical address of Buffer 2 when a descriptor ring structure is used. If the Second address chained (TDES1 [20]) bit is set, this address contains the pointer to the physical memory where the next descriptor is present. The buffer address pointer must be aligned to the bus width only when TDES1 [20] is set. (LSBs are ignored internally.)

**TTSH:** Before it clears the OWN bit in TDES0, the DMA updates this field with the 32 most significant bits of the time stamp captured for the corresponding transmit frame (overwriting the value for TBAP2). This field has the time stamp only if activated for this frame (see TDES0 bit 25, TTSE) and if the Last segment control bit (LS) in the descriptor is set.

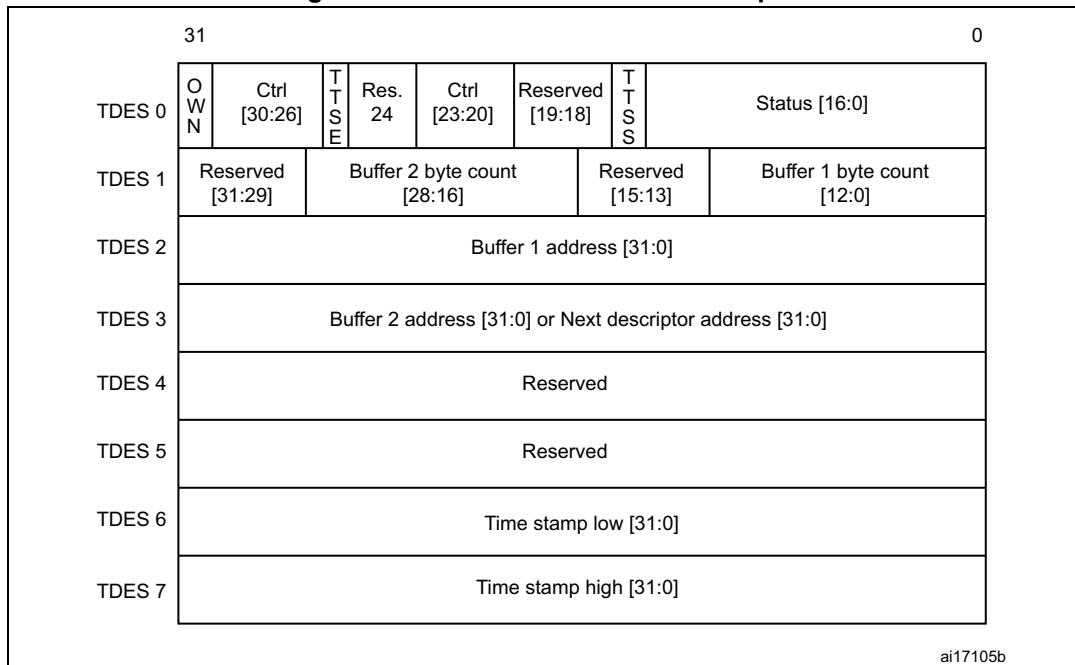
### Enhanced Tx DMA descriptors

Enhanced descriptors (enabled with EDFE=1, ETHDMABMR bit 7), must be used if time stamping is activated (TSE=1, ETH\_PTPTSCR bit 0) or if IPv4 checksum offload is activated (IPCO=1, ETH\_MACCR bit 10).

Enhanced descriptors comprise eight 32-bit words, twice the size of normal descriptors. TDES0, TDES1, TDES2 and TDES3 have the same definitions as for normal transmit descriptors (refer to [Normal Tx DMA descriptors](#)). TDES6 and TDES7 hold the time stamp. TDES4, TDES5, TDES6 and TDES7 are defined below.

When the Enhanced descriptor mode is selected, the software needs to allocate 32-bytes (8 DWORDS) of memory for every descriptor. When time stamping or IPv4 checksum offload are not being used, the enhanced descriptor format may be disabled and the software can use normal descriptors with the default size of 16 bytes.

Figure 516. Enhanced transmit descriptor



- **TDES4: Transmit descriptor Word4**  
Reserved
- **TDES5: Transmit descriptor Word5**  
Reserved
- **TDES6: Transmit descriptor Word6**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TTSL															
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
TTSH															
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

#### Bits 31:0 **TTSL: Transmit frame time stamp low**

This field is updated by DMA with the 32 least significant bits of the time stamp captured for the corresponding transmit frame. This field has the time stamp only if the Last segment control bit (LS) in the descriptor is set.

- **TDES7: Transmit descriptor Word7**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TTSH															
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
TTSH															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **TTS<sub>H</sub>**: Transmit frame time stamp high

This field is updated by DMA with the 32 most significant bits of the time stamp captured for the corresponding transmit frame. This field has the time stamp only if the Last segment control bit (LS) in the descriptor is set.

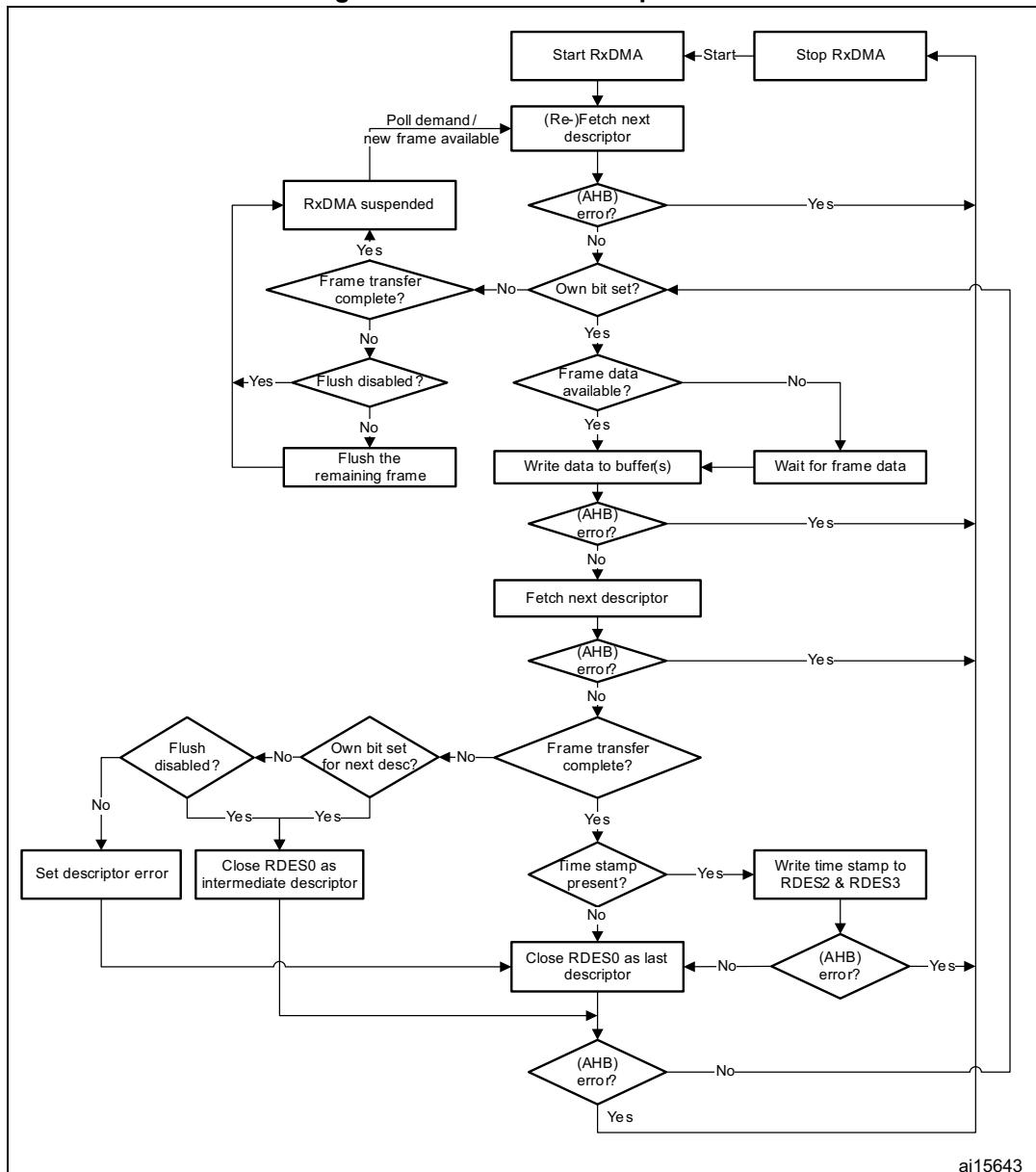
### 38.6.8 Rx DMA configuration

The Receive DMA engine's reception sequence is illustrated in [Figure 517](#) and described below:

1. The CPU sets up Receive descriptors (RDES0-RDES3) and sets the OWN bit (RDES0[31]).
2. Once the SR (ETH\_DMAOMR register[1]) bit is set, the DMA enters the Run state. While in the Run state, the DMA polls the receive descriptor list, attempting to acquire free descriptors. If the fetched descriptor is not free (is owned by the CPU), the DMA enters the Suspend state and jumps to Step 9.
3. The DMA decodes the receive data buffer address from the acquired descriptors.
4. Incoming frames are processed and placed in the acquired descriptor's data buffers.
5. When the buffer is full or the frame transfer is complete, the Receive engine fetches the next descriptor.
6. If the current frame transfer is complete, the DMA proceeds to step 7. If the DMA does not own the next fetched descriptor and the frame transfer is not complete (EOF is not yet transferred), the DMA sets the Descriptor error bit in RDES0 (unless flushing is disabled). The DMA closes the current descriptor (clears the OWN bit) and marks it as intermediate by clearing the Last segment (LS) bit in the RDES1 value (marks it as last descriptor if flushing is not disabled), then proceeds to step 8. If the DMA owns the next descriptor but the current frame transfer is not complete, the DMA closes the current descriptor as intermediate and returns to step 4.
7. If IEEE 1588 time stamping is enabled, the DMA writes the time stamp (if available) to the current descriptor's RDES2 and RDES3. It then takes the received frame's status and writes the status word to the current descriptor's RDES0, with the OWN bit cleared and the Last segment bit set.
8. The Receive engine checks the latest descriptor's OWN bit. If the CPU owns the descriptor (OWN bit is at 0) the Receive buffer unavailable bit (in ETH\_DMASR register[7]) is set and the DMA Receive engine enters the Suspended state (step 9). If the DMA owns the descriptor, the engine returns to step 4 and awaits the next frame.
9. Before the Receive engine enters the Suspend state, partial frames are flushed from the Receive FIFO (you can control flushing using bit 24 in the ETH\_DMAOMR register).
10. The Receive DMA exits the Suspend state when a Receive Poll demand is given or the start of next frame is available from the Receive FIFO. The engine proceeds to step 2 and re-fetches the next descriptor.

The DMA does not acknowledge accepting the status until it has completed the time stamp write-back and is ready to perform status write-back to the descriptor. If software has enabled time stamping through CSR, when a valid time stamp value is not available for the frame (for example, because the receive FIFO was full before the time stamp could be written to it), the DMA writes all ones to RDES2 and RDES3. Otherwise (that is, if time stamping is not enabled), RDES2 and RDES3 remain unchanged.

Figure 517. Receive DMA operation



ai15643

### Receive descriptor acquisition

The receive engine always attempts to acquire an extra descriptor in anticipation of an incoming frame. Descriptor acquisition is attempted if any of the following conditions is/are satisfied:

- The receive Start/Stop bit (ETH\_DMAOMR register[1]) has been set immediately after the DMA has been placed in the Run state.
- The data buffer of the current descriptor is full before the end of the frame currently being transferred
- The controller has completed frame reception, but the current receive descriptor has not yet been closed.
- The receive process has been suspended because of a CPU-owned buffer (RDES0[31] = 0) and a new frame is received.
- A Receive poll demand has been issued.

### Receive frame processing

The MAC transfers the received frames to the STM32F75xxx and STM32F74xxx memory only when the frame passes the address filter and the frame size is greater than or equal to the configurable threshold bytes set for the Receive FIFO, or when the complete frame is written to the FIFO in Store-and-forward mode. If the frame fails the address filtering, it is dropped in the MAC block itself (unless Receive All ETH\_MACFFR [31] bit is set). Frames that are shorter than 64 bytes, because of collision or premature termination, can be purged from the Receive FIFO. After 64 (configurable threshold) bytes have been received, the DMA block begins transferring the frame data to the receive buffer pointed to by the current descriptor. The DMA sets the first descriptor (RDES0[9]) after the DMA AHB Interface becomes ready to receive a data transfer (if DMA is not fetching transmit data from the memory), to delimit the frame. The descriptors are released when the OWN (RDES0[31]) bit is reset to 0, either as the data buffer fills up or as the last segment of the frame is transferred to the receive buffer. If the frame is contained in a single descriptor, both the last descriptor (RDES0[8]) and first descriptor (RDES0[9]) bits are set. The DMA fetches the next descriptor, sets the last descriptor (RDES0[8]) bit, and releases the RDES0 status bits in the previous frame descriptor. Then the DMA sets the receive interrupt bit (ETH\_DMASR register [6]). The same process repeats unless the DMA encounters a descriptor flagged as being owned by the CPU. If this occurs, the receive process sets the receive buffer unavailable bit (ETH\_DMASR register[7]) and then enters the Suspend state. The position in the receive list is retained.

### Receive process suspended

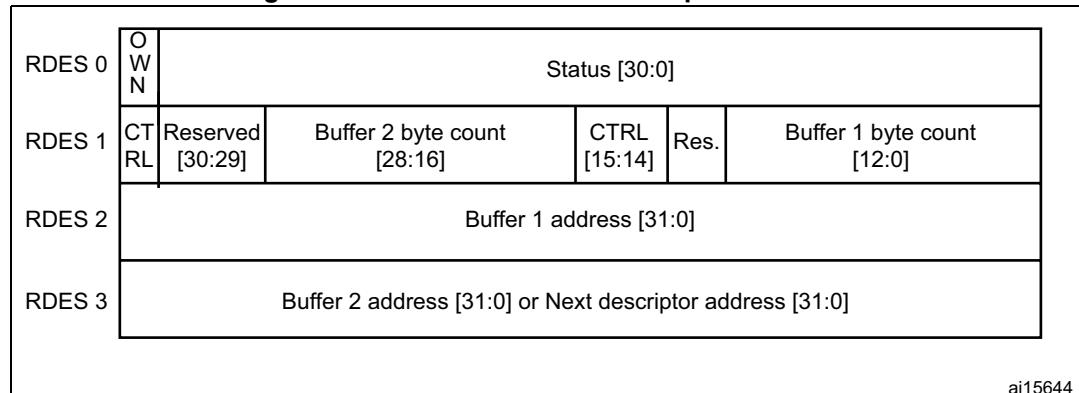
If a new receive frame arrives while the receive process is in Suspend state, the DMA re-fetches the current descriptor in the STM32F75xxx and STM32F74xxx memory. If the descriptor is now owned by the DMA, the receive process re-enters the Run state and starts frame reception. If the descriptor is still owned by the host, by default, the DMA discards the current frame at the top of the Rx FIFO and increments the missed frame counter. If more than one frame is stored in the Rx FIFO, the process repeats. The discarding or flushing of the frame at the top of the Rx FIFO can be avoided by setting the DMA Operation mode register bit 24 (DFRF). In such conditions, the receive process sets the receive buffer unavailable status bit and returns to the Suspend state.

### Normal Rx DMA descriptors

The normal receive descriptor structure consists of four 32-bit words (16 bytes). These are shown in [Figure 518](#). The bit descriptions of RDES0, RDES1, RDES2 and RDES3 are given below.

Note that enhanced descriptors must be used if time stamping is activated (TSE=1, ETH\_PTPTSCR bit 0) or if IPv4 checksum offload is activated (IPCO=1, ETH\_MACCR bit 10).

**Figure 518. Normal Rx DMA descriptor structure**



- RDES0: Receive descriptor Word0**

RDES0 contains the received frame status, the frame length and the descriptor ownership information.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OWN	AFM	FL													
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
ES	DE	SAF	LE	OE	VLAN	FS	LS	IPHCE/TSV	LCO	FT	RWT	RE	DE	CE	PCE/ESA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

**Bit 31 OWN: Own bit**

When set, this bit indicates that the descriptor is owned by the DMA of the MAC Subsystem. When this bit is reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.

**Bit 30 AFM: Destination address filter fail**

When set, this bit indicates a frame that failed the DA filter in the MAC Core.

**Bits 29:16 FL: Frame length**

These bits indicate the byte length of the received frame that was transferred to host memory (including CRC). This field is valid only when last descriptor (RDES0[8]) is set and descriptor error (RDES0[14]) is reset.

This field is valid when last descriptor (RDES0[8]) is set. When the last descriptor and error summary bits are not set, this field indicates the accumulated number of bytes that have been transferred for the current frame.

**Bit 15 ES:** Error summary

Indicates the logical OR of the following bits:

RDES0[1]: CRC error

RDES0[3]: Receive error

RDES0[4]: Watchdog timeout

RDES0[6]: Late collision

RDES0[7]: Giant frame (This is not applicable when RDES0[7] indicates an IPv4 header checksum error.)

RDES0[11]: Overflow error

RDES0[14]: Descriptor error.

This field is valid only when the last descriptor (RDES0[8]) is set.

**Bit 14 DE:** Descriptor error

When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the next descriptor. The frame is truncated. This field is valid only when the last descriptor (RDES0[8]) is set.

**Bit 13 SAF:** Source address filter fail

When set, this bit indicates that the SA field of frame failed the SA filter in the MAC Core.

**Bit 12 LE:** Length error

When set, this bit indicates that the actual length of the received frame does not match the value in the Length/ Type field. This bit is valid only when the Frame type (RDES0[5]) bit is reset.

**Bit 11 OE:** Overflow error

When set, this bit indicates that the received frame was damaged due to buffer overflow.

**Bit 10 VLAN:** VLAN tag

When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the MAC core.

**Bit 9 FS:** First descriptor

When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next descriptor contains the beginning of the frame.

**Bit 8 LS:** Last descriptor

When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame.

**Bit 7 IPHCE/TSV:** IPv header checksum error / time stamp valid

If IPHCE is set, it indicates an error in the IPv4 or IPv6 header. This error can be due to inconsistent Ethernet Type field and IP header Version field values, a header checksum mismatch in IPv4, or an Ethernet frame lacking the expected number of IP header bytes. This bit can take on special meaning as specified in [Table 285](#).

If enhanced descriptor format is enabled (EDFE=1, bit 7 of ETH\_DMABMR), this bit takes on the TSV function (otherwise it is IPHCE). When TSV is set, it indicates that a snapshot of the timestamp is written in descriptor words 6 (RDES6) and 7 (RDES7). TSV is valid only when the Last descriptor bit (RDES0[8]) is set.

**Bit 6 LCO:** Late collision

When set, this bit indicates that a late collision has occurred while receiving the frame in Half-duplex mode.

**Bit 5 FT:** Frame type

When set, this bit indicates that the Receive frame is an Ethernet-type frame (the LT field is greater than or equal to 0x0600). When this bit is reset, it indicates that the received frame is an IEEE802.3 frame. This bit is not valid for Runt frames less than 14 bytes. When the normal descriptor format is used (ETH\_DMABMR EDFE=0), FT can take on special meaning as specified in [Table 285](#).

**Bit 4 RWT:** Receive watchdog timeout

When set, this bit indicates that the Receive watchdog timer has expired while receiving the current frame and the current frame is truncated after the watchdog timeout.

**Bit 3 RE:** Receive error

When set, this bit indicates that the RX\_ERR signal is asserted while RX\_DV is asserted during frame reception.

**Bit 2 DE:** Dribble bit error

When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles). This bit is valid only in MII mode.

**Bit 1 CE:** CRC error

When set, this bit indicates that a cyclic redundancy check (CRC) error occurred on the received frame. This field is valid only when the last descriptor (RDES0[8]) is set.

**Bit 0 PCE/ESA:** Payload checksum error / extended status available

When set, it indicates that the TCP, UDP or ICMP checksum the core calculated does not match the received encapsulated TCP, UDP or ICMP segment's Checksum field. This bit is also set when the received number of payload bytes does not match the value indicated in the Length field of the encapsulated IPv4 or IPv6 datagram in the received Ethernet frame. This bit can take on special meaning as specified in [Table 285](#).

If the enhanced descriptor format is enabled (EDFE=1, bit 7 in ETH\_DMABMR), this bit takes on the ESA function (otherwise it is PCE). When ESA is set, it indicates that the extended status is available in descriptor word 4 (RDES4). ESA is valid only when the last descriptor bit (RDES0[8]) is set.

Bits 5, 7, and 0 reflect the conditions discussed in [Table 285](#).

**Table 285. Receive descriptor 0 - encoding for bits 7, 5 and 0 (normal descriptor format only, EDFE=0)**

Bit 5: frame type	Bit 7: IPC checksum error	Bit 0: payload checksum error	Frame status
0	0	0	IEEE 802.3 Type frame (Length field value is less than 0x0600.)
1	0	0	IPv4/IPv6 Type frame, no checksum error detected
1	0	1	IPv4/IPv6 Type frame with a payload checksum error (as described for PCE) detected
1	1	0	IPv4/IPv6 Type frame with an IP header checksum error (as described for IPC CE) detected
1	1	1	IPv4/IPv6 Type frame with both IP header and payload checksum errors detected
0	0	1	IPv4/IPv6 Type frame with no IP header checksum error and the payload check bypassed, due to an unsupported payload
0	1	1	A Type frame that is neither IPv4 or IPv6 (the checksum offload engine bypasses checksum completely.)
0	1	0	Reserved

- **RDES1: Receive descriptor Word1**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
DIC	Res.	Res.	RBS2														
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		
RER	RCH	Res.	RBS														
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 31 **DIC:** Disable interrupt on completion

When set, this bit prevents setting the Status register's RS bit (CSR5[6]) for the received frame ending in the buffer indicated by this descriptor. This, in turn, disables the assertion of the interrupt to Host due to RS for that frame.

Bits 30:29 Reserved, must be kept at reset value.

Bits 28:16 **RBS2:** Receive buffer 2 size

These bits indicate the second data buffer size, in bytes. The buffer size must be a multiple of 4, 8, or 16, depending on the bus widths (32, 64 or 128, respectively), even if the value of RDES3 (buffer2 address pointer) is not aligned to bus width. If the buffer size is not an appropriate multiple of 4, 8 or 16, the resulting behavior is undefined. This field is not valid if RDES1 [14] is set.

Bit 15 **RER:** Receive end of ring

When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.

**Bit 14 RCH:** Second address chained

When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a “don’t care” value. RDES1[15] takes precedence over RDES1[14].

**Bit 13 Reserved, must be kept at reset value.**

**Bits 12:0 RBS1:** Receive buffer 1 size

Indicates the first data buffer size in bytes. The buffer size must be a multiple of 4, 8 or 16, depending upon the bus widths (32, 64 or 128), even if the value of RDES2 (buffer1 address pointer) is not aligned. When the buffer size is not a multiple of 4, 8 or 16, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or next descriptor depending on the value of RCH (bit 14).

- RDES2: Receive descriptor Word2**

RDES2 contains the address pointer to the first data buffer in the descriptor, or it contains time stamp data.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RBP1 / RTSL															
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
RBP1 / RTSL															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

**Bits 31:0 RBAP1 / RTSL:** Receive buffer 1 address pointer / Receive frame time stamp low

These bits take on two different functions: the application uses them to indicate to the DMA where to store the data in memory, and then after transferring all the data the DMA may use these bits to pass back time stamp data.

**RBAP1:** When the software makes this descriptor available to the DMA (at the moment that the OWN bit is set to 1 in RDES0), these bits indicate the physical address of Buffer 1. There are no limitations on the buffer address alignment except for the following condition: the DMA uses the configured value for its address generation when the RDES2 value is used to store the start of frame. Note that the DMA performs a write operation with the RDES2[3/2/1:0] bits as 0 during the transfer of the start of frame but the frame data is shifted as per the actual Buffer address pointer. The DMA ignores RDES2[3/2/1:0] (corresponding to bus width of 128/64/32) if the address pointer is to a buffer where the middle or last part of the frame is stored.

**RTSL:** Before it clears the OWN bit in RDES0, the DMA updates this field with the 32 least significant bits of the time stamp captured for the corresponding receive frame (overwriting the value for RBAP1). This field has the time stamp only if time stamping is activated for this frame and if the Last segment control bit (LS) in the descriptor is set.

- RDES3: Receive descriptor Word3**

RDES3 contains the address pointer either to the second data buffer in the descriptor or to the next descriptor, or it contains time stamp data.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RBP2 / RTSH															
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
RBP2 / RTSH															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **RBAP2 / RTSH:** Receive buffer 2 address pointer (next descriptor address) / Receive frame time stamp high

These bits take on two different functions: the application uses them to indicate to the DMA the location of where to store the data in memory, and then after transferring all the data the DMA may use these bits to pass back time stamp data.

**RBAP1:** When the software makes this descriptor available to the DMA (at the moment that the OWN bit is set to 1 in RDES0), these bits indicate the physical address of buffer 2 when a descriptor ring structure is used. If the second address chained (RDES1[24]) bit is set, this address contains the pointer to the physical memory where the next descriptor is present. If RDES1[24] is set, the buffer (next descriptor) address pointer must be bus width-aligned (RDES3[3, 2, or 1:0] = 0, corresponding to a bus width of 128, 64 or 32. LSBs are ignored internally.)

However, when RDES1[24] is reset, there are no limitations on the RDES3 value, except for the following condition: the DMA uses the configured value for its buffer address generation when the RDES3 value is used to store the start of frame. The DMA ignores RDES3[3, 2, or 1:0] (corresponding to a bus width of 128, 64 or 32) if the address pointer is to a buffer where the middle or last part of the frame is stored.

**RTSH:** Before it clears the OWN bit in RDES0, the DMA updates this field with the 32 most significant bits of the time stamp captured for the corresponding receive frame (overwriting the value for RBAP2). This field has the time stamp only if time stamping is activated and if the Last segment control bit (LS) in the descriptor is set.

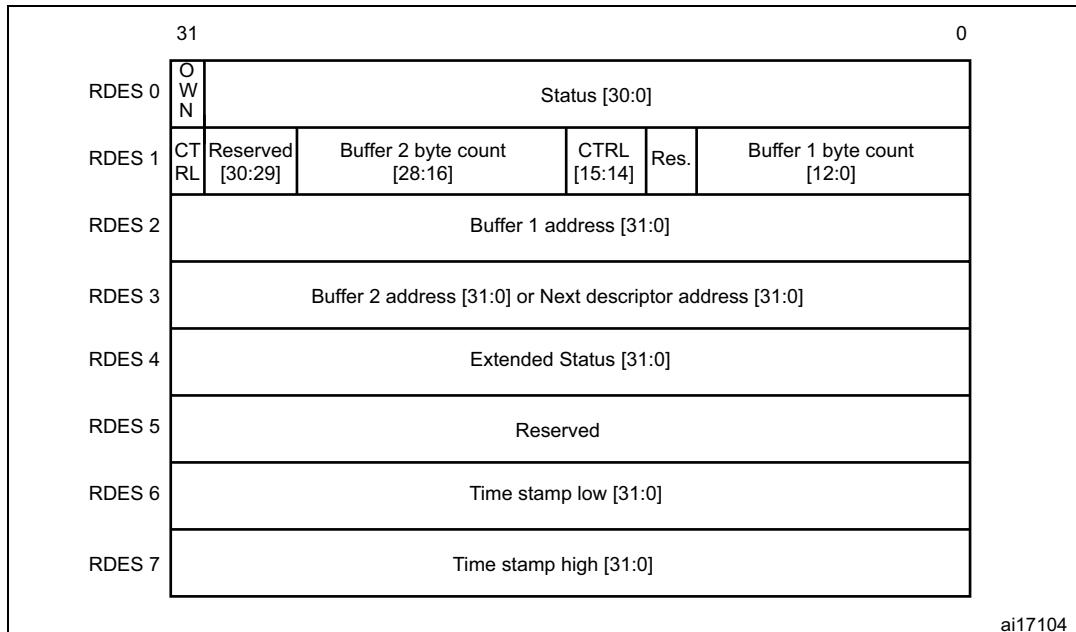
### Enhanced Rx DMA descriptors format with IEEE1588 time stamp

Enhanced descriptors (enabled with EDFE=1, ETHDMABMR bit 7), must be used if time stamping is activated (TSE=1, ETH\_PTPTSCR bit 0) or if IPv4 checksum offload is activated (IPCO=1, ETH\_MACCR bit 10).

Enhanced descriptors comprise eight 32-bit words, twice the size of normal descriptors. RDES0, RDES1, RDES2 and RDES3 have the same definitions as for normal receive descriptors (refer to [Normal Rx DMA descriptors](#)). RDES4 contains extended status while RDES6 and RDES7 hold the time stamp. RDES4, RDES5, RDES6 and RDES7 are defined below.

When the Enhanced descriptor mode is selected, the software needs to allocate 32 bytes (8 DWORDS) of memory for every descriptor. When time stamping or IPv4 checksum offload are not being used, the enhanced descriptor format may be disabled and the software can use normal descriptors with the default size of 16 bytes.

**Figure 519. Enhanced receive descriptor field format with IEEE1588 time stamp enabled**



- RDES4: Receive descriptor Word4

The extended status, shown below, is valid only when there is status related to IPv4 checksum or time stamp available as indicated by bit 0 in RDES0.

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.	Res.	PV	PFT	PMT				IPV6PR	IPV4PR	IPCB	IPPE	IPHE	IPPT		
		rw	rw	rw	rw	rw	rw	rw	rw						

Bits 31:14 Reserved, must be kept at reset value.

#### Bit 13 PV: PTP version

When set, indicates that the received PTP message uses the IEEE 1588 version 2 format. When cleared, it uses version 1 format. This is valid only if the message type is non-zero.

#### Bit 12 PFT: PTP frame type

When set, this bit indicates that the PTP message is sent directly over Ethernet. When this bit is cleared and the message type is non-zero, it indicates that the PTP message is sent over UDP-IPv4 or UDP-IPv6. The information on IPv4 or IPv6 can be obtained from bits 6 and 7.

Bits 11:8 **PMT:** PTP message type

These bits are encoded to give the type of the message received.

- 0000: No PTP message received
- 0001: SYNC (all clock types)
- 0010: Follow\_Up (all clock types)
- 0011: Delay\_Req (all clock types)
- 0100: Delay\_Resp (all clock types)
- 0101: Pdelay\_Req (in peer-to-peer transparent clock) or Announce (in ordinary or boundary clock)
- 0110: Pdelay\_Resp (in peer-to-peer transparent clock) or Management (in ordinary or boundary clock)
- 0111: Pdelay\_Resp\_Follow\_Up (in peer-to-peer transparent clock) or Signaling (for ordinary or boundary clock)
- 1xxx - Reserved

Bit 7 **IPV6PR:** IPv6 packet received

When set, this bit indicates that the received packet is an IPv6 packet.

Bit 6 **IPV4PR:** IPv4 packet received

When set, this bit indicates that the received packet is an IPv4 packet.

Bit 5 **IPCB:** IP checksum bypassed

When set, this bit indicates that the checksum offload engine is bypassed.

Bit 4 **IPPE:** IP payload error

When set, this bit indicates that the 16-bit IP payload checksum (that is, the TCP, UDP, or ICMP checksum) that the core calculated does not match the corresponding checksum field in the received segment. It is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP Header field.

Bit 3 **IPHE:** IP header error

When set, this bit indicates either that the 16-bit IPv4 header checksum calculated by the core does not match the received checksum bytes, or that the IP datagram version is not consistent with the Ethernet Type value.

Bits 2:0 **IPPT:** IP payload type

If IPv4 checksum offload is activated (IPCO=1, ETH\_MACCR bit 10), these bits indicate the type of payload encapsulated in the IP datagram. These bits are '00' if there is an IP header error or fragmented IP.

- 000: Unknown or did not process IP payload
- 001: UDP
- 010: TCP
- 011: ICMP
- 1xx: Reserved

- **RDES5: Receive descriptor Word5**

Reserved.

- **RDES6: Receive descriptor Word6**

The table below describes the fields that have different meaning for RDES6 when the receive descriptor is closed and time stamping is enabled.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RTSL															
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
RTSL															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Bits 31:0 **RTSL:** Receive frame time stamp low

The DMA updates this field with the 32 least significant bits of the time stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame indicated by last descriptor status bit (RDES0[8]). When this field and the RTSH field in RDES7 show all ones, the time stamp must be treated as corrupt.

- **RDES7: Receive descriptor Word7**

The table below describes the fields that have a different meaning for RDES7 when the receive descriptor is closed and time stamping is enabled.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RTSH															
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
RTSH															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Bits 31:0 **RTSH:** Receive frame time stamp high

The DMA updates this field with the 32 most significant bits of the time stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame indicated by last descriptor status bit (RDES0[8]).

When this field and RDES7's RTSL field show all ones, the time stamp must be treated as corrupt.

### 38.6.9 DMA interrupts

Interrupts can be generated as a result of various events. The ETH\_DMASR register contains all the bits that might cause an interrupt. The ETH\_DMAIER register contains an enable bit for each of the events that can cause an interrupt.

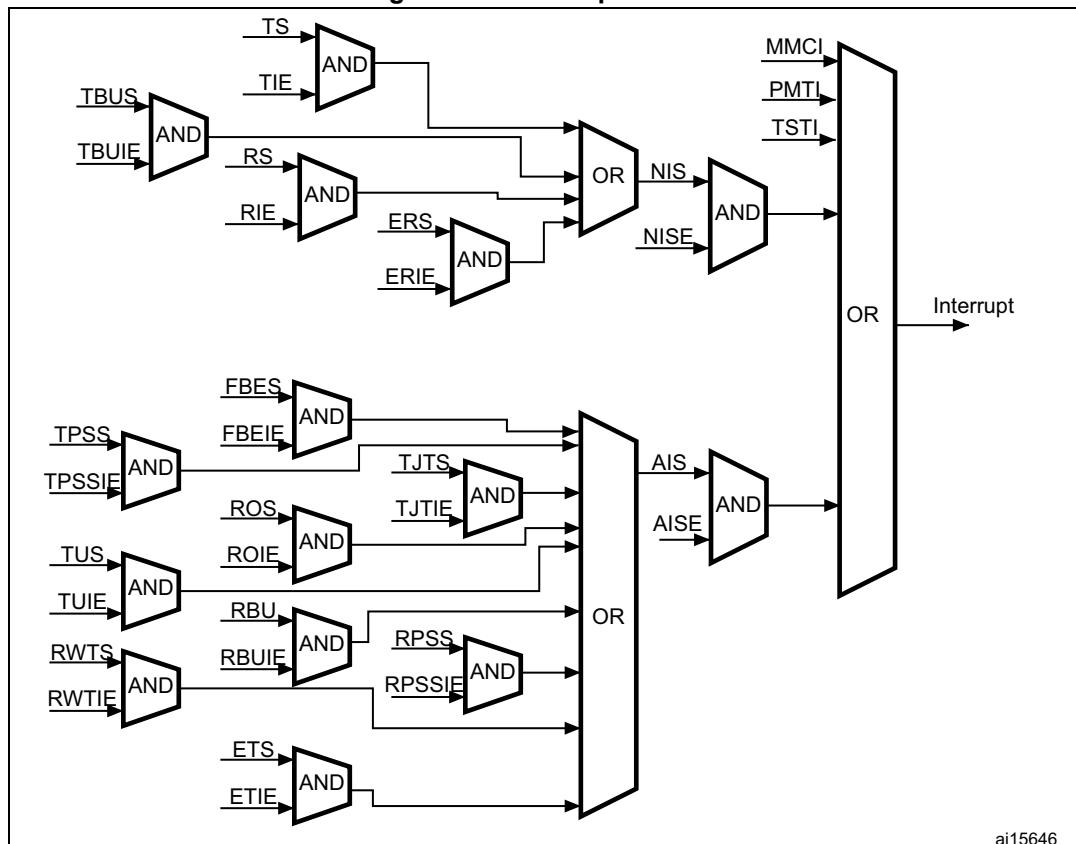
There are two groups of interrupts, Normal and Abnormal, as described in the ETH\_DMASR register. Interrupts are cleared by writing a 1 to the corresponding bit position. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared. If the MAC core is the cause for assertion of the interrupt, then any of the TSTS or PMTS bits in the ETH\_DMASR register is set high.

Interrupts are not queued and if the interrupt event occurs before the driver has responded to it, no additional interrupts are generated. For example, the Receive Interrupt bit (ETH\_DMASR register [6]) indicates that one or more frames were transferred to the STM32F75xxx and STM32F74xxx buffer. The driver must scan all descriptors, from the last recorded position to the first one owned by the DMA.

An interrupt is generated only once for simultaneous, multiple events. The driver must scan the ETH\_DMASR register for the cause of the interrupt. The interrupt is not generated again unless a new interrupting event occurs, after the driver has cleared the appropriate bit in the

ETH\_DMASR register. For example, the controller generates a Receive interrupt (ETH\_DMASR register[6]) and the driver begins reading the ETH\_DMASR register. Next, receive buffer unavailable (ETH\_DMASR register[7]) occurs. The driver clears the Receive interrupt. Even then, a new interrupt is generated, due to the active or pending Receive buffer unavailable interrupt.

Figure 520. Interrupt scheme



## 38.7 Ethernet interrupts

The Ethernet controller has two interrupt vectors: one dedicated to normal Ethernet operations and the other, used only for the Ethernet wakeup event (with wakeup frame or Magic Packet detection) when it is mapped on EXTI Line19.

The first Ethernet vector is reserved for interrupts generated by the MAC and the DMA as listed in the [MAC interrupts](#) and [DMA interrupts](#) sections.

The second vector is reserved for interrupts generated by the PMT on wakeup events. The mapping of a wakeup event on EXTI line19 causes the STM32F75xxx and STM32F74xxx to exit the low-power mode, and generates an interrupt.

When an Ethernet wakeup event mapped on EXTI Line19 occurs and the MAC PMT interrupt is enabled and the EXTI Line19 interrupt, with detection on rising edge, is also enabled, both interrupts are generated.

A watchdog timer (see ETH\_DMARSWTR register) is given for flexible control of the RS bit (ETH\_DMASR register). When this watchdog timer is programmed with a non-zero value, it

gets activated as soon as the RxDMA completes a transfer of a received frame to system memory without asserting the Receive Status because it is not enabled in the corresponding Receive descriptor (RDES1[31]). When this timer runs out as per the programmed value, the RS bit is set and the interrupt is asserted if the corresponding RIE is enabled in the ETH\_DMAIER register. This timer is disabled before it runs out, when a frame is transferred to memory and the RS is set because it is enabled for that descriptor.

**Note:** *Reading the PMT control and status register automatically clears the Wakeup Frame Received and Magic Packet Received PMT interrupt flags. However, since the registers for these flags are in the CLK\_RX domain, there may be a significant delay before this update is visible by the firmware. The delay is especially long when the RX clock is slow (in 10 Mbit mode) and when the AHB bus is high-frequency.*

*Since interrupt requests from the PMT to the CPU are based on the same registers in the CLK\_RX domain, the CPU may spuriously call the interrupt routine a second time even after reading PMT\_CSR. Thus, it may be necessary that the firmware polls the Wakeup Frame Received and Magic Packet Received bits and exits the interrupt service routine only when they are found to be at '0'.*

## 38.8 Ethernet registers

The peripheral registers can be accessed by bytes (8-bit), half-words (16-bit) or words (32-bit).

### 38.8.1 MAC register description

#### Ethernet MAC configuration register (ETH\_MACCR)

Address offset: 0x0000

Reset value: 0x0000 8000

The MAC configuration register is the operation mode register of the MAC. It establishes receive and transmit operating modes.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	CSTF	Res.	WD	JD	Res.	Res.	IFG[2:0]			CSD
						rw		rw	rw			rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	FES	ROD	LM	DM	IPCO	RD	Res.	APCS	BL[1:0]		DC	TE	RE	Res.	Res.
	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw		

Bits 31:26 Reserved, must be kept at reset value.

Bit 25 **CSTF:** CRC stripping for Type frames

When set, the last 4 bytes (FCS) of all frames of Ether type (type field greater than 0x0600) will be stripped and dropped before forwarding the frame to the application.

Bit 24 Reserved, must be kept at reset value.

Bit 23 **WD:** Watchdog disable

When this bit is set, the MAC disables the watchdog timer on the receiver, and can receive frames of up to 16 384 bytes.

When this bit is reset, the MAC allows no more than 2 048 bytes of the frame being received and cuts off any bytes received after that.

Bit 22 **JD:** Jabber disable

When this bit is set, the MAC disables the jabber timer on the transmitter, and can transfer frames of up to 16 384 bytes.

When this bit is reset, the MAC cuts off the transmitter if the application sends out more than 2 048 bytes of data during transmission.

Bits 21:20 Reserved, must be kept at reset value.

Bits 19:17 **IFG[2:0]:** Interframe gap

These bits control the minimum interframe gap between frames during transmission.

000: 96 bit times

001: 88 bit times

010: 80 bit times

....

111: 40 bit times

*Note: In Half-duplex mode, the minimum IFG can be configured for 64 bit times (IFG[2:0] = 100) only. Lower values are not considered.*

Bit 16 **CSD:** Carrier sense disable

When set high, this bit makes the MAC transmitter ignore the MII CRS signal during frame transmission in Half-duplex mode. No error is generated due to Loss of Carrier or No Carrier during such transmission.

When this bit is low, the MAC transmitter generates such errors due to Carrier Sense and even aborts the transmissions.

Bit 15 Reserved, must be kept at reset value.

Bit 14 **FES:** Fast Ethernet speed

Indicates the speed in Fast Ethernet (MII) mode:

0: 10 Mbit/s

1: 100 Mbit/s

Bit 13 **ROD:** Receive own disable

When this bit is set, the MAC disables the reception of frames in Half-duplex mode.

When this bit is reset, the MAC receives all packets that are given by the PHY while transmitting.

This bit is not applicable if the MAC is operating in Full-duplex mode.

Bit 12 **LM:** Loopback mode

When this bit is set, the MAC operates in loopback mode at the MII. The MII receive clock input (RX\_CLK) is required for the loopback to work properly, as the transmit clock is not looped-back internally.

Bit 11 **DM:** Duplex mode

When this bit is set, the MAC operates in a Full-duplex mode where it can transmit and receive simultaneously.

Bit 10 **IPCO:** IPv4 checksum offload

When set, this bit enables IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. When this bit is reset, the checksum offload function in the receiver is disabled and the corresponding PCE and IP HCE status bits (see [Table 282](#)) are always cleared.

Bit 9 **RD:** Retry disable

When this bit is set, the MAC attempts only 1 transmission. When a collision occurs on the MII, the MAC ignores the current frame transmission and reports a Frame Abort with excessive collision error in the transmit frame status.

When this bit is reset, the MAC attempts retries based on the settings of BL.

*Note: This bit is applicable only in the Half-duplex mode.*

## Bit 8 Reserved, must be kept at reset value.

Bit 7 **APCS:** Automatic pad/CRC stripping

When this bit is set, the MAC strips the Pad/FCS field on incoming frames only if the length's field value is less than or equal to 1 500 bytes. All received frames with length field greater than or equal to 1 501 bytes are passed on to the application without stripping the Pad/FCS field.

When this bit is reset, the MAC passes all incoming frames unmodified.

Bits 6:5 **BL[1:0]:** Back-off limit

The Back-off limit determines the random integer number ( $r$ ) of slot time delays (4 096 bit times for 1000 Mbit/s and 512 bit times for 10/100 Mbit/s) the MAC waits before rescheduling a transmission attempt during retries after a collision.

*Note: This bit is applicable only to Half-duplex mode.*

00:  $k = \min(n, 10)$

01:  $k = \min(n, 8)$

10:  $k = \min(n, 4)$

11:  $k = \min(n, 1)$ ,

where  $n$  = retransmission attempt. The random integer  $r$  takes the value in the range  $0 \leq r \leq 2^k$

Bit 4 **DC:** Deferral check

When this bit is set, the deferral check function is enabled in the MAC. The MAC issues a Frame Abort status, along with the excessive deferral error bit set in the transmit frame status when the transmit state machine is deferred for more than 24 288 bit times in 10/100-Mbit/s mode. Deferral begins when the transmitter is ready to transmit, but is prevented because of an active CRS (carrier sense) signal on the MII. Defer time is not cumulative. If the transmitter defers for 10 000 bit times, then transmits, collides, backs off, and then has to defer again after completion of back-off, the deferral timer resets to 0 and restarts.

When this bit is reset, the deferral check function is disabled and the MAC defers until the CRS signal goes inactive. This bit is applicable only in Half-duplex mode.

Bit 3 **TE:** Transmitter enable

When this bit is set, the transmit state machine of the MAC is enabled for transmission on the MII. When this bit is reset, the MAC transmit state machine is disabled after the completion of the transmission of the current frame, and does not transmit any further frames.

Bit 2 **RE:** Receiver enable

When this bit is set, the receiver state machine of the MAC is enabled for receiving frames from the MII. When this bit is reset, the MAC receive state machine is disabled after the completion of the reception of the current frame, and will not receive any further frames from the MII.

## Bits 1:0 Reserved, must be kept at reset value.

### Ethernet MAC frame filter register (ETH\_MACFFR)

Address offset: 0x0004

Reset value: 0x0000 0000

The MAC frame filter register contains the filter controls for receiving frames. Some of the controls from this register go to the address check block of the MAC, which performs the first level of address filtering. The second level of filtering is performed on the incoming frame, based on other controls such as pass bad frames and pass control frames.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RA	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	HPF	SAF	SAIF	PCF[1:0]		BFD	PAM	DAIF	HM	HU	PM
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Bit 31 **RA:** Receive all

When this bit is set, the MAC receiver passes all received frames on to the application, irrespective of whether they have passed the address filter. The result of the SA/DA filtering is updated (pass or fail) in the corresponding bits in the receive status word. When this bit is reset, the MAC receiver passes on to the application only those frames that have passed the SA/DA address filter.

Bits 30:11 Reserved, must be kept at reset value.

#### Bit 10 **HPF:** Hash or perfect filter

When this bit is set and if the HM or HU bit is set, the address filter passes frames that match either the perfect filtering or the hash filtering.

When this bit is cleared and if the HU or HM bit is set, only frames that match the Hash filter are passed.

#### Bit 9 **SAF:** Source address filter

The MAC core compares the SA field of the received frames with the values programmed in the enabled SA registers. If the comparison matches, then the SAMatch bit in the RxStatus word is set high. When this bit is set high and the SA filter fails, the MAC drops the frame. When this bit is reset, the MAC core forwards the received frame to the application. It also forwards the updated SA Match bit in RxStatus depending on the SA address comparison.

#### Bit 8 **SAIF:** Source address inverse filtering

When this bit is set, the address check block operates in inverse filtering mode for the SA address comparison. The frames whose SA matches the SA registers are marked as failing the SA address filter.

When this bit is reset, frames whose SA does not match the SA registers are marked as failing the SA address filter.

**Bits 7:6 PCF[1:0]: Pass control frames**

These bits control the forwarding of all control frames (including unicast and multicast PAUSE frames). Note that the processing of PAUSE control frames depends only on RFCE in Flow Control Register[2].

00: MAC prevents all control frames from reaching the application

01: MAC forwards all control frames to application except Pause control frames

10: MAC forwards all control frames to application even if they fail the address filter

11: MAC forwards control frames that pass the address filter.

These bits control the forwarding of all control frames (including unicast and multicast PAUSE frames). Note that the processing of PAUSE control frames depends only on RFCE in Flow Control Register[2].

00 or 01: MAC prevents all control frames from reaching the application

10: MAC forwards all control frames to application even if they fail the address filter

11: MAC forwards control frames that pass the address filter.

**Bit 5 BFD: Broadcast frames disable**

When this bit is set, the address filters filter all incoming broadcast frames.

When this bit is reset, the address filters pass all received broadcast frames.

**Bit 4 PAM: Pass all multicast**

When set, this bit indicates that all received frames with a multicast destination address (first bit in the destination address field is '1') are passed.

When reset, filtering of multicast frame depends on the HM bit.

**Bit 3 DAIF: Destination address inverse filtering**

When this bit is set, the address check block operates in inverse filtering mode for the DA address comparison for both unicast and multicast frames.

When reset, normal filtering of frames is performed.

**Bit 2 HM: Hash multicast**

When set, MAC performs destination address filtering of received multicast frames according to the hash table.

When reset, the MAC performs a perfect destination address filtering for multicast frames, that is, it compares the DA field with the values programmed in DA registers.

**Bit 1 HU: Hash unicast**

When set, MAC performs destination address filtering of unicast frames according to the hash table.

When reset, the MAC performs a perfect destination address filtering for unicast frames, that is, it compares the DA field with the values programmed in DA registers.

**Bit 0 PM: Promiscuous mode**

When this bit is set, the address filters pass all incoming frames regardless of their destination or source address. The SA/DA filter fails status bits in the receive status word are always cleared when PM is set.

**Ethernet MAC hash table high register (ETH\_MACHTHR)**

Address offset: 0x0008

Reset value: 0x0000 0000

The 64-bit Hash table is used for group address filtering. For hash filtering, the contents of the destination address in the incoming frame are passed through the CRC logic, and the upper 6 bits in the CRC register are used to index the contents of the Hash table. This CRC

is a 32-bit value coded by the following polynomial (for more details refer to [Section 38.5.3: MAC frame reception](#)):

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The most significant bit determines the register to be used (hash table high/hash table low), and the other 5 bits determine which bit within the register. A hash value of 0b0 0000 selects bit 0 in the selected register, and a value of 0b1 1111 selects bit 31 in the selected register.

For example, if the DA of the incoming frame is received as 0x1F52 419C B6AF (0x1F is the first byte received on the MII interface), then the internally calculated 6-bit Hash value is 0x2C and the HTH register bit[12] is checked for filtering. If the DA of the incoming frame is received as 0xA00A 9800 0045, then the calculated 6-bit Hash value is 0x07 and the HTL register bit[7] is checked for filtering.

If the corresponding bit value in the register is 1, the frame is accepted. Otherwise, it is rejected. If the PAM (pass all multicast) bit is set in the ETH\_MACFFR register, then all multicast frames are accepted regardless of the multicast hash values.

The Hash table high register contains the higher 32 bits of the multicast Hash table.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HTH[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
HTH[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **HTH[31:0]**: Hash table high

This field contains the upper 32 bits of Hash table.

### Ethernet MAC hash table low register (ETH\_MACHTLR)

Address offset: 0x000C

Reset value: 0x0000 0000

The Hash table low register contains the lower 32 bits of the multi-cast Hash table.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HTL[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
HTL[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **HTL[31:0]**: Hash table low

This field contains the lower 32 bits of the Hash table.

### Ethernet MAC MII address register (ETH\_MACMIIAR)

Address offset: 0x0010

Reset value: 0x0000 0000

The MII address register controls the management cycles to the external PHY through the management interface.

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
PA[4:0]					MR[4:0]					Res.	CR[2:0]			MW	MB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rc_w1

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:11 **PA[4:0]:** PHY address

This field tells which of the 32 possible PHY devices are being accessed.

Bits 10:6 **MR[4:0]:** MII register

These bits select the desired MII register in the selected PHY device.

Bit 5 Reserved, must be kept at reset value.

Bits 4:2 **CR[2:0]:** Clock range

The CR clock range selection determines the HCLK frequency and is used to decide the frequency of the MDC clock:

Selection HCLK MDC Clock

000 60-100 MHz HCLK/42

001 100-150 MHz HCLK/62

010 20-35 MHz HCLK/16

011 35-60 MHz HCLK/26

100 150-216 MHz HCLK/102

101, 110, 111 Reserved

Bit 1 **MW:** MII write

When set, this bit tells the PHY that this will be a Write operation using the MII Data register. If this bit is not set, this will be a Read operation, placing the data in the MII Data register.

Bit 0 **MB:** MII busy

This bit should read a logic 0 before writing to ETH\_MACMIIAR and ETH\_MACMIIDR. This bit must also be reset to 0 during a Write to ETH\_MACMIIAR. During a PHY register access, this bit is set to 0b1 by the application to indicate that a read or write access is in progress. ETH\_MACMIIDR (MII Data) should be kept valid until this bit is cleared by the MAC during a PHY Write operation. The ETH\_MACMIIDR is invalid until this bit is cleared by the MAC during a PHY Read operation. The ETH\_MACMIIAR (MII Address) should not be written to until this bit is cleared.

### Ethernet MAC MII data register (ETH\_MACMIIDR)

Address offset: 0x0014

Reset value: 0x0000 0000

The MAC MII Data register stores write data to be written to the PHY register located at the address specified in ETH\_MACMIIAR. ETH\_MACMIIDR also stores read data from the PHY register located at the address specified by ETH\_MACMIIAR.

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
MD[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 **MD[15:0]**: MII data

This contains the 16-bit data value read from the PHY after a Management Read operation, or the 16-bit data value to be written to the PHY before a Management Write operation.

### Ethernet MAC flow control register (ETH\_MACFCR)

Address offset: 0x0018

Reset value: 0x0000 0000

The Flow control register controls the generation and reception of the control (Pause Command) frames by the MAC. A write to a register with the Busy bit set to '1' causes the MAC to generate a pause control frame. The fields of the control frame are selected as specified in the 802.3x specification, and the Pause Time value from this register is used in the Pause Time field of the control frame. The Busy bit remains set until the control frame is transferred onto the cable. The Host must make sure that the Busy bit is cleared before writing to the register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PT[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.	ZQPD	Res.	PLT[1:0]	UPFD	RFCE	TFCE	FCB/BPA	rc_w1/rw
								rw		rw	rw	rw	rw	rw	rc_w1/rw

Bits 31:16 **PT[15:0]**: Pause time

This field holds the value to be used in the Pause Time field in the transmit control frame. If the Pause Time bits is configured to be double-synchronized to the MII clock domain, then consecutive write operations to this register should be performed only after at least 4 clock cycles in the destination clock domain.

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **ZQPD**: Zero-quanta pause disable

When set, this bit disables the automatic generation of Zero-quanta pause control frames on the deassertion of the flow-control signal from the FIFO layer.

When this bit is reset, normal operation with automatic Zero-quanta pause control frame generation is enabled.

Bit 6 Reserved, must be kept at reset value.

**Bits 5:4 PLT:** Pause low threshold

This field configures the threshold of the Pause timer at which the Pause frame is automatically retransmitted. The threshold values should always be less than the Pause Time configured in bits[31:16]. For example, if PT = 100H (256 slot-times), and PLT = 01, then a second PAUSE frame is automatically transmitted if initiated at 228 (256 – 28) slot-times after the first PAUSE frame is transmitted.

## Selection Threshold

- 00 Pause time minus 4 slot times
- 01 Pause time minus 28 slot times
- 10 Pause time minus 144 slot times
- 11 Pause time minus 256 slot times

Slot time is defined as time taken to transmit 512 bits (64 bytes) on the MII interface.

**Bit 3 UPFD:** Unicast pause frame detect

When this bit is set, the MAC detects the Pause frames with the station's unicast address specified in the ETH\_MACA0HR and ETH\_MACA0LR registers, in addition to detecting Pause frames with the unique multicast address.

When this bit is reset, the MAC detects only a Pause frame with the unique multicast address specified in the 802.3x standard.

**Bit 2 RFCE:** Receive flow control enable

When this bit is set, the MAC decodes the received Pause frame and disables its transmitter for a specified (Pause Time) time.

When this bit is reset, the decode function of the Pause frame is disabled.

**Bit 1 TFCE:** Transmit flow control enable

In Full-duplex mode, when this bit is set, the MAC enables the flow control operation to transmit Pause frames. When this bit is reset, the flow control operation in the MAC is disabled, and the MAC does not transmit any Pause frames.

In Half-duplex mode, when this bit is set, the MAC enables the back-pressure operation. When this bit is reset, the back pressure feature is disabled.

**Bit 0 FCB/BPA:** Flow control busy/back pressure activate

This bit initiates a Pause Control frame in Full-duplex mode and activates the back pressure function in Half-duplex mode if TFCE bit is set.

In Full-duplex mode, this bit should be read as 0 before writing to the Flow control register. To initiate a Pause control frame, the Application must set this bit to 1. During a transfer of the Control frame, this bit continues to be set to signify that a frame transmission is in progress. After completion of the Pause control frame transmission, the MAC resets this bit to 0. The Flow control register should not be written to until this bit is cleared.

In Half-duplex mode, when this bit is set (and TFCE is set), back pressure is asserted by the MAC core. During back pressure, when the MAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision. When the MAC is configured to Full-duplex mode, the BPA is automatically disabled.

**Ethernet MAC VLAN tag register (ETH\_MACVLANTR)**

Address offset: 0x001C

Reset value: 0x0000 0000

The VLAN tag register contains the IEEE 802.1Q VLAN Tag to identify the VLAN frames. The MAC compares the 13<sup>th</sup> and 14<sup>th</sup> bytes of the receiving frame (Length/Type) with 0x8100, and the following 2 bytes are compared with the VLAN tag; if a match occurs, the received VLAN bit in the receive frame status is set. The legal length of the frame is increased from 1518 bytes to 1522 bytes.

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	VLANTC
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VLANTI[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

#### Bit 16 **VLANTC**: 12-bit VLAN tag comparison

When this bit is set, a 12-bit VLAN identifier, rather than the complete 16-bit VLAN tag, is used for comparison and filtering. Bits[11:0] of the VLAN tag are compared with the corresponding field in the received VLAN-tagged frame.

When this bit is reset, all 16 bits of the received VLAN frame's fifteenth and sixteenth bytes are used for comparison.

#### Bits 15:0 **VLANTI[15:0]**: VLAN tag identifier (for receive frames)

This contains the 802.1Q VLAN tag to identify VLAN frames, and is compared to the fifteenth and sixteenth bytes of the frames being received for VLAN frames. Bits[15:13] are the user priority, Bit[12] is the canonical format indicator (CFI) and bits[11:0] are the VLAN tag's VLAN identifier (VID) field. When the VLANTC bit is set, only the VID (bits[11:0]) is used for comparison.

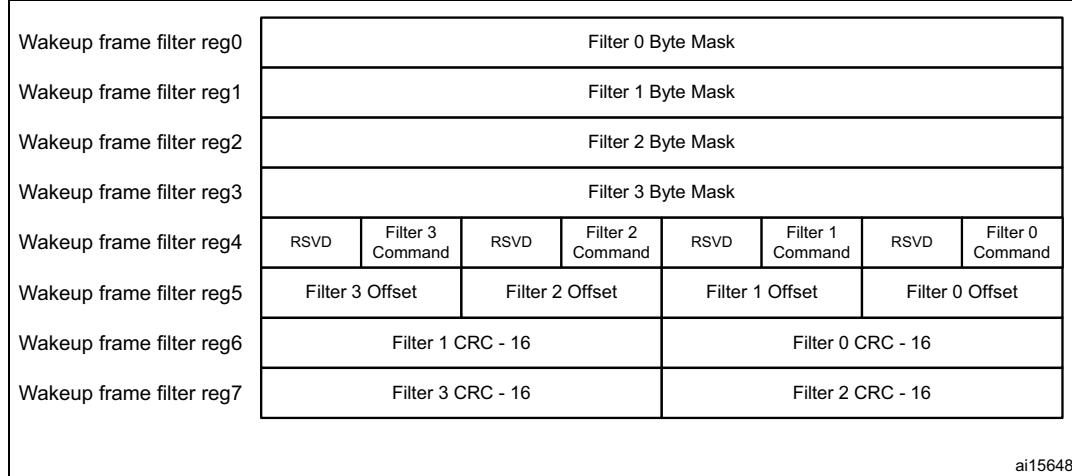
If VLANTI (VLANTI[11:0] if VLANTC is set) is all zeros, the MAC does not check the fifteenth and sixteenth bytes for VLAN tag comparison, and declares all frames with a Type field value of 0x8100 as VLAN frames.

## Ethernet MAC remote wakeup frame filter register (ETH\_MACRWUFFR)

Address offset: 0x0028

Reset value: 0x0000 0000

This is the address through which the remote wakeup frame filter registers are written/read by the application. The Wakeup frame filter register is actually a pointer to eight (not transparent) such wakeup frame filter registers. Eight sequential write operations to this address with the offset (0x0028) will write all wakeup frame filter registers. Eight sequential read operations from this address with the offset (0x0028) will read all wakeup frame filter registers. This register contains the higher 16 bits of the 7<sup>th</sup> MAC address. Refer to [Remote wakeup frame filter register](#) section for additional information.

**Figure 521. Ethernet MAC remote wakeup frame filter register (ETH\_MACRWUFFR)**

### Ethernet MAC PMT control and status register (ETH\_MACPMTCSR)

Address offset: 0x002C

Reset value: 0x0000 0000

The ETH\_MACPMTCSR programs the request wakeup events and monitors the wakeup events.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WFFRPR	Res.														
rs															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	GU	Res.	Res.	WFR	MPR	Res.	Res.	WFE	MPE	PD
					rw			rc_r	rc_r				rw	rw	rs

Bit 31 **WFFRPR**: Wakeup frame filter register pointer reset

When set, it resets the Remote wakeup frame filter register pointer to 0b000. It is automatically cleared after 1 clock cycle.

Bits 30:10 Reserved, must be kept at reset value.

Bit 9 **GU**: Global unicast

When set, it enables any unicast packet filtered by the MAC (DAF) address recognition to be a wakeup frame.

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **WFR**: Wakeup frame received

When set, this bit indicates the power management event was generated due to reception of a wakeup frame. This bit is cleared by a read into this register.

Bit 5 **MPR**: Magic packet received

When set, this bit indicates the power management event was generated by the reception of a Magic Packet. This bit is cleared by a read into this register.

Bits 4:3 Reserved, must be kept at reset value.

Bit 2 **WFE**: Wakeup frame enable

When set, this bit enables the generation of a power management event due to wakeup frame reception.

Bit 1 **MPE**: Magic Packet enable

When set, this bit enables the generation of a power management event due to Magic Packet reception.

Bit 0 **PD**: Power down

When this bit is set, all received frames will be dropped. This bit is cleared automatically when a magic packet or wakeup frame is received, and Power-down mode is disabled. Frames received after this bit is cleared are forwarded to the application. This bit must only be set when either the Magic Packet Enable or Wakeup Frame Enable bit is set high.

### Ethernet MAC debug register (ETH\_MACDBGR)

Address offset: 0x0034

Reset value: 0x0000 0000

This debug register gives the status of all the main modules of the transmit and receive data paths and the FIFOs. An all-zero status indicates that the MAC core is in Idle state (and FIFOs are empty) and no activity is going on in the data paths.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	TFF	TFNE	Res.	TFWA	TFRS[1:0]		MTP	MTFCS[1:0]		MMTEA
						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.	RFFL[1:0]		Res.	RFRCS[1:0]		RFWRA	Res.	MSFRWCS[1:0]		MMRPEA
						r	r		r	r	r		r	r	r

Bits 31:26 Reserved, must be kept at reset value.

Bit 25 **TFF**: Tx FIFO full

When high, it indicates that the Tx FIFO is full and hence no more frames will be accepted for transmission.

Bit 24 **TFNE**: Tx FIFO not empty

When high, it indicates that the Tx FIFO is not empty and has some data left for transmission.

Bit 23 Reserved, must be kept at reset value.

Bit 22 **TFWA**: Tx FIFO write active

When high, it indicates that the Tx FIFO write controller is active and transferring data to the Tx FIFO.

Bits 21:20 **TFRS[1:0]**: Tx FIFO read status

This indicates the state of the Tx FIFO read controller:

- 00: Idle state
- 01: Read state (transferring data to the MAC transmitter)
- 10: Waiting for TxStatus from MAC transmitter
- 11: Writing the received TxStatus or flushing the Tx FIFO

Bit 19 **MTP**: MAC transmitter in pause

When high, it indicates that the MAC transmitter is in Pause condition (in full-duplex mode only) and hence will not schedule any frame for transmission

Bits 18:17 **MTFCS[1:0]**: MAC transmit frame controller status

This indicates the state of the MAC transmit frame controller:

- 00: Idle
- 01: Waiting for Status of previous frame or IFG/backoff period to be over
- 10: Generating and transmitting a Pause control frame (in full duplex mode)
- 11: Transferring input frame for transmission

Bit 16 **MMTEA**: MAC MII transmit engine active

When high, it indicates that the MAC MII transmit engine is actively transmitting data and that it is not in the Idle state.

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **RFFL:** Rx FIFO fill level

This gives the status of the Rx FIFO fill-level:

- 00: RxFIFO empty
- 01: RxFIFO fill-level below flow-control de-activate threshold
- 10: RxFIFO fill-level above flow-control activate threshold
- 11: RxFIFO full

Bit 7 Reserved, must be kept at reset value.

Bits 6:5 **RFRCS[1:0]:** Rx FIFO read controller status

It gives the state of the Rx FIFO read controller:

- 00: IDLE state
- 01: Reading frame data
- 10: Reading frame status (or time-stamp)
- 11: Flushing the frame data and status

Bit 4 **RFWRA:** Rx FIFO write controller active

When high, it indicates that the Rx FIFO write controller is active and transferring a received frame to the FIFO.

Bit 3 Reserved, must be kept at reset value.

Bits 2:1 **MSFRWCS[1:0]:** MAC small FIFO read / write controllers status

When high, these bits indicate the respective active state of the small FIFO read and write controllers of the MAC receive frame controller module.

Bit 0 **MMRPEA:** MAC MII receive protocol engine active

When high, it indicates that the MAC MII receive protocol engine is actively receiving data and is not in the Idle state.

### Ethernet MAC interrupt status register (ETH\_MACSR)

Address offset: 0x0038

Reset value: 0x0000 0000

The ETH\_MACSR register contents identify the events in the MAC that can generate an interrupt.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	TSTS	Res.	Res.	MMCTS	MMCFS	MMCS	PMTS	Res.	Res.	Res.

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **TSTS:** Time stamp trigger status

This bit is set high when the system time value equals or exceeds the value specified in the Target time high and low registers. This bit is cleared by reading the ETH\_PTPTSSR register.

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **MMCTS:** MMC transmit status

This bit is set high whenever an interrupt is generated in the ETH\_MMCTIR Register. This bit is cleared when all the bits in this interrupt register (ETH\_MMCTIR) are cleared.

Bit 5 **MMCFS:** MMC receive status

This bit is set high whenever an interrupt is generated in the ETH\_MMCRIR register. This bit is cleared when all the bits in this interrupt register (ETH\_MMCRIR) are cleared.

Bit 4 **MMCS:** MMC status

This bit is set high whenever any of bits 6:5 is set high. It is cleared only when both bits are low.

Bit 3 **PMTS:** PMT status

This bit is set whenever a Magic packet or Wake-on-LAN frame is received in Power-down mode (See bits 5 and 6 in the ETH\_MACPMTCSR register *Ethernet MAC PMT control and status register (ETH\_MACPMTCSR)*). This bit is cleared when both bits[6:5], of this last register, are cleared due to a read operation to the ETH\_MACPMTCSR register.

Bits 2:0 Reserved, must be kept at reset value.

**Ethernet MAC interrupt mask register (ETH\_MACIMR)**

Address offset: 0x003C

Reset value: 0x0000 0000

The ETH\_MACIMR register bits make it possible to mask the interrupt signal due to the corresponding event in the ETH\_MACSR register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	TSTIM	Res.	Res.	Res.	Res.	Res.	PMTIM	Res.	Res.	Res.
						rw						rw			

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **TSTIM:** Time stamp trigger interrupt mask

When set, this bit disables the time stamp interrupt generation.

Bits 8:4 Reserved, must be kept at reset value.

Bit 3 **PMTIM:** PMT interrupt mask

When set, this bit disables the assertion of the interrupt signal due to the setting of the PMT Status bit in ETH\_MACSR.

Bits 2:0 Reserved, must be kept at reset value.

**Ethernet MAC address 0 high register (ETH\_MACA0HR)**

Address offset: 0x0040

Reset value: 0x8000 FFFF

The MAC address 0 high register holds the upper 16 bits of the 6-byte first MAC address of the station. Note that the first DA byte that is received on the MII interface corresponds to the LS Byte (bits [7:0]) of the MAC address low register. For example, if 0x1122 3344 5566 is received (0x11 is the first byte) on the MII as the destination address, then the MAC address 0 register [47:0] is compared with 0x6655 4433 2211.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MO	Res.														
rw															
MACA0H[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **MO**: Always 1.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **MACA0H[15:0]**: MAC address0 high [47:32]

This field contains the upper 16 bits (47:32) of the 6-byte MAC address0. This is used by the MAC for filtering for received frames and for inserting the MAC address in the transmit flow control (Pause) frames.

### Ethernet MAC address 0 low register (ETH\_MACA0LR)

Address offset: 0x0044

Reset value: 0xFFFF FFFF

The MAC address 0 low register holds the lower 32 bits of the 6-byte first MAC address of the station.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MACA0L[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
MACA0L[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MACA0L[31:0]**: MAC address0 low [31:0]

This field contains the lower 32 bits of the 6-byte MAC address0. This is used by the MAC for filtering for received frames and for inserting the MAC address in the transmit flow control (Pause) frames.

### Ethernet MAC address 1 high register (ETH\_MACA1HR)

Address offset: 0x0048

Reset value: 0x0000 FFFF

The MAC address 1 high register holds the upper 16 bits of the 6-byte second MAC address of the station.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AE	SA	MBC[5:0]								Res	Res	Res	Res	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
MACA1H[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **AE:** Address enable

When this bit is set, the address filters use the MAC address1 for perfect filtering. When this bit is cleared, the address filters ignore the address for filtering.

Bit 30 **SA:** Source address

When this bit is set, the MAC address1[47:0] is used for comparison with the SA fields of the received frame.

When this bit is cleared, the MAC address1[47:0] is used for comparison with the DA fields of the received frame.

Bits 29:24 **MBC[5:0]:** Mask byte control

These bits are mask control bits for comparison of each of the MAC address1 bytes. When they are set high, the MAC core does not compare the corresponding byte of received DA/SA with the contents of the MAC address1 registers. Each bit controls the masking of the bytes as follows:

- Bit 29: ETH\_MACA1HR [15:8]
- Bit 28: ETH\_MACA1HR [7:0]
- Bit 27: ETH\_MACA1LR [31:24]
- ...
- Bit 24: ETH\_MACA1LR [7:0]

Bits 23:16 Reserved, must be kept at reset value.

Bits 15:0 **MACA1H[15:0]:** MAC address1 high [47:32]

This field contains the upper 16 bits (47:32) of the 6-byte second MAC address.

**Ethernet MAC address1 low register (ETH\_MACA1LR)**

Address offset: 0x004C

Reset value: 0xFFFF FFFF

The MAC address 1 low register holds the lower 32 bits of the 6-byte second MAC address of the station.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MACA1L[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
MACA1L[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MACA1L[31:0]:** MAC address1 low [31:0]

This field contains the lower 32 bits of the 6-byte MAC address1. The content of this field is undefined until loaded by the application after the initialization process.

**Ethernet MAC address 2 high register (ETH\_MACA2HR)**

Address offset: 0x0050

Reset value: 0x0000 FFFF

The MAC address 2 high register holds the upper 16 bits of the 6-byte second MAC address of the station.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AE	SA	MBC[5:0]						Res							
rw	rw	rw	rw	rw	rw	rw	rw								
MACA2H[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **AE:** Address enable

When this bit is set, the address filters use the MAC address2 for perfect filtering. When reset, the address filters ignore the address for filtering.

Bit 30 **SA:** Source address

When this bit is set, the MAC address 2 [47:0] is used for comparison with the SA fields of the received frame.

When this bit is reset, the MAC address 2 [47:0] is used for comparison with the DA fields of the received frame.

Bits 29:24 **MBC[5:0]:** Mask byte control

These bits are mask control bits for comparison of each of the MAC address2 bytes. When set high, the MAC core does not compare the corresponding byte of received DA/SA with the contents of the MAC address 2 registers. Each bit controls the masking of the bytes as follows:

- Bit 29: ETH\_MACA2HR [15:8]
- Bit 28: ETH\_MACA2HR [7:0]
- Bit 27: ETH\_MACA2LR [31:24]
- ...
- Bit 24: ETH\_MACA2LR [7:0]

Bits 23:16 Reserved, must be kept at reset value.

Bits 15:0 **MACA2H[15:0]:** MAC address2 high [47:32]

This field contains the upper 16 bits (47:32) of the 6-byte MAC address2.

### Ethernet MAC address 2 low register (ETH\_MACA2LR)

Address offset: 0x0054

Reset value: 0xFFFF FFFF

The MAC address 2 low register holds the lower 32 bits of the 6-byte second MAC address of the station.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MACA2L[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
MACA2L[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MACA2L[31:0]:** MAC address2 low [31:0]

This field contains the lower 32 bits of the 6-byte second MAC address2. The content of this field is undefined until loaded by the application after the initialization process.

### Ethernet MAC address 3 high register (ETH\_MACA3HR)

Address offset: 0x0058

Reset value: 0x0000 FFFF

The MAC address 3 high register holds the upper 16 bits of the 6-byte second MAC address of the station.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AE	SA	MBC[5:0]						Re s.							
rw	rw	rw	rw	rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MACA3H[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **AE**: Address enable

When this bit is set, the address filters use the MAC address3 for perfect filtering. When this bit is cleared, the address filters ignore the address for filtering.

Bit 30 **SA**: Source address

When this bit is set, the MAC address 3 [47:0] is used for comparison with the SA fields of the received frame.

When this bit is cleared, the MAC address 3[47:0] is used for comparison with the DA fields of the received frame.

Bits 29:24 **MBC[5:0]**: Mask byte control

These bits are mask control bits for comparison of each of the MAC address3 bytes. When these bits are set high, the MAC core does not compare the corresponding byte of received DA/SA with the contents of the MAC address 3 registers. Each bit controls the masking of the bytes as follows:

- Bit 29: ETH\_MACA3HR [15:8]
- Bit 28: ETH\_MACA3HR [7:0]
- Bit 27: ETH\_MACA3LR [31:24]
- ...
- Bit 24: ETH\_MACA3LR [7:0]

Bits 23:16 Reserved, must be kept at reset value.

Bits 15:0 **MACA3H[15:0]**: MAC address3 high [47:32]

This field contains the upper 16 bits (47:32) of the 6-byte MAC address3.

### Ethernet MAC address 3 low register (ETH\_MACA3LR)

Address offset: 0x005C

Reset value: 0xFFFF FFFF

The MAC address 3 low register holds the lower 32 bits of the 6-byte second MAC address of the station.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MACA3L[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
MACA3L[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

**Bits 31:0 MACA3L[31:0]: MAC address3 low [31:0]**

This field contains the lower 32 bits of the 6-byte second MAC address3. The content of this field is undefined until loaded by the application after the initialization process.

### 38.8.2 MMC register description

#### Ethernet MMC control register (ETH\_MMCCR)

Address offset: 0x0100

Reset value: 0x0000 0000

The Ethernet MMC Control register establishes the operating mode of the management counters.

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	MCFHP	MCP	MCF	ROR	CSR	CR									
										rw	rw	rw	rw	rw	rw

Bits 31:6 Reserved, must be kept at reset value.

##### MCFHP: MMC counter Full-Half preset

When MCFHP is low and bit4 is set, all MMC counters get preset to almost-half value. All Bit 5 frame-counters get preset to 0xFFFF\_FFF0 (half - 16)

When MCFHP is high and bit4 is set, all MMC counters get preset to almost-full value. All frame-counters get preset to 0xFFFF\_FFF0 (full - 16)

##### MCP: MMC counter preset

When set, all counters will be initialized or preset to almost full or almost half as per Bit 4 Bit5 above. This bit will be cleared automatically after 1 clock cycle. This bit along with bit5 is useful for debugging and testing the assertion of interrupts due to MMC counter becoming half-full or full.

##### Bit 3 MCF: MMC counter freeze

When set, this bit freezes all the MMC counters to their current value. (None of the MMC counters are updated due to any transmitted or received frame until this bit is cleared to 0. If any MMC counter is read with the Reset on Read bit set, then that counter is also cleared in this mode.)

##### Bit 2 ROR: Reset on read

When this bit is set, the MMC counters is reset to zero after read (self-clearing after reset). The counters are cleared when the least significant byte lane (bits [7:0]) is read.

##### Bit 1 CSR: Counter stop rollover

When this bit is set, the counter does not roll over to zero after it reaches the maximum value.

##### Bit 0 CR: Counter reset

When it is set, all counters are reset. This bit is cleared automatically after 1 clock cycle.

#### Ethernet MMC receive interrupt register (ETH\_MMCRIR)

Address offset: 0x0104

Reset value: 0x0000 0000

The Ethernet MMC receive interrupt register maintains the interrupts generated when receive statistic counters reach half their maximum values. (MSB of the counter is set.) It is

a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits [7:0]) of the respective counter must be read in order to clear the interrupt bit.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	RGUFS	Res.									
														rc_r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RFAES	RF CES	Res.	Res.	Res.	Res.	Res.								
									rc_r	rc_r					

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **RGUFS:** Received Good Unicast Frames Status

This bit is set when the received, good unicast frames, counter reaches half the maximum value.

Bits 16:7 Reserved, must be kept at reset value.

Bit 6 **RFAES:** Received frames alignment error status

This bit is set when the received frames, with alignment error, counter reaches half the maximum value.

Bit 5 **RF CES:** Received frames CRC error status

This bit is set when the received frames, with CRC error, counter reaches half the maximum value.

Bits 4:0 Reserved, must be kept at reset value.

### Ethernet MMC transmit interrupt register (ETH\_MMCTIR)

Address offset: 0x0108

Reset value: 0x0000 0000

The Ethernet MMC transmit Interrupt register maintains the interrupts generated when transmit statistic counters reach half their maximum values. (MSB of the counter is set.) It is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits [7:0]) of the respective counter must be read in order to clear the interrupt bit.

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.	TGFS	Res.	Res.	Res.	Res.	Res.
										rc_r					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TGFMSCS	TGFSCS	Res.													
rc_r	rc_r														

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **TGFS:** Transmitted good frames status

This bit is set when the transmitted, good frames, counter reaches half the maximum value.

Bits 20:16 Reserved, must be kept at reset value.

Bit 15 **TGFMSCS:** Transmitted good frames more single collision status

This bit is set when the transmitted, good frames after more than a single collision, counter reaches half the maximum value.

Bit 14 **TGFSCS:** Transmitted good frames single collision status

This bit is set when the transmitted, good frames after a single collision, counter reaches half the maximum value.

Bits 13:0 Reserved, must be kept at reset value.

### Ethernet MMC receive interrupt mask register (ETH\_MMCRIMR)

Address offset: 0x010C

Reset value: 0x0000 0000

The Ethernet MMC receive interrupt mask register maintains the masks for interrupts generated when the receive statistic counters reach half their maximum value. (MSB of the counter is set.) It is a 32-bit wide register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	RGUFM	Res.									
														rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RFAEM	RFCEM	Res.	Res.	Res.	Res.	Res.								
									rw	rw					

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **RGUFM:** Received good unicast frames mask

Setting this bit masks the interrupt when the received, good unicast frames, counter reaches half the maximum value.

Bits 16:7 Reserved, must be kept at reset value.

Bit 6 **RFAEM:** Received frames alignment error mask

Setting this bit masks the interrupt when the received frames, with alignment error, counter reaches half the maximum value.

Bit 5 **RFCEM:** Received frame CRC error mask

Setting this bit masks the interrupt when the received frames, with CRC error, counter reaches half the maximum value.

Bits 4:0 Reserved, must be kept at reset value.

### Ethernet MMC transmit interrupt mask register (ETH\_MMCTIMR)

Address offset: 0x0110

Reset value: 0x0000 0000

The Ethernet MMC transmit interrupt mask register maintains the masks for interrupts generated when the transmit statistic counters reach half their maximum value. (MSB of the counter is set). It is a 32-bit wide 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.	TGFM	Res.	Res.	Res.	Res.	Res.
										rw					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TGFSCM	TGFSCM	Res.													
rw	rw														

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **TGFM:** Transmitted good frames mask

Setting this bit masks the interrupt when the transmitted, good frames, counter reaches half the maximum value.

Bits 20:16 Reserved, must be kept at reset value.

Bit 15 **TGFMSCM:** Transmitted good frames more single collision mask

Setting this bit masks the interrupt when the transmitted good frames after more than a single collision counter reaches half the maximum value.

Bit 14 **TGFSCM:** Transmitted good frames single collision mask

Setting this bit masks the interrupt when the transmitted good frames after a single collision counter reaches half the maximum value.

Bits 13:0 Reserved, must be kept at reset value.

### Ethernet MMC transmitted good frames after a single collision counter register (ETH\_MMCTGFSCCR)

Address offset: 0x014C

Reset value: 0x0000 0000

This register contains the number of successfully transmitted frames after a single collision in Half-duplex mode.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TGFSCC[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
TGFSCC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **TGFSCC[31:0]:** Transmitted good frames single collision counter

Transmitted good frames after a single collision counter.

### Ethernet MMC transmitted good frames after more than a single collision counter register (ETH\_MMCTGFMSCCR)

Address offset: 0x0150

Reset value: 0x0000 0000

This register contains the number of successfully transmitted frames after more than a single collision in Half-duplex mode.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TGFMSCC[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
TGFMSCC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **TGFMSCC[31:0]**: Transmitted good frames more single collision counter  
Transmitted good frames after more than a single collision counter

### Ethernet MMC transmitted good frames counter register (ETH\_MMCTGFCR)

Address offset: 0x0168

Reset value: 0x0000 0000

This register contains the number of good frames transmitted.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TGFC[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
TGFC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **TGFC[31:0]**: Transmitted good frames counter

### Ethernet MMC received frames with CRC error counter register (ETH\_MMCRFCECR)

Address offset: 0x0194

Reset value: 0x0000 0000

This register contains the number of frames received with CRC error.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RFCEC[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
RFCEC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **RFCEC[31:0]**: Received frames CRC error counter  
Received frames with CRC error counter

### Ethernet MMC received frames with alignment error counter register (ETH\_MMCRFAECR)

Address offset: 0x0198

Reset value: 0x0000 0000

This register contains the number of frames received with alignment (dribble) error.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RFAEC[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
RFAEC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **RFAEC[31:0]**: Received frames alignment error counter

Received frames with alignment error counter

### MMC received good unicast frames counter register (ETH\_MMCRGUFCR)

Address offset: 0x01C4

Reset value: 0x0000 0000

This register contains the number of good unicast frames received.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RGUFC[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
RGUFC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **RGUFC[31:0]**: Received good unicast frames counter

### 38.8.3 IEEE 1588 time stamp registers

This section describes the registers required to support precision network clock synchronization functions under the IEEE 1588 standard.

#### Ethernet PTP time stamp control register (ETH\_PTPTSCR)

Address offset: 0x0700

Reset value: 0x0000 2000

This register controls the time stamp generation and update logic.

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.	TSPFF MAE	TSCNT[1:0]
														rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSSMR ME	TSSEME	TSSIPV 4FE	TSSIPV 6FE	TSSPT POEFE	TSPTP PSV2E	TSSSR	TSSAR FE	Res.	Res.	TTSARU	TSITE	TSSTU	TSSTI	TSFCU	TSE
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 **TSPFFMAE**: Time stamp PTP frame filtering MAC address enable

When set, this bit uses the MAC address (except for MAC address 0) to filter the PTP frames when PTP is sent directly over Ethernet.

Bits 17:16 **TSCNT[1:0]**: Time stamp clock node type

The following are the available types of clock node:

- 00: Ordinary clock
- 01: Boundary clock
- 10: End-to-end transparent clock
- 11: Peer-to-peer transparent clock

Bit 15 **TSSMRME**: Time stamp snapshot for message relevant to master enable

When this bit is set, the snapshot is taken for messages relevant to the master node only. When this bit is cleared the snapshot is taken for messages relevant to the slave node only. This is valid only for the ordinary clock and boundary clock nodes.

Bit 14 **TSSEME**: Time stamp snapshot for event message enable

When this bit is set, the time stamp snapshot is taken for event messages only (SYNC, Delay\_Req, Pdelay\_Req or Pdelay\_Resp). When this bit is cleared the snapshot is taken for all other messages except for Announce, Management and Signaling.

Bit 13 **TSSIPV4FE**: Time stamp snapshot for IPv4 frames enable

When this bit is set, the time stamp snapshot is taken for IPv4 frames.

Bit 12 **TSSIPV6FE**: Time stamp snapshot for IPv6 frames enable

When this bit is set, the time stamp snapshot is taken for IPv6 frames.

Bit 11 **TSSPTPOEFE**: Time stamp snapshot for PTP over ethernet frames enable

When this bit is set, the time stamp snapshot is taken for frames which have PTP messages in Ethernet frames (PTP over Ethernet) also. By default snapshots are taken for UDP-IPEthernet PTP packets.

Bit 10 **TSPTPPSV2E**: Time stamp PTP packet snooping for version2 format enable

When this bit is set, the PTP packets are snooped using the version 2 format. When the bit is cleared, the PTP packets are snooped using the version 1 format.

*Note: IEEE 1588 Version 1 and Version 2 formats as indicated in IEEE standard 1588-2008 (Revision of IEEE STD. 1588-2002).*

Bit 9 **TSSSR**: Time stamp subsecond rollover: digital or binary rollover control

When this bit is set, the Time stamp low register rolls over when the subsecond counter reaches the value 0x3B9A C9FF (999 999 999 in decimal), and increments the Time Stamp (high) seconds.

When this bit is cleared, the rollover value of the subsecond register reaches 0x7FFF FFFF. The subsecond increment has to be programmed correctly depending on the PTP's reference clock frequency and this bit value.

Bit 8 **TSSARFE**: Time stamp snapshot for all received frames enable

When this bit is set, the time stamp snapshot is enabled for all frames received by the core.

Bits 7:6 Reserved, must be kept at reset value.

Bit 5 **TTSARU**: Time stamp addend register update

When this bit is set, the Time stamp addend register's contents are updated to the PTP block for fine correction. This bit is cleared when the update is complete. This register bit must be read as zero before you can set it.

Bit 4 **TSITE**: Time stamp interrupt trigger enable

When this bit is set, a time stamp interrupt is generated when the system time becomes greater than the value written in the Target time register. When the Time stamp trigger interrupt is generated, this bit is cleared.

Bit 3 **TSSTU**: Time stamp system time update

When this bit is set, the system time is updated (added to or subtracted from) with the value specified in the Time stamp high update and Time stamp low update registers. Both the TSSTU and TSSTI bits must be read as zero before you can set this bit. Once the update is completed in hardware, this bit is cleared.

Bit 2 **TSSTI**: Time stamp system time initialize

When this bit is set, the system time is initialized (overwritten) with the value specified in the Time stamp high update and Time stamp low update registers. This bit must be read as zero before you can set it. When initialization is complete, this bit is cleared.

Bit 1 **TSFCU**: Time stamp fine or coarse update

When set, this bit indicates that the system time stamp is to be updated using the Fine Update method. When cleared, it indicates the system time stamp is to be updated using the Coarse method.

Bit 0 **TSE**: Time stamp enable

When this bit is set, time stamping is enabled for transmit and receive frames. When this bit is cleared, the time stamp function is suspended and time stamps are not added for transmit and receive frames. Because the maintained system time is suspended, you must always initialize the time stamp feature (system time) after setting this bit high.

The table below indicates the messages for which a snapshot is taken depending on the clock, enable master and enable snapshot for event message register settings.

**Table 286. Time stamp snapshot dependency on registers bits**

TSCNT (bits 17:16)	TSSMRME (bit 15) <sup>(1)</sup>	TSSEME (bit 14)	Messages for which snapshots are taken
00 or 01	X <sup>(2)</sup>	0	SYNC, Follow_Up, Delay_Req, Delay_Resp
00 or 01	1	1	Delay_Req
00 or 01	0	1	SYNC
10	N/A	0	SYNC, Follow_Up, Delay_Req, Delay_Resp
10	N/A	1	SYNC, Follow_Up
11	N/A	0	SYNC, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp
11	N/A	1	SYNC, Pdelay_Req, Pdelay_Resp

1. N/A = not applicable.

2. X = don't care.

### Ethernet PTP subsecond increment register (ETH\_PTPSSIR)

Address offset: 0x0704

Reset value: 0x0000 0000

This register contains the 8-bit value by which the subsecond register is incremented. In Coarse update mode (TSFCU bit in ETH\_PTPTSCR), the value in this register is added to

the system time every clock cycle of HCLK. In Fine update mode, the value in this register is added to the system time whenever the accumulator gets an overflow.

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.								
								rw							
STSSI[7:0]															

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **STSSI[7:0]: System time subsecond increment**

The value programmed in this register is added to the contents of the subsecond value of the system time in every update.

For example, to achieve 20 ns accuracy, the value is:  $20 / 0.467 = \sim 43$  (or 0x2A).

### Ethernet PTP time stamp high register (ETH\_PTPTSHR)

Address offset: 0x0708

Reset value: 0x0000 0000

This register contains the most significant (higher) 32 time bits. This read-only register contains the seconds system time value. The Time stamp high register, along with Time stamp low register, indicates the current value of the system time maintained by the MAC. Though it is updated on a continuous basis.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STS[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
STS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **STS[31:0]: System time second**

The value in this field indicates the current value in seconds of the System Time maintained by the core.

### Ethernet PTP time stamp low register (ETH\_PTPTSLR)

Address offset: 0x070C

Reset value: 0x0000 0000

This register contains the least significant (lower) 32 time bits. This read-only register contains the subsecond system time value.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STPNS	STSS[30:16]															
	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	
	STSS[15:0]															
	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bit 31 **STPNS**: System time positive or negative sign

This bit indicates a positive or negative time value. When set, the bit indicates that time representation is negative. When cleared, it indicates that time representation is positive. Because the system time should always be positive, this bit is normally zero.

Bits 30:0 **STSS[30:0]**: System time subseconds

The value in this field has the subsecond time representation, with 0.46 ns accuracy.

### Ethernet PTP time stamp high update register (ETH\_PTPTSHUR)

Address offset: 0x0710

Reset value: 0x0000 0000

This register contains the most significant (higher) 32 bits of the time to be written to, added to, or subtracted from the System Time value. The Time stamp high update register, along with the Time stamp update low register, initializes or updates the system time maintained by the MAC. You have to write both of these registers before setting the TSSTI or TSSTU bits in the Time stamp control register.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TSUS[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	
	TSUS[15:0]															
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **TSUS[31:0]**: Time stamp update second

The value in this field indicates the time, in seconds, to be initialized or added to the system time.

### Ethernet PTP time stamp low update register (ETH\_PTPTSLUR)

Address offset: 0x0714

Reset value: 0x0000 0000

This register contains the least significant (lower) 32 bits of the time to be written to, added to, or subtracted from the System Time value.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TSUPNS	TSUSS[30:16]														
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSUSS[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **TSUPNS**: Time stamp update positive or negative sign

This bit indicates positive or negative time value. When set, the bit indicates that time representation is negative. When cleared, it indicates that time representation is positive. When TSSTI is set (system time initialization) this bit should be zero. If this bit is set when TSSTU is set, the value in the Time stamp update registers is subtracted from the system time. Otherwise it is added to the system time.

Bits 30:0 **TSUSS[30:0]**: Time stamp update subseconds

The value in this field indicates the subsecond time to be initialized or added to the system time. This value has an accuracy of 0.46 ns (in other words, a value of 0x0000\_0001 is 0.46 ns).

### Ethernet PTP time stamp addend register (ETH\_PTPTSAR)

Address offset: 0x0718

Reset value: 0x0000 0000

This register is used by the software to readjust the clock frequency linearly to match the master clock frequency. This register value is used only when the system time is configured for Fine update mode (TSFCU bit in ETH\_PTPTSCR). This register content is added to a 32-bit accumulator in every clock cycle and the system time is updated whenever the accumulator overflows.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TSA[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
TSA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **TSA[31:0]**: Time stamp addend

This register indicates the 32-bit time value to be added to the Accumulator register to achieve time synchronization.

### Ethernet PTP target time high register (ETH\_PTPTTHR)

Address offset: 0x071C

Reset value: 0x0000 0000

This register contains the higher 32 bits of time to be compared with the system time for interrupt event generation. The Target time high register, along with Target time low register, is used to schedule an interrupt event (TSARU bit in ETH\_PTPTSCR) when the system time exceeds the value programmed in these registers.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TTSH[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
TTSH[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **TTSH[31:0]: Target time stamp high**

This register stores the time in seconds. When the time stamp value matches or exceeds both Target time stamp registers, the MAC, if enabled, generates an interrupt.

### Ethernet PTP target time low register (ETH\_PTPTTLR)

Address offset: 0x0720

Reset value: 0x0000 0000

This register contains the lower 32 bits of time to be compared with the system time for interrupt event generation.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TTSL[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
TTSL[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **TTSL[31:0]: Target time stamp low**

This register stores the time in (signed) nanoseconds. When the value of the time stamp matches or exceeds both Target time stamp registers, the MAC, if enabled, generates an interrupt.

### Ethernet PTP time stamp status register (ETH\_PTPTSSR)

Address offset: 0x0728

Reset value: 0x0000 0000

This register contains the time stamp status register.

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.	TSTTR	TSSO													
														r	r

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **TSTTR**: Time stamp target time reached

When set, this bit indicates that the value of the system time is greater than or equal to the value specified in the Target time high and low registers. This bit is cleared when the ETH\_PTPTSSR register is read.

Bit 0 **TSSO**: Time stamp second overflow

When set, this bit indicates that the second value of the time stamp has overflowed beyond 0xFFFF FFFF.

### Ethernet PTP PPS control register (ETH\_PTTPPSCR)

Address offset: 0x072C

Reset value: 0x0000 0000

This register controls the frequency of the PPS output.

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.	PPSFREQ[3:0]														
														rw	rw

Bits 31:4 Reserved, must be kept at reset value.

Bits 3:0 **PPSFREQ[3:0]**: PPS frequency selection

The PPS output frequency is set to  $2^{PPSFREQ}$  Hz.

0000: 1 Hz with a pulse width of 125 ms for binary rollover and, of 100 ms for digital rollover

0001: 2 Hz with 50% duty cycle for binary rollover (digital rollover not recommended)

0010: 4 Hz with 50% duty cycle for binary rollover (digital rollover not recommended)

0011: 8 Hz with 50% duty cycle for binary rollover (digital rollover not recommended)

0100: 16 Hz with 50% duty cycle for binary rollover (digital rollover not recommended)

...

1111: 32768 Hz with 50% duty cycle for binary rollover (digital rollover not recommended)

*Note: If digital rollover is used (TSSSR=1, bit 9 in ETH\_PTPTSCR), it is recommended not to use the PPS output with a frequency other than 1 Hz. Otherwise, with digital rollover, the PPS output has irregular waveforms at higher frequencies (though its average frequency will always be correct during any one-second window).*

### 38.8.4 DMA register description

This section defines the bits for each DMA register. Non-32 bit accesses are allowed as long as the address is word-aligned.

#### Ethernet DMA bus mode register (ETH\_DMABMR)

Address offset: 0x1000

Reset value: 0x0002 0101

The bus mode register establishes the bus operating modes for the DMA.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	MB	AAB	FPM	USP	RDP[5:0]						FB	
					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	
PM[1:0]		PBL[5:0]						EDFE	DSL[4:0]						DA	SR
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 **MB:** Mixed burst

When this bit is set high and the FB bit is low, the AHB master interface starts all bursts of a length greater than 16 with INCR (undefined burst). When this bit is cleared, it reverts to fixed burst transfers (INCRx and SINGLE) for burst lengths of 16 and below.

Bit 25 **AAB:** Address-aligned beats

When this bit is set high and the FB bit equals 1, the AHB interface generates all bursts aligned to the start address LS bits. If the FB bit equals 0, the first burst (accessing the data buffer's start address) is not aligned, but subsequent bursts are aligned to the address.

Bit 24 **FPM:** 4xPBL mode

When set high, this bit multiplies the PBL value programmed (bits [22:17] and bits [13:8]) four times. Thus the DMA transfers data in a maximum of 4, 8, 16, 32, 64 and 128 beats depending on the PBL value.

Bit 23 **USP:** Use separate PBL

When set high, it configures the RxDMA to use the value configured in bits [22:17] as PBL while the PBL value in bits [13:8] is applicable to TxDMA operations only. When this bit is cleared, the PBL value in bits [13:8] is applicable for both DMA engines.

Bits 22:17 **RDP[5:0]:** Rx DMA PBL

These bits indicate the maximum number of beats to be transferred in one RxDMA transaction. This is the maximum value that is used in a single block read/write operation. The RxDMA always attempts to burst as specified in RDP each time it starts a burst transfer on the host bus. RDP can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior.

These bits are valid and applicable only when USP is set high.

Bit 16 **FB:** Fixed burst

This bit controls whether the AHB Master interface performs fixed burst transfers or not. When set, the AHB uses only SINGLE, INCR4, INCR8 or INCR16 during start of normal burst transfers. When reset, the AHB uses SINGLE and INCR burst transfer operations.

Bits 15:14 **PM[1:0]**: Rx Tx priority ratio

RxDMA requests are given priority over TxDMA requests in the following ratio:

- 00: 1:1
- 01: 2:1
- 10: 3:1
- 11: 4:1

This is valid only when the DA bit is cleared.

Bits 13:8 **PBL[5:0]**: Programmable burst length

These bits indicate the maximum number of beats to be transferred in one DMA transaction. This is the maximum value that is used in a single block read/write operation. The DMA always attempts to burst as specified in PBL each time it starts a burst transfer on the host bus. PBL can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. When USP is set, this PBL value is applicable for TxDMA transactions only.

The PBL values have the following limitations:

- The maximum number of beats (PBL) possible is limited by the size of the Tx FIFO and Rx FIFO.
- The FIFO has a constraint that the maximum beat supported is half the depth of the FIFO.
- If the PBL is common for both transmit and receive DMA, the minimum Rx FIFO and Tx FIFO depths must be considered.
- Do not program out-of-range PBL values, because the system may not behave properly.

Bit 7 **EDFE**: Enhanced descriptor format enable

When this bit is set, the enhanced descriptor format is enabled and the descriptor size is increased to 32 bytes (8 DWORDS). This is required when time stamping is activated (TSE=1, ETH\_PTPTSCR bit 0) or if IPv4 checksum offload is activated (IPCO=1, ETH\_MACCR bit 10).

Bits 6:2 **DSL[4:0]**: Descriptor skip length

This bit specifies the number of words to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When DSL value equals zero, the descriptor table is taken as contiguous by the DMA, in Ring mode.

Bit 1 **DA**: DMA Arbitration

- 0: Round-robin with Rx:Tx priority given in bits [15:14]
- 1: Rx has priority over Tx

Bit 0 **SR**: Software reset

When this bit is set, the MAC DMA controller resets all MAC Subsystem internal registers and logic. It is cleared automatically after the reset operation has completed in all of the core clock domains. Read a 0 value in this bit before re-programming any register of the core.

## Ethernet DMA transmit poll demand register (ETH\_DMATPDR)

Address offset: 0x1004

Reset value: 0x0000 0000

This register is used by the application to instruct the DMA to poll the transmit descriptor list. The transmit poll demand register enables the Transmit DMA to check whether or not the current descriptor is owned by DMA. The Transmit Poll Demand command is given to wake up the TxDMA if it is in Suspend mode. The TxDMA can go into Suspend mode due to an underflow error in a transmitted frame or due to the unavailability of descriptors owned by

transmit DMA. User can issue this command anytime and the TxDMA resets it once it starts re-fetching the current descriptor from host memory.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TPD[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
TPD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **TPD[31:0]**: Transmit poll demand

When these bits are written with any value, the DMA reads the current descriptor pointed to by the ETH\_DMACHTDR register. If that descriptor is not available (owned by Host), transmission returns to the Suspend state and ETH\_DMASR register bit 2 is asserted. If the descriptor is available, transmission resumes.

### EHERNET DMA receive poll demand register (ETH\_DMARPDR)

Address offset: 0x1008

Reset value: 0x0000 0000

This register is used by the application to instruct the DMA to poll the receive descriptor list. The Receive poll demand register enables the receive DMA to check for new descriptors. This command is given to wake up the RxDMA from Suspend state. The RxDMA can go into Suspend state only due to the unavailability of descriptors owned by it.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RPD[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
RPD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **RPD[31:0]**: Receive poll demand

When these bits are written with any value, the DMA reads the current descriptor pointed to by the ETH\_DMACHRDR register. If that descriptor is not available (owned by Host), reception returns to the Suspended state and ETH\_DMASR register bit 7 is not asserted. If the descriptor is available, the Receive DMA returns to active state.

### Ethernet DMA receive descriptor list address register (ETH\_DMARDLAR)

Address offset: 0x100C

Reset value: 0x0000 0000

The Receive descriptor list address register points to the start of the receive descriptor list. The descriptor list resides in the STM32F75xxx and STM32F74xxx physical memory space and must be word-aligned. The DMA internally converts it to bus-width aligned address by making the corresponding LS bits low. Writing to the ETH\_DMARDLAR register is permitted only when reception is stopped. When stopped, the ETH\_DMARDLAR register must be written to before the receive Start command is given.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRL[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
SRL[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **SRL[31:0]**: Start of receive list

This field contains the base address of the first descriptor in the receive descriptor list. The LSB bits [1/2/3:0] for 32/64/128-bit bus width) are internally ignored and taken as all-zero by the DMA. Hence these LSB bits are read only.

### Ethernet DMA transmit descriptor list address register (ETH\_DMATDLAR)

Address offset: 0x1010

Reset value: 0x0000 0000

The Transmit descriptor list address register points to the start of the transmit descriptor list. The descriptor list resides in the STM32F75xxx and STM32F74xxx physical memory space and must be word-aligned. The DMA internally converts it to bus-width-aligned address by taking the corresponding LSB to low. Writing to the ETH\_DMATDLAR register is permitted only when transmission has stopped. Once transmission has stopped, the ETH\_DMATDLAR register can be written before the transmission Start command is given.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STL[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
STL[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **STL[31:0]**: Start of transmit list

This field contains the base address of the first descriptor in the transmit descriptor list. The LSB bits [1/2/3:0] for 32/64/128-bit bus width) are internally ignored and taken as all-zero by the DMA. Hence these LSB bits are read-only.

### Ethernet DMA status register (ETH\_DMASR)

Address offset: 0x1014

Reset value: 0x0000 0000

The Status register contains all the status bits that the DMA reports to the application. The ETH\_DMASR register is usually read by the software driver during an interrupt service routine or polling. Most of the fields in this register cause the host to be interrupted. The ETH\_DMASR register bits are not cleared when read. Writing 1 to (unreserved) bits in ETH\_DMASR register[16:0] clears them and writing 0 has no effect. Each field (bits [16:0]) can be masked by masking the appropriate bit in the ETH\_DMAIER register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res.	TSTS	PMTS	MMCS	Res.	EBS[2:0]			TPS[2:0]			RPS[2:0]			NIS
		r	r	r		r	r	r	r	r	r	r	r	r	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIS	ERS	FBES	Res.	Res.	ETS	RWTS	RPSS	RBUS	RS	TUS	ROS	TJTS	TBUS	TPSS	TS
rc_w1	rc_w1	rc_w1			rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 **TSTS:** Time stamp trigger status

This bit indicates an interrupt event in the MAC core's Time stamp generator block. The software must read the MAC core's status register, clearing its source (bit 9), to reset this bit to 0. When this bit is high an interrupt is generated if enabled.

Bit 28 **PMTS:** PMT status

This bit indicates an event in the MAC core's PMT. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear its source to reset this bit to 0. The interrupt is generated when this bit is high if enabled.

Bit 27 **MMCS:** MMC status

This bit reflects an event in the MMC of the MAC core. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear the source of interrupt to make this bit as 0. The interrupt is generated when this bit is high if enabled.

Bit 26 Reserved, must be kept at reset value.

Bits 25:23 **EBS[2:0]:** Error bits status

These bits indicate the type of error that caused a bus error (error response on the AHB interface). Valid only with the fatal bus error bit (ETH\_DMASR register [13]) set. This field does not generate an interrupt.

Bit 231 Error during data transfer by TxDMA

0 Error during data transfer by RxDMA

Bit 24 1 Error during read transfer

0 Error during write transfer

Bit 25 1 Error during descriptor access

0 Error during data buffer access

Bits 22:20 **TPS[2:0]:** Transmit process state

These bits indicate the Transmit DMA FSM state. This field does not generate an interrupt.

000: Stopped; Reset or Stop Transmit Command issued

001: Running; Fetching transmit transfer descriptor

010: Running; Waiting for status

011: Running; Reading Data from host memory buffer and queuing it to transmit buffer (Tx FIFO)

100, 101: Reserved for future use

110: Suspended; Transmit descriptor unavailable or transmit buffer underflow

111: Running; Closing transmit descriptor

Bits 19:17 **RPS[2:0]**: Receive process state

These bits indicate the Receive DMA FSM state. This field does not generate an interrupt.

000: Stopped: Reset or Stop Receive Command issued

001: Running: Fetching receive transfer descriptor

010: Reserved for future use

011: Running: Waiting for receive packet

100: Suspended: Receive descriptor unavailable

101: Running: Closing receive descriptor

110: Reserved for future use

111: Running: Transferring the receive packet data from receive buffer to host memory

Bit 16 **NIS**: Normal interrupt summary

The normal interrupt summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in the ETH\_DMAIER register:

- ETH\_DMASR [0]: Transmit interrupt
- ETH\_DMASR [2]: Transmit buffer unavailable
- ETH\_DMASR [6]: Receive interrupt
- ETH\_DMASR [14]: Early receive interrupt

Only unmasked bits affect the normal interrupt summary bit.

This is a sticky bit and it must be cleared (by writing a 1 to this bit) each time a corresponding bit that causes NIS to be set is cleared.

Bit 15 **AIS**: Abnormal interrupt summary

The abnormal interrupt summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in the ETH\_DMAIER register:

- ETH\_DMASR [1]: Transmit process stopped
- ETH\_DMASR [3]: Transmit jabber timeout
- ETH\_DMASR [4]: Receive FIFO overflow
- ETH\_DMASR [5]: Transmit underflow
- ETH\_DMASR [7]: Receive buffer unavailable
- ETH\_DMASR [8]: Receive process stopped
- ETH\_DMASR [9]: Receive watchdog timeout
- ETH\_DMASR [10]: Early transmit interrupt
- ETH\_DMASR [13]: Fatal bus error

Only unmasked bits affect the abnormal interrupt summary bit.

This is a sticky bit and it must be cleared each time a corresponding bit that causes AIS to be set is cleared.

Bit 14 **ERS**: Early receive status

This bit indicates that the DMA had filled the first data buffer of the packet. Receive Interrupt ETH\_DMASR [6] automatically clears this bit.

Bit 13 **FBES**: Fatal bus error status

This bit indicates that a bus error occurred, as detailed in [25:23]. When this bit is set, the corresponding DMA engine disables all its bus accesses.

## Bits 12:11 Reserved, must be kept at reset value.

Bit 10 **ETS**: Early transmit status

This bit indicates that the frame to be transmitted was fully transferred to the Transmit FIFO.

Bit 9 **RWTS**: Receive watchdog timeout status

This bit is asserted when a frame with a length greater than 2 048 bytes is received.

Bit 8 **RPSS:** Receive process stopped status

This bit is asserted when the receive process enters the Stopped state.

Bit 7 **RBUS:** Receive buffer unavailable status

This bit indicates that the next descriptor in the receive list is owned by the host and cannot be acquired by the DMA. Receive process is suspended. To resume processing receive descriptors, the host should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, receive process resumes when the next recognized incoming frame is received. ETH\_DMASR [7] is set only when the previous receive descriptor was owned by the DMA.

Bit 6 **RS:** Receive status

This bit indicates the completion of the frame reception. Specific frame status information has been posted in the descriptor. Reception remains in the Running state.

Bit 5 **TUS:** Transmit underflow status

This bit indicates that the transmit buffer had an underflow during frame transmission. Transmission is suspended and an underflow error TDES0[1] is set.

Bit 4 **ROS:** Receive overflow status

This bit indicates that the receive buffer had an overflow during frame reception. If the partial frame is transferred to the application, the overflow status is set in RDES0[11].

Bit 3 **TJTS:** Transmit jabber timeout status

This bit indicates that the transmit jabber timer expired, meaning that the transmitter had been excessively active. The transmission process is aborted and placed in the Stopped state. This causes the transmit jabber timeout TDES0[14] flag to be asserted.

Bit 2 **TBUS:** Transmit buffer unavailable status

This bit indicates that the next descriptor in the transmit list is owned by the host and cannot be acquired by the DMA. Transmission is suspended. Bits [22:20] explain the transmit process state transitions. To resume processing transmit descriptors, the host should change the ownership of the bit of the descriptor and then issue a Transmit Poll Demand command.

Bit 1 **TPSS:** Transmit process stopped status

This bit is set when the transmission is stopped.

Bit 0 **TS:** Transmit status

This bit indicates that frame transmission is finished and TDES1[31] is set in the first descriptor.

**Ethernet DMA operation mode register (ETH\_DMAOMR)**

Address offset: 0x1018

Reset value: 0x0000 0000

The operation mode register establishes the Transmit and Receive operating modes and commands. The ETH\_DMAOMR register should be the last CSR to be written as part of DMA initialization.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	DTCEFD	RSF	DFRF	Res.	Res.	TSF	FTF	Res.	Res.	Res.	TTC[2]
					rw	rw	rw			rw	rs				rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTC[1:0]		ST	Res.	Res.	Res.	Res.	Res.	FEF	FUGF	Res.	RTC[1:0]		OSF	SR	Res.
rw	rw	rw						rw	rw		rw	rw	rw	rw	

Bits 31:27 Reserved, must be kept at reset value.

Bit 26 **DTCEFD:** Dropping of TCP/IP checksum error frames disable

When this bit is set, the core does not drop frames that only have errors detected by the receive checksum offload engine. Such frames do not have any errors (including FCS error) in the Ethernet frame received by the MAC but have errors in the encapsulated payload only. When this bit is cleared, all error frames are dropped if the FEF bit is reset.

Bit 25 **RSF:** Receive store and forward

When this bit is set, a frame is read from the Rx FIFO after the complete frame has been written to it, ignoring RTC bits. When this bit is cleared, the Rx FIFO operates in Cut-through mode, subject to the threshold specified by the RTC bits.

Bit 24 **DFRF:** Disable flushing of received frames

When this bit is set, the RxDMA does not flush any frames due to the unavailability of receive descriptors/buffers as it does normally when this bit is cleared. (See [Receive process suspended on page 1575](#))

Bits 23:22 Reserved, must be kept at reset value.

Bit 21 **TSF:** Transmit store and forward

When this bit is set, transmission starts when a full frame resides in the Transmit FIFO. When this bit is set, the TTC values specified by the ETH\_DMAOMR register bits [16:14] are ignored.

When this bit is cleared, the TTC values specified by the ETH\_DMAOMR register bits [16:14] are taken into account.

This bit should be changed only when transmission is stopped.

Bit 20 **FTF:** Flush transmit FIFO

When this bit is set, the transmit FIFO controller logic is reset to its default values and thus all data in the Tx FIFO are lost/flushed. This bit is cleared internally when the flushing operation is complete. The Operation mode register should not be written to until this bit is cleared.

Bits 19:17 Reserved, must be kept at reset value.

Bits 16:14 **TTC[2:0]:** Transmit threshold control

These three bits control the threshold level of the Transmit FIFO. Transmission starts when the frame size within the Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when the TSF bit (Bit 21) is cleared.

000: 64  
001: 128  
010: 192  
011: 256  
100: 40  
101: 32  
110: 24  
111: 16

**Bit 13 ST:** Start/stop transmission

When this bit is set, transmission is placed in the Running state, and the DMA checks the transmit list at the current position for a frame to be transmitted. Descriptor acquisition is attempted either from the current position in the list, which is the transmit list base address set by the ETH\_DMATDLAR register, or from the position retained when transmission was stopped previously. If the current descriptor is not owned by the DMA, transmission enters the Suspended state and the transmit buffer unavailable bit (ETH\_DMASR [2]) is set. The Start Transmission command is effective only when transmission is stopped. If the command is issued before setting the DMA ETH\_DMATDLAR register, the DMA behavior is unpredictable.

When this bit is cleared, the transmission process is placed in the Stopped state after completing the transmission of the current frame. The next descriptor position in the transmit list is saved, and becomes the current position when transmission is restarted. The Stop Transmission command is effective only when the transmission of the current frame is complete or when the transmission is in the Suspended state.

Bits 12:8 Reserved, must be kept at reset value.

**Bit 7 FEF:** Forward error frames

When this bit is set, all frames except runt error frames are forwarded to the DMA.

When this bit is cleared, the Rx FIFO drops frames with error status (CRC error, collision error, giant frame, watchdog timeout, overflow). However, if the frame's start byte (write) pointer is already transferred to the read controller side (in Threshold mode), then the frames are not dropped. The Rx FIFO drops the error frames if that frame's start byte is not transferred (output) on the ARI bus.

**Bit 6 FUGF:** Forward undersized good frames

When this bit is set, the Rx FIFO forwards undersized frames (frames with no error and length less than 64 bytes) including pad-bytes and CRC).

When this bit is cleared, the Rx FIFO drops all frames of less than 64 bytes, unless such a frame has already been transferred due to lower value of receive threshold (e.g., RTC = 01).

Bit 5 Reserved, must be kept at reset value.

**Bits 4:3 RTC[1:0]:** Receive threshold control

These two bits control the threshold level of the Receive FIFO. Transfer (request) to DMA starts when the frame size within the Receive FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are transferred automatically.

*Note: Note that value of 11 is not applicable if the configured Receive FIFO size is 128 bytes.*

*Note: These bits are valid only when the RSF bit is zero, and are ignored when the RSF bit is set to 1.*

00: 64

01: 32

10: 96

11: 128

**Bit 2 OSF:** Operate on second frame

When this bit is set, this bit instructs the DMA to process a second frame of Transmit data even before status for first frame is obtained.

Bit 1 **SR:** Start/stop receive

When this bit is set, the receive process is placed in the Running state. The DMA attempts to acquire the descriptor from the receive list and processes incoming frames. Descriptor acquisition is attempted from the current position in the list, which is the address set by the DMA ETH\_DMARDLAR register or the position retained when the receive process was previously stopped. If no descriptor is owned by the DMA, reception is suspended and the receive buffer unavailable bit (ETH\_DMASR [7]) is set. The Start Receive command is effective only when reception has stopped. If the command was issued before setting the DMA ETH\_DMARDLAR register, the DMA behavior is unpredictable.

When this bit is cleared, RxDMA operation is stopped after the transfer of the current frame. The next descriptor position in the receive list is saved and becomes the current position when the receive process is restarted. The Stop Receive command is effective only when the Receive process is in either the Running (waiting for receive packet) or the Suspended state.

Bit 0 Reserved, must be kept at reset value.

**Ethernet DMA interrupt enable register (ETH\_DMAIER)**

Address offset: 0x101C

Reset value: 0x0000 0000

The Interrupt enable register enables the interrupts reported by ETH\_DMASR. Setting a bit to 1 enables a corresponding interrupt. After a hardware or software reset, all interrupts are disabled.

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.	NISE	
																rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
AISE	ERIE	FBEIE	Res.	Res.	ETIE	RWTIE	RPSIE	RBUIE	RIE	TUIE	ROIE	TJTIE	TBUIE	TPSIE	TIE	
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 **NISE:** Normal interrupt summary enable

When this bit is set, a normal interrupt is enabled. When this bit is cleared, a normal interrupt is disabled. This bit enables the following bits:

- ETH\_DMASR [0]: Transmit Interrupt
- ETH\_DMASR [2]: Transmit buffer unavailable
- ETH\_DMASR [6]: Receive interrupt
- ETH\_DMASR [14]: Early receive interrupt

**Bit 15 AISE:** Abnormal interrupt summary enable

When this bit is set, an abnormal interrupt is enabled. When this bit is cleared, an abnormal interrupt is disabled. This bit enables the following bits:

- ETH\_DMASR [1]: Transmit process stopped
- ETH\_DMASR [3]: Transmit jabber timeout
- ETH\_DMASR [4]: Receive overflow
- ETH\_DMASR [5]: Transmit underflow
- ETH\_DMASR [7]: Receive buffer unavailable
- ETH\_DMASR [8]: Receive process stopped
- ETH\_DMASR [9]: Receive watchdog timeout
- ETH\_DMASR [10]: Early transmit interrupt
- ETH\_DMASR [13]: Fatal bus error

**Bit 14 ERIE:** Early receive interrupt enable

When this bit is set with the normal interrupt summary enable bit (ETH\_DMAIER register[16]), the early receive interrupt is enabled.

When this bit is cleared, the early receive interrupt is disabled.

**Bit 13 FBEIE:** Fatal bus error interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register[15]), the fatal bus error interrupt is enabled.

When this bit is cleared, the fatal bus error enable interrupt is disabled.

Bits 12:11 Reserved, must be kept at reset value.

**Bit 10 ETIE:** Early transmit interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register [15]), the early transmit interrupt is enabled.

When this bit is cleared, the early transmit interrupt is disabled.

**Bit 9 RWTIE:** receive watchdog timeout interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register[15]), the receive watchdog timeout interrupt is enabled.

When this bit is cleared, the receive watchdog timeout interrupt is disabled.

**Bit 8 RPSIE:** Receive process stopped interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register[15]), the receive stopped interrupt is enabled. When this bit is cleared, the receive stopped interrupt is disabled.

**Bit 7 RBUIE:** Receive buffer unavailable interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register[15]), the receive buffer unavailable interrupt is enabled.

When this bit is cleared, the receive buffer unavailable interrupt is disabled.

**Bit 6 RIE:** Receive interrupt enable

When this bit is set with the normal interrupt summary enable bit (ETH\_DMAIER register[16]), the receive interrupt is enabled.

When this bit is cleared, the receive interrupt is disabled.

**Bit 5 TUIE:** Underflow interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register[15]), the transmit underflow interrupt is enabled.

When this bit is cleared, the underflow interrupt is disabled.

Bit 4 **ROIE:** Overflow interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register[15]), the receive overflow interrupt is enabled.

When this bit is cleared, the overflow interrupt is disabled.

Bit 3 **TJIE:** Transmit jabber timeout interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register[15]), the transmit jabber timeout interrupt is enabled.

When this bit is cleared, the transmit jabber timeout interrupt is disabled.

Bit 2 **TBUIE:** Transmit buffer unavailable interrupt enable

When this bit is set with the normal interrupt summary enable bit (ETH\_DMAIER register[16]), the transmit buffer unavailable interrupt is enabled.

When this bit is cleared, the transmit buffer unavailable interrupt is disabled.

Bit 1 **TPSIE:** Transmit process stopped interrupt enable

When this bit is set with the abnormal interrupt summary enable bit (ETH\_DMAIER register[15]), the transmission stopped interrupt is enabled.

When this bit is cleared, the transmission stopped interrupt is disabled.

Bit 0 **TIE:** Transmit interrupt enable

When this bit is set with the normal interrupt summary enable bit (ETH\_DMAIER register[16]), the transmit interrupt is enabled.

When this bit is cleared, the transmit interrupt is disabled.

The Ethernet interrupt is generated only when the TSTS or PMTS bits of the DMA Status register is asserted with their corresponding interrupt are unmasked, or when the NIS/AIS Status bit is asserted and the corresponding Interrupt Enable bits (NISE/AISE) are enabled.

### Ethernet DMA missed frame and buffer overflow counter register (ETH\_DMAMFBOCR)

Address offset: 0x1020

Reset value: 0x0000 0000

The DMA maintains two counters to track the number of missed frames during reception. This register reports the current value of the counter. The counter is used for diagnostic purposes. Bits [15:0] indicate missed frames due to the STM32F75xxx and STM32F74xxx buffer being unavailable (no receive descriptor was available). Bits [27:17] indicate missed frames due to Rx FIFO overflow conditions and runt frames (good frames of less than 64 bytes).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
Res.	Res.	Res.	OFOC	MFA[10:0]														OMFC
				rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
MFC[15:0]																		
rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r	rc_r			

Bits 31:29 Reserved, must be kept at reset value.

Bit 28 **OFOC:** Overflow bit for FIFO overflow counter

Bits 27:17 **MFA[10:0]:** Missed frames by the application

Indicates the number of frames missed by the application

Bit 16 **OMFC**: Overflow bit for missed frame counter

Bits 15:0 **MFC[15:0]**: Missed frames by the controller

Indicates the number of frames missed by the Controller due to the host receive buffer being unavailable. This counter is incremented each time the DMA discards an incoming frame.

### Ethernet DMA receive status watchdog timer register (ETH\_DMARSWTR)

Address offset: 0x1024

Reset value: 0x0000 0000

This register, when written with a non-zero value, enables the watchdog timer for the receive status (RS, ETH\_DMASR[6]).

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.	RSWTC[7:0]														
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **RSWTC[7:0]**: Receive status (RS) watchdog timer count

Indicates the number of HCLK clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the RxDMA completes the transfer of a frame for which the RS status bit is not set due to the setting of RDES1[31] in the corresponding descriptor. When the watchdog timer runs out, the RS bit is set and the timer is stopped. The watchdog timer is reset when the RS bit is set high due to automatic setting of RS as per RDES1[31] of any received frame.

### Ethernet DMA current host transmit descriptor register (ETH\_DMACHTDR)

Address offset: 0x1048

Reset value: 0x0000 0000

The Current host transmit descriptor register points to the start address of the current transmit descriptor read by the DMA.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HTDAP[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
HTDAP[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **HTDAP[31:0]**: Host transmit descriptor address pointer

Cleared . Pointer updated by DMA during operation.

**Ethernet DMA current host receive descriptor register (ETH\_DMACHRDR)**

Address offset: 0x104C

Reset value: 0x0000 0000

The Current host receive descriptor register points to the start address of the current receive descriptor read by the DMA.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HRDAP[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
HRDAP[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **HRDAP[31:0]: Host receive descriptor address pointer**

Cleared On Reset. Pointer updated by DMA during operation.

**Ethernet DMA current host transmit buffer address register (ETH\_DMACTBAP)**

Address offset: 0x1050

Reset value: 0x0000 0000

The Current host transmit buffer address register points to the current transmit buffer address being read by the DMA.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HTBAP[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
HTBAP[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **HTBAP[31:0]: Host transmit buffer address pointer**

Cleared On Reset. Pointer updated by DMA during operation.

**Ethernet DMA current host receive buffer address register  
(ETH\_DMACHRBAR)**

Address offset: 0x1054

Reset value: 0x0000 0000

The current host receive buffer address register points to the current receive buffer address being read by the DMA.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HRBAP[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
HRBAP[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **HRBAP[31:0]**: Host receive buffer address pointer

Cleared On Reset. Pointer updated by DMA during operation.

### **38.8.5 Ethernet register maps**

[Table 287](#) gives the ETH register map and reset values.

**Table 287. Ethernet register map and reset values**

Off-set	Register name reset value	Register name reset value	
0x00	ETH_MACCR	Res.	31
	Reset value	Res.	Res.
0x04	ETH_MACFFR	RA	30
	Reset value	0	Res.
0x08	ETH_MACHTHR	Res.	
	Reset value	0 0	HTH[31:0]
0x0C	ETH_MACHTLR	Res.	
	Reset value	0 0	HTL[31:0]
0x10	ETH_MACMIIAR	Res.	Res.
	Reset value	0 0	Res.
0x14	ETH_MACMIIDR	Res.	WD
	Reset value	0 0	JD
0x18	ETH_MACFCR	Res.	
	Reset value	0 0	PTT[15:0]
0x1C	ETH_MACVLANTR	Res.	
	Reset value	0 0	Res.
0x28	ETH_MACRWUFFR	Frame filter reg0\Frame filter reg1\Frame filter reg2\Frame filter reg3\Frame filter reg4...\Frame filter reg7	
	Reset value	0	
0x2C	ETH_MACPMTC SR	WFFRPR	16
-	Reset value	0	Res.
0x34	ETH_MACDBGR	Res.	
	Reset value	Res.	Res.
0x38	ETH_MACCSR	Res.	
	Reset value	Res.	Res.
0x3C	ETH_MACIMR	Res.	
	Reset value	Res.	Res.
0x40	ETH_MACA0HR	Res.	
	Reset value	1 1	MACA0H[15:0]

Table 287. Ethernet register map and reset values (continued)

Off-set	Register name reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
0x44	ETH_MACA0LR																																						
	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	1	1	1	1	1	1	1	1		
0x48	ETH_MACA1HR	AE	SA			MBC[5:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0																													
0x4C	ETH_MACA1LR																																						
	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	1	1	1	1	1	1	1	1		
0x50	ETH_MACA2HR	AE	SA			MBC[5:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0																													
0x54	ETH_MACA2LR																																						
	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	1	1	1	1	1	1	1	1		
0x58	ETH_MACA3HR	AE	SA			MBC[5:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0																													
0x5C	ETH_MACA3LR																																						
	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	1	1	1	1	1	1	1	1		
0x100	ETH_MMCCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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																																						
0x104	ETH_MMCRIR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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																																						
0x108	ETH_MMCTIR																																						
	Reset value																																						
0x10C	ETH_MMCRIMR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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																																						
0x110	ETH_MMCTIMR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGFM	Res.	0	TGFS	Res.																
	Reset value																																						
0x14C	ETH_MMCTGFS_CCR																																						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x150	ETH_MMCTGMSCCR																																						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x168	ETH_MMCTGFCSR																																						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x194	ETH_MMCRFCECR																																						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x198	ETH_MMCRFAECR																																						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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 287. Ethernet register map and reset values (continued)**

**Table 287. Ethernet register map and reset values (continued)**

Off-set	Register name reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x101C	ETH_DMAIER	Res	Res	Res	Res	Res	Res	Res	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																																			
0x1020	ETH_DMAMFBO CR	Res	Res	Res	Res	OFOC	[MFA[10:0]																													
	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			
0x1024	ETH_DMARSWTR	Res	Res	Res	Res	Res	Res	Res	Res	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																																			
0x1048	ETH_DMACHTDR																																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x104C	ETH_DMACHRDR																																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x1050	ETH_DMACHTBAR																																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x1054	ETH_DMACHRBAR																																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	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 69](#) for the register boundary addresses.

## 39 HDMI-CEC controller (CEC)

### 39.1 HDMI-CEC introduction

Consumer electronics control (CEC) is part of HDMI (high-definition multimedia interface) standard. It contains a protocol that provides high-level control functions between various audiovisual products. CEC operates at low speeds, with minimum processing and memory overhead.

The HDMI-CEC controller provides hardware support for this protocol.

### 39.2 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)

## 39.3 HDMI-CEC functional description

### 39.3.1 HDMI-CEC pin

The CEC bus consists of a single bidirectional line that is used to transfer data in and out of the device. It is connected to a +3.3 V supply voltage via a 27 kΩ pull-up resistor. The output stage of the device must have an open-drain or open-collector to allow a wired-AND connection.

The HDMI-CEC controller manages the CEC bidirectional line as an alternate function of a standard GPIO, assuming that it is configured as alternate function open drain. The 27 kΩ pull-up must be added externally to the microcontroller.

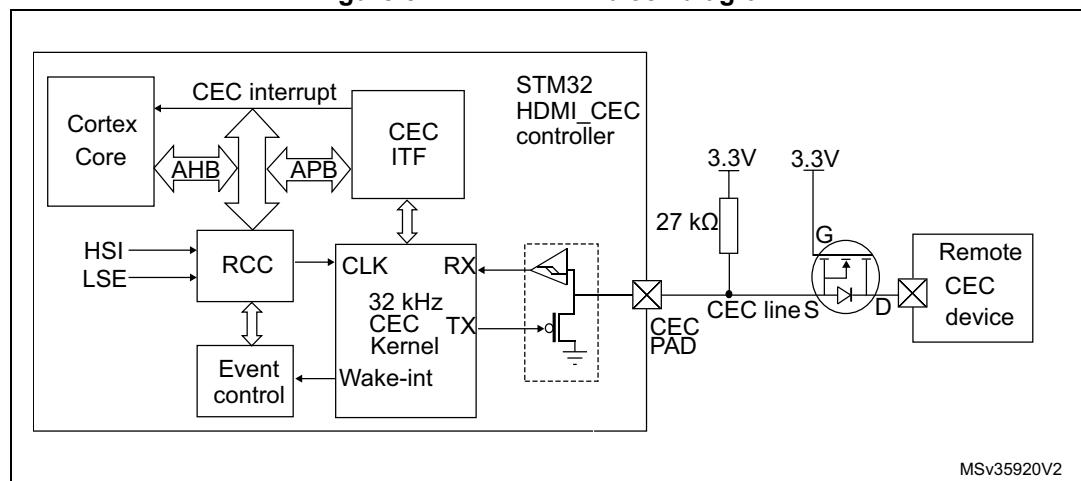
To not interfere with the CEC bus when the application power is removed, it is mandatory to isolate the CEC pin from the bus in such conditions. This can be done by using a MOS transistor, as shown on [Figure 522](#).

**Table 288. HDMI pin**

Name	Signal type	Remarks
CEC	Bidirectional	Two states: – 1 = high impedance – 0 = low impedance A 27 kΩ resistor must be added externally.

### 39.3.2 HDMI-CEC block diagram

**Figure 522. HDMI-CEC block diagram**



### 39.3.3 Message description

All transactions on the CEC line consist of an initiator and one or more followers. The initiator is responsible for sending the message structure and the data. The follower is the recipient of any data and is responsible for setting any acknowledgment bits.

A message is conveyed in a single frame that consists of a start bit followed by a header block and optionally an opcode and a variable number of operand blocks.

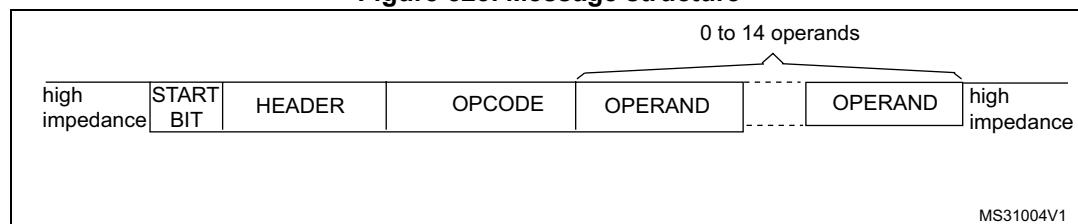
All these blocks are made of a 8-bit payload - most significant bit is transmitted first - followed by an end of message (EOM) bit and an acknowledge (ACK) bit.

The EOM bit is set in the last block of a message and kept reset in all others. In case a message contains additional blocks after an EOM is indicated, those additional blocks must be ignored. The EOM bit may be set in the header block to 'ping' other devices, to make sure they are active.

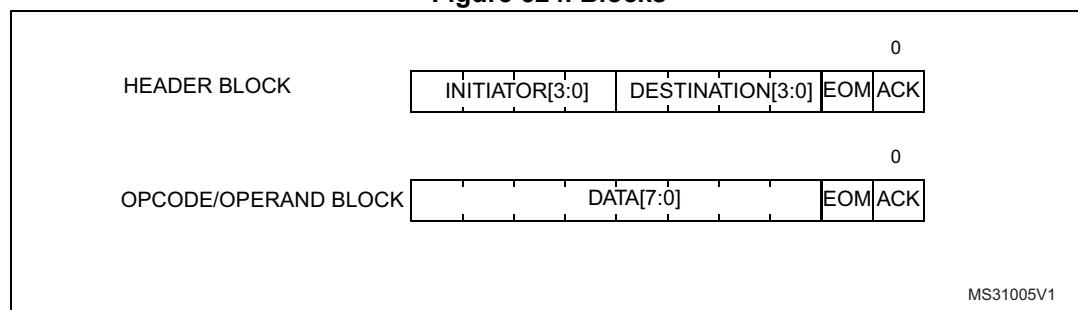
The acknowledge bit is always set to high impedance by the initiator so that it can be driven low either by the follower that has read its own address in the header, or by the follower that needs to reject a broadcast message.

The header consists of the source logical address field, and the destination logical address field. Note that the special address 0xF is used for broadcast messages.

**Figure 523. Message structure**



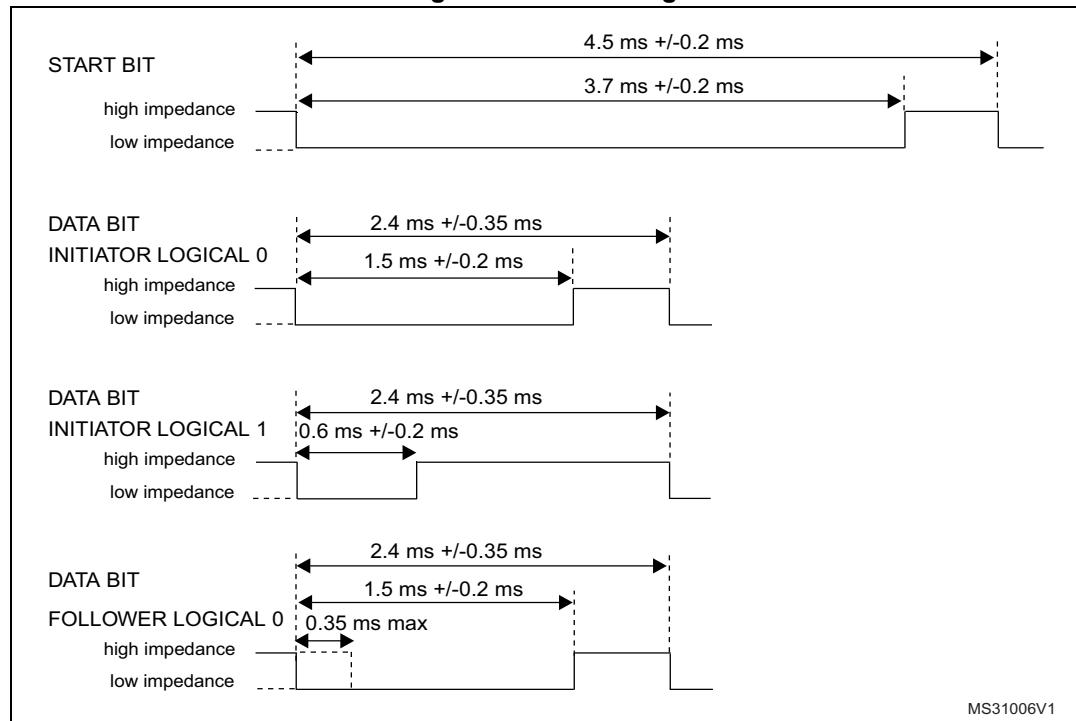
**Figure 524. Blocks**



### 39.3.4 Bit timing

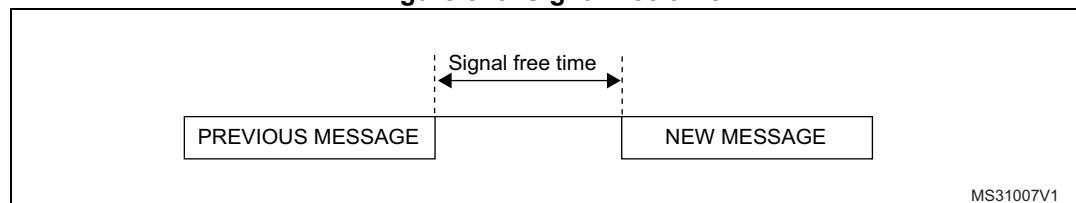
The format of the start bit is unique and identifies the start of a message. It must be validated by its low duration and its total duration.

All remaining data bits in the message, after the start bit, have consistent timing. The high-to-low transition at the end of the data bit is the start of the next data bit except for the final bit where the CEC line remains high.

**Figure 525. Bit timings**

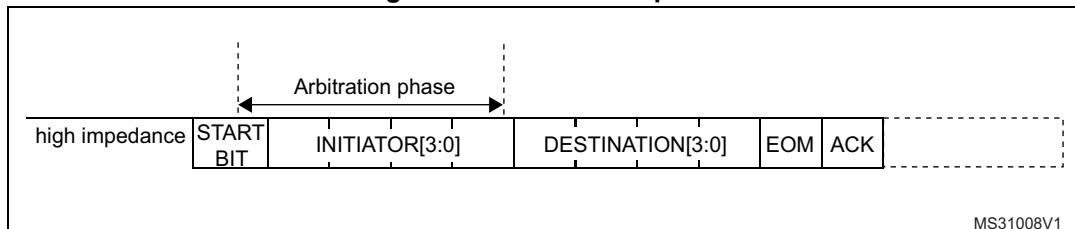
## 39.4 Arbitration

All devices transmitting - or retransmitting - a message onto the CEC line must ensure that it has been inactive for a number of bit periods. This signal-free time is defined as the time starting from the final bit of the previous frame and depends on the initiating device and the current status as shown in the figure below.

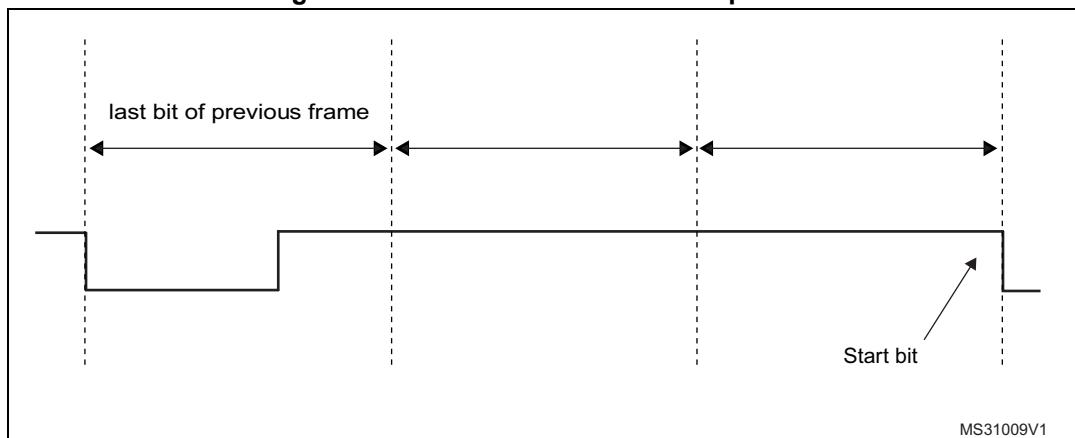
**Figure 526. Signal free time**

Since only one initiator is allowed at any one time, an arbitration mechanism is provided to avoid conflict when more than one initiator begins transmitting at the same time.

CEC line arbitration starts with the leading edge of the start bit and continues until the end of the initiator address bits within the header block. During this period, the initiator must monitor the CEC line, if whilst driving the line to high impedance it reads it back to 0. Assuming then it has lost arbitration, it stops transmitting and becomes a follower.

**Figure 527. Arbitration phase**

[Figure 528](#) shows an example for a SFT of three nominal bit periods.

**Figure 528. SFT of three nominal bit periods**

A configurable time window is counted before starting the transmission.

In the SFT = 0 configuration, HDMI-CEC performs automatic SFT calculation ensuring compliance with the HDMI-CEC standard:

- 2.5 data bit periods if the CEC is the last bus initiator with unsuccessful transmission
- 4 data bit periods if the CEC is the new bus initiator
- 6 data bit periods if the CEC is the last bus initiator with successful transmission

This is done to guarantee the maximum priority to a failed transmission and the lowest one to the last initiator that completed successfully its transmission.

Otherwise there is the possibility to configure the SFT bits to count a fixed timing value. Possible values are 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5 data bit periods.

### 39.4.1 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:

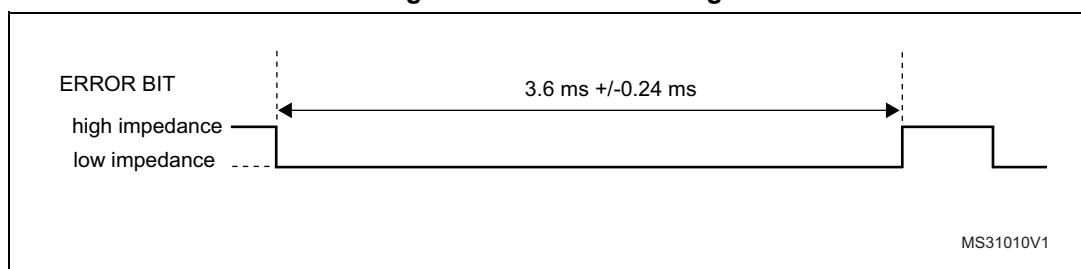
- In case of a regular end of transmission/reception, when TXEND/RXEND bits are set at the minimum nominal data bit duration of the last bit in the message (ACK bit).
- In case of a transmission error detection, SFT timer starts when the TXERR transmission error is detected (TXERR = 1).
- In case of 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.

## 39.5 Error handling

### 39.5.1 Bit error

If a data bit - excluding the start bit - is considered invalid, the follower is expected to notify such error by generating a low bit period on the CEC line of 1.4 to 1.6 times the nominal data bit period (3.6 ms nominally).

**Figure 529. Error bit timing**



### 39.5.2 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:

### 39.5.3 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 530](#)). BRE flag also generates a CEC interrupt if the BREIE = 1.

In the case of a BRE detection, the message reception can be stopped according to the BRESTP bit value and an error bit can be generated if BREGEN bit is set.

When BRE is detected in a broadcast message with BRESTP = 1 an error bit is generated even if BREGEN = 0 to enforce initiator's retry of the failed transmission. Error bit generation can be disabled by configuring BREGEN = 0, BRDNOGEN = 1.

### 39.5.4 Short bit period error (SBPE)

SBPE is set when a bit falling edge is detected earlier than expected (see [Figure 530](#)). 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

### 39.5.5 Long bit period error (LBPE)

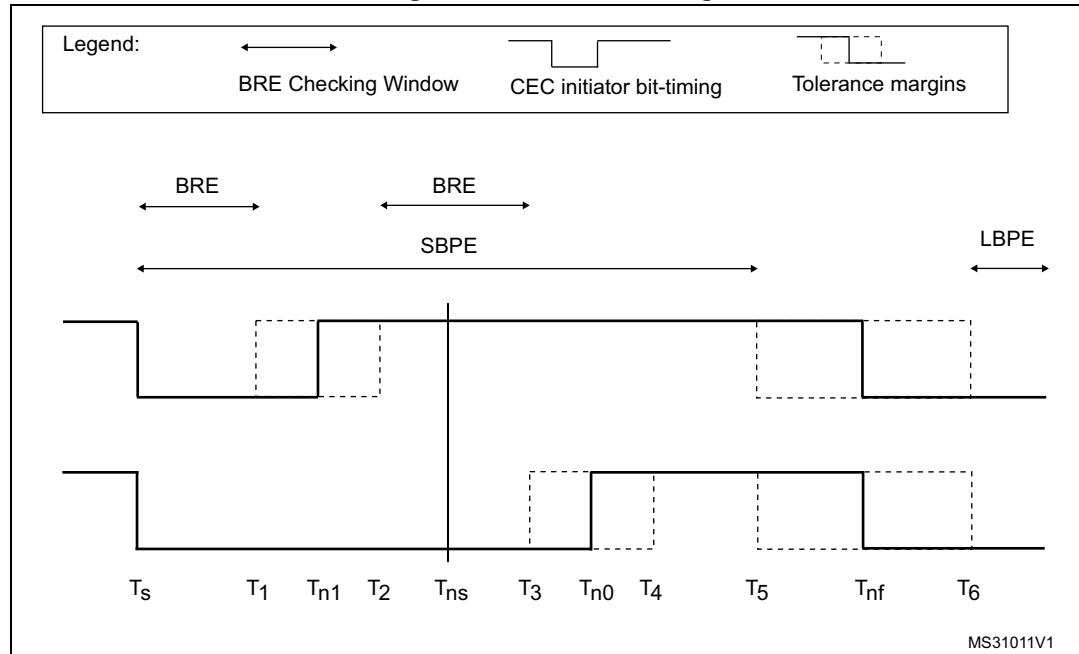
LBPE is set when a bit falling edge is not detected in a valid window (see [Figure 530](#)). LBPE flag also generates a CEC interrupt if the LBPEIE = 1.

LBPE always stops the reception, an error bit is generated on the line when LBPEGEN bit is set.

When LBPE is detected in a broadcast message an error bit is generated even if LBPEGEN = 0 to enforce initiator's retry of the failed transmission. Error bit generation can be disabled by configuring LBPEGEN = 0, BRDNOGEN = 1.

*Note:* The BREGEN = 1, BRESTP = 0 configuration must be avoided.

**Figure 530. Error handling**



**Table 289. Error handling timing parameters**

Time	RXTOL	ms	Description
T <sub>s</sub>	x	0	Bit start event.
T <sub>1</sub>	1	0.3	The earliest time for a low - high transition when indicating a logical 1.
	0	0.4	
T <sub>n1</sub>	x	0.6	The nominal time for a low - high transition when indicating a logical 1.
T <sub>2</sub>	0	0.8	The latest time for a low - high transition when indicating a logical 1.
	1	0.9	
T <sub>ns</sub>	x	1.05	Nominal sampling time.
T <sub>3</sub>	1	1.2	The earliest time a device is permitted return to a high impedance state (logical 0).
	0	1.3	
T <sub>n0</sub>	x	1.5	The nominal time a device is permitted return to a high impedance state (logical 0).
T <sub>4</sub>	0	1.7	The latest time a device is permitted return to a high impedance state (logical 0).
	1	1.8	
T <sub>5</sub>	1	1.85	The earliest time for the start of a following bit.
	0	2.05	
T <sub>nf</sub>	x	2.4	The nominal data bit period.
T <sub>6</sub>	0	2.75	The latest time for the start of a following bit.
	1	2.95	

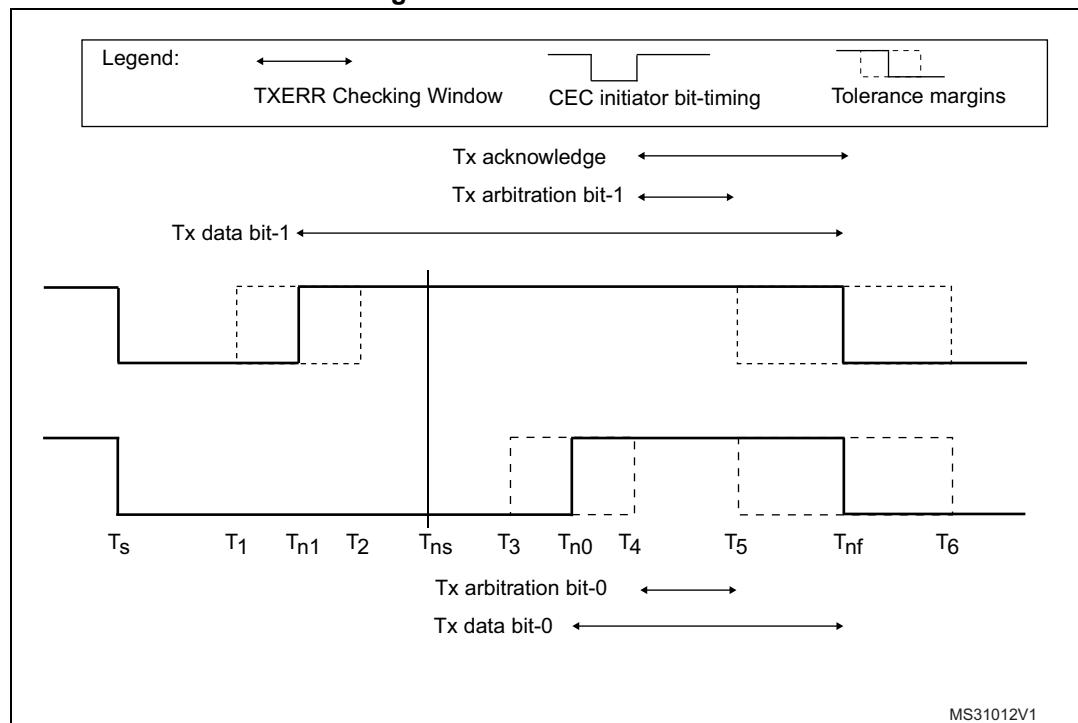
### 39.5.6 Transmission error detection (TXERR)

The CEC initiator sets the TXERR flag if detecting low impedance on the CEC line when it is transmitting high impedance and is not expecting a follower asserted bit. TXERR flag also generates a CEC interrupt if the TXERRIE = 1.

TXERR assertion stops the message transmission. Application is in charge to retry the failed transmission up to five times.

TXERR checks are performed differently depending on the different states of the CEC line and on the RX tolerance configuration.

Figure 531. TXERR detection



MS31012V1

Table 290. TXERR timing parameters

Time	RXTOL	ms	Description
$T_s$	x	0	Bit start event.
$T_1$	1	0.3	The earliest time for a low - high transition when indicating a logical 1.
	0	0.4	
$T_{n1}$	x	0.6	The nominal time for a low - high transition when indicating a logical 1.
$T_2$	0	0.8	The latest time for a low - high transition when indicating a logical 1.
	1	0.9	
$T_{ns}$	x	1.05	Nominal sampling time.
$T_3$	1	1.2	The earliest time a device is permitted return to a high impedance state (logical 0).
	0	1.3	
$T_{n0}$	x	1.5	The nominal time a device is permitted return to a high impedance state (logical 0).
$T_4$	0	1.7	The latest time a device is permitted return to a high impedance state (logical 0).
	1	1.8	
$T_5$	1	1.85	The earliest time for the start of a following bit.
	0	2.05	
$T_{nf}$	x	2.4	The nominal data bit period.

**Table 290. TXERR timing parameters (continued)**

Time	RXTOL	ms	Description
$T_6$	0	2.75	The latest time for the start of a following bit.
	1	2.95	

## 39.6 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 291. 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

## 39.7 HDMI-CEC registers

Refer to [Section 1.2](#) for a list of abbreviations used in register descriptions. The registers have to be accessed by words (32 bits).

### 39.7.1 CEC control register (CEC\_CR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TXEOM	TXSOM	CECEN												
													rs	rs	rw

Bits 31:3 Reserved, must be kept at reset value.

#### Bit 2 TXEOM: Tx end of message

The TXEOM bit is set by software to command transmission of the last byte of a CEC message.

TXEOM is cleared by hardware at the same time and under the same conditions as for TXSOM.

0: TXDR data byte is transmitted with EOM = 0.

1: TXDR data byte is transmitted with EOM = 1.

*Note:* TXEOM must be set when CECEN = 1.

*TXEOM must be set before writing transmission data to TXDR.*

*If TXEOM is set when TXSOM = 0, transmitted message consists of 1 byte (HEADER) only (PING message).*

#### Bit 1 TXSOM: Tx start of message

TXSOM is set by software to command transmission of the first byte of a CEC message. If the CEC message consists of only one byte, TXEOM must be set before 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 starts after the end of reception.

TXSOM is cleared by hardware after the last byte of the message is sent with a positive acknowledge (TXEND = 1), in case of transmission underrun (TXUDR = 1), negative acknowledge (TXACKE = 1), and transmission error (TXERR = 1). It is also cleared by CECEN = 0. It is not cleared and transmission is automatically retried in case of arbitration lost (ARBLST = 1).

TXSOM can be also used as a status bit informing application whether any transmission request is pending or under execution. The application can abort a transmission request at any time by clearing the CECEN bit.

0: No CEC transmission is on-going.

1: CEC transmission command

*Note:* TXSOM must be set when CECEN = 1.

*TXSOM must be set when transmission data is available into TXDR.*

*HEADER first four bits containing own peripheral address are taken from TXDR[7:4], not from CEC\_CFGR.OAR that is used only for reception.*

Bit 0 **CECEN:** CEC enable

The CECEN bit is set and cleared by software. CECEN = 1 starts message reception and enables the TXSOM control. CECEN = 0 disables the CEC peripheral, clears all bits of CEC\_CR register and aborts any on-going reception or transmission.

0: CEC peripheral is off.

1: CEC peripheral is on.

### 39.7.2 CEC configuration register (CEC\_CFGR)

This register is used to configure the HDMI-CEC controller.

Address offset: 0x04

Reset value: 0x0000 0000

**Caution:** It is mandatory to write CEC\_CFGR only when CECEN = 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LSTN	OAR[14:0]														
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	SFTOP	BRDN OGEN	LBPEG EN	BREGE N	BREST P	RXTOL	SFT[2:0]		
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **LSTN:** Listen mode

LSTN bit is set and cleared by software.

0: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.

1: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.

Bits 30:16 **OAR[14:0]:** Own addresses configuration

The OAR bits are set by software to select which destination logical addresses has to be considered in receive mode. Each bit, when set, enables the CEC logical address identified by the given bit position.

At the end of HEADER reception, the received destination address is compared with the enabled addresses. In case of matching address, the incoming message is acknowledged and received. In case of non-matching address, the incoming message is received only in listen mode (LSTN = 1), but without acknowledge sent. Broadcast messages are always received.

Example:

OAR = 0b000 0000 0010 0001 means that CEC acknowledges addresses 0x0 and 0x5.

Consequently, each message directed to one of these addresses is received.

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **SFTOP:** SFT option bit

The SFTOPT bit is set and cleared by software.

0: SFT timer starts when TXSOM is set by software.

1: SFT timer starts automatically at the end of message transmission/reception.

**Bit 7 BRDNOGEN:** Avoid error-bit generation in broadcast

The BRDNOGEN bit is set and cleared by software.

0: BRE detection with BRESTOP = 1 and BREGEN = 0 on a broadcast message generates an error-bit on the CEC line. LBPE detection with LBPEGEN = 0 on a broadcast message generates an error-bit on the CEC line.

1: Error-bit is not generated in the same condition as above. An error-bit is not generated even in case of an SBPE detection in a broadcast message if listen mode is set.

**Bit 6 LBPEGEN:** Generate error-bit on long bit period error

The LBPEGEN bit is set and cleared by software.

0: LBPE detection does not generate an error-bit on the CEC line.

1: LBPE detection generates an error-bit on the CEC line.

*Note: If BRDNOGEN = 0, an error-bit is generated upon LBPE detection in broadcast even if LBPEGEN = 0.*

**Bit 5 BREGEN:** Generate error-bit on bit rising error

The BREGEN bit is set and cleared by software.

0: BRE detection does not generate an error-bit on the CEC line.

1: BRE detection generates an error-bit on the CEC line (if BRESTOP is set).

*Note: If BRDNOGEN = 0, an error-bit is generated upon BRE detection with BRESTOP = 1 in broadcast even if BREGEN = 0.*

**Bit 4 BRESTOP:** Rx-stop on bit rising error

The BRESTOP bit is set and cleared by software.

0: BRE detection does not stop reception of the CEC message. Data bit is sampled at 1.05 ms.

1: BRE detection stops message reception.

**Bit 3 RXTOL:** Rx-tolerance

The RXTOL bit is set and cleared by software.

0: Standard tolerance margin:

- Start-bit, +/- 200 µs rise, +/- 200 µs fall
- Data-bit: +/- 200 µs rise, +/- 350 µs fall

1: Extended tolerance

- Start-bit: +/- 400 µs rise, +/- 400 µs fall
- Data-bit: +/-300 µs rise, +/- 500 µs fall

**Bits 2:0 SFT[2:0]:** Signal free time

SFT bits are set by software. In the SFT = 0x0 configuration, the number of nominal data bit periods waited before transmission is ruled by hardware according to the transmission history. In all the other configurations the SFT number is determined by software.

0x0

- 2.5 data-bit periods if CEC is the last bus initiator with unsuccessful transmission (ARBLST = 1, TXERR = 1, TXUDR = 1 or TXACKE = 1)
- 4 data-bit periods if CEC is the new bus initiator
- 6 data-bit periods if CEC is the last bus initiator with successful transmission (TXEOM = 1)

0x1: 0.5 nominal data bit periods

0x2: 1.5 nominal data bit periods

0x3: 2.5 nominal data bit periods

0x4: 3.5 nominal data bit periods

0x5: 4.5 nominal data bit periods

0x6: 5.5 nominal data bit periods

0x7: 6.5 nominal data bit periods

### 39.7.3 CEC Tx data register (CEC\_TXDR)

Address offset: 0x8

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
TXD[7:0]															
								w	w	w	w	w	w	w	w

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **TXD[7:0]**: Tx data

TXD is a write-only register containing the data byte to be transmitted.

### 39.7.4 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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
RXD[7:0]															
								r	r	r	r	r	r	r	r

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **RXD[7:0]**: Rx data

RXD is read-only and contains the last data byte that has been received from the CEC line.

### 39.7.5 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.	TXACK_E	TXERR	TXUDR	TXEND	TXBR	ARBLS_T	RXACK_E	LBPE	SBPE	BRE	RXOV_R	RXEND	RXBR
			rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:13 Reserved, must be kept at reset value.

**Bit 12 TXACKE:** Tx-missing acknowledge error

In transmission mode, TXACKE is set by hardware to inform application that no acknowledge was received. In case of broadcast transmission, TXACKE informs application that a negative acknowledge was received. TXACKE aborts message transmission and clears TXSOM and TXEOM controls.

TXACKE is cleared by software write at 1.

**Bit 11 TXERR:** Tx-error

In transmission mode, TXERR is set by hardware if the CEC initiator detects low impedance on the CEC line while it is released. TXERR aborts message transmission and clears TXSOM and TXEOM controls.

TXERR is cleared by software write at 1.

**Bit 10 TXUDR:** Tx-buffer underrun

In transmission mode, TXUDR is set by hardware if application was not in time to load TXDR before of next byte transmission. TXUDR aborts message transmission and clears TXSOM and TXEOM control bits.

TXUDR is cleared by software write at 1

**Bit 9 TXEND:** End of transmission

TXEND is set by hardware to inform application that the last byte of the CEC message has been successfully transmitted. TXEND clears the TXSOM and TXEOM control bits.

TXEND is cleared by software write at 1.

**Bit 8 TXBR:** Tx-byte request

TXBR is set by hardware to inform application that the next transmission data has to be written to TXDR. TXBR is set when the 4th bit of currently transmitted byte is sent. Application must write the next byte to TXDR within six nominal data-bit periods before transmission underrun error occurs (TXUDR).

TXBR is cleared by software write at 1.

**Bit 7 ARBLST:** Arbitration lost

ARBLST is set by hardware to inform application that CEC device is switching to reception due to arbitration lost event following the TXSOM command. ARBLST can be due either to a contending CEC device starting earlier or starting at the same time but with higher HEADER priority. After ARBLST assertion TXSOM bit keeps pending for next transmission attempt.

ARBLST is cleared by software write at 1.

**Bit 6 RXACKE:** Rx-missing acknowledge

In receive mode, RXACKE is set by hardware to inform application that no acknowledge was seen on the CEC line. RXACKE applies only for broadcast messages and in listen mode also for not directly addressed messages (destination address not enabled in OAR). RXACKE aborts message reception.

RXACKE is cleared by software write at 1.

**Bit 5 LBPE:** Rx-long bit period error

LBPE is set by hardware in case a data-bit waveform is detected with long bit period error. LBPE is set at the end of the maximum bit-extension tolerance allowed by RXTOL, in case falling edge is still longing. LBPE always stops reception of the CEC message. LBPE generates an error-bit on the CEC line if LBPEGEN = 1. In case of broadcast, error-bit is generated even in case of LBPEGEN = 0.

LBPE is cleared by software write at 1.

**Bit 4 SBPE:** Rx-short bit period error

SBPE is set by hardware in case a data-bit waveform is detected with short bit period error. SBPE is set at the time the anticipated falling edge occurs. SBPE generates an error-bit on the CEC line.

SBPE is cleared by software write at 1.

**Bit 3 BRE:** Rx-bit rising error

BRE is set by hardware in case a data-bit waveform is detected with bit rising error. BRE is set either at the time the misplaced rising edge occurs, or at the end of the maximum BRE tolerance allowed by RXTOL, in case rising edge is still longing. BRE stops message reception if BRESTP = 1. BRE generates an error-bit on the CEC line if BREGEN = 1.

BRE is cleared by software write at 1.

**Bit 2 RXOVR:** Rx-overrun

RXOVR is set by hardware if RXBR is not yet cleared at the time a new byte is received on the CEC line and stored into RXD. RXOVR assertion stops message reception so that no acknowledge is sent. In case of broadcast, a negative acknowledge is sent.

RXOVR is cleared by software write at 1.

**Bit 1 RXEND:** End of reception

RXEND is set by hardware to inform application that the last byte of a CEC message is received from the CEC line and stored into the RXD buffer. RXEND is set at the same time of RXBR.

RXEND is cleared by software write at 1.

**Bit 0 RXBR:** Rx-byte received

The RXBR bit is set by hardware to inform application that a new byte has been received from the CEC line and stored into the RXD buffer.

RXBR is cleared by software write at 1.

### 39.7.6 CEC interrupt enable register (CEC\_IER)

Address offset: 0x14

Reset value: 0x0000 0000

**Caution:** It is mandatory to write CEC\_IER only when CECEN = 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	TXACKIE	TXERRIE	TXUDRIE	TXENDIE	TXBRIE	ARBLSIE	RXACKIE	LBPEIE	SBPEIE	BREIE	RXOVRIE	RXENDIE	RXBRIE
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:13 Reserved, must be kept at reset value.

**Bit 12 TXACKIE:** Tx-missing acknowledge error interrupt enable

The TXACKEIE 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

### **39.7.7 HDMI-CEC register map**

**Table 292. HDMI-CEC register map and reset values**

Refer to [Section 2.2](#) for the register boundary addresses.

## 40 Debug support (DBG)

### 40.1 Overview

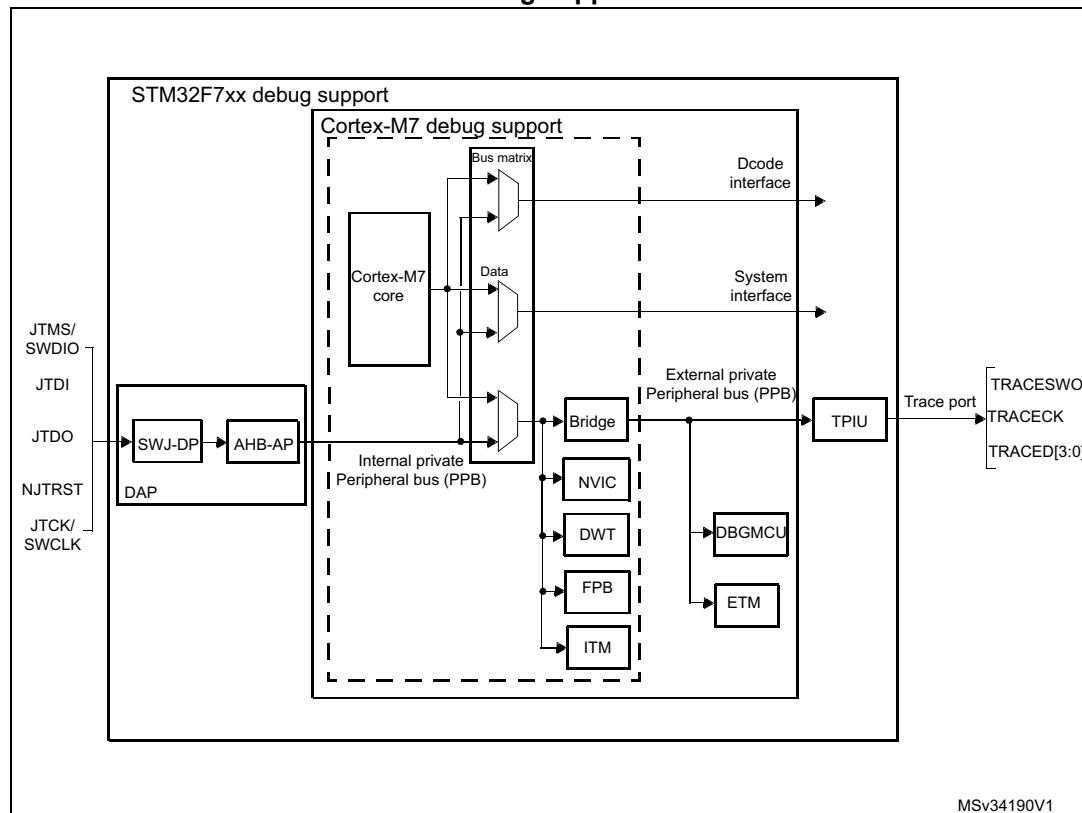
The STM32F75xxx and STM32F74xxx are built around a Cortex<sup>®</sup>-M7 with FPU core which contains hardware extensions for advanced debugging features. The debug extensions allow the core to be stopped either on a given instruction fetch (breakpoint) or data access (watchpoint). When stopped, the core's internal state and the system's external state may be examined. Once examination is complete, the core and the system may be restored and program execution resumed.

The debug features are used by the debugger host when connecting to and debugging the STM32F75xxx and STM32F74xxx MCUs.

Two interfaces for debug are available:

- Serial wire
- JTAG debug port

**Figure 532. Block diagram of STM32 MCU and Cortex<sup>®</sup>-M7 with FPU -level debug support**



MSv34190V1

Note:

*The debug features embedded in the Cortex<sup>®</sup>-M7 with FPU core are a subset of the Arm<sup>®</sup> CoreSight Components Technical Reference Manual.*

The Arm® Cortex®-M7 with FPU core provides integrated on-chip debug support. It is comprised of:

- SWJ-DP: Serial wire / JTAG debug port
- AHP-AP: AHB access port
- ITM: Instrumentation trace macrocell
- FPB: Flash patch breakpoint
- DWT: Data watchpoint trigger
- TPIU: Trace port unit interface (available on larger packages, where the corresponding pins are mapped)
- ETM: Embedded Trace Macrocell (available on larger packages, where the corresponding pins are mapped)

It also includes debug features dedicated to the STM32F75xxx and STM32F74xxx:

- Flexible debug pinout assignment
- MCU debug box (support for low-power modes, control over peripheral clocks, etc.)

**Note:**

*For further information on debug functionality supported by the Arm® Cortex®-M7 with FPU core, refer to the Cortex®-M7 with FPU technical reference manual and to the CoreSight Components Technical Reference Manual (see [Section 40.2: Reference Arm® documentation](#)).*

## 40.2 Reference Arm® documentation

- Cortex®-M7 with FPU technical reference manual (TRM)  
(see Related documents on page 1)
- Arm® Debug Interface V5 architecture specification
- Arm® CoreSight Components Technical Reference Manual

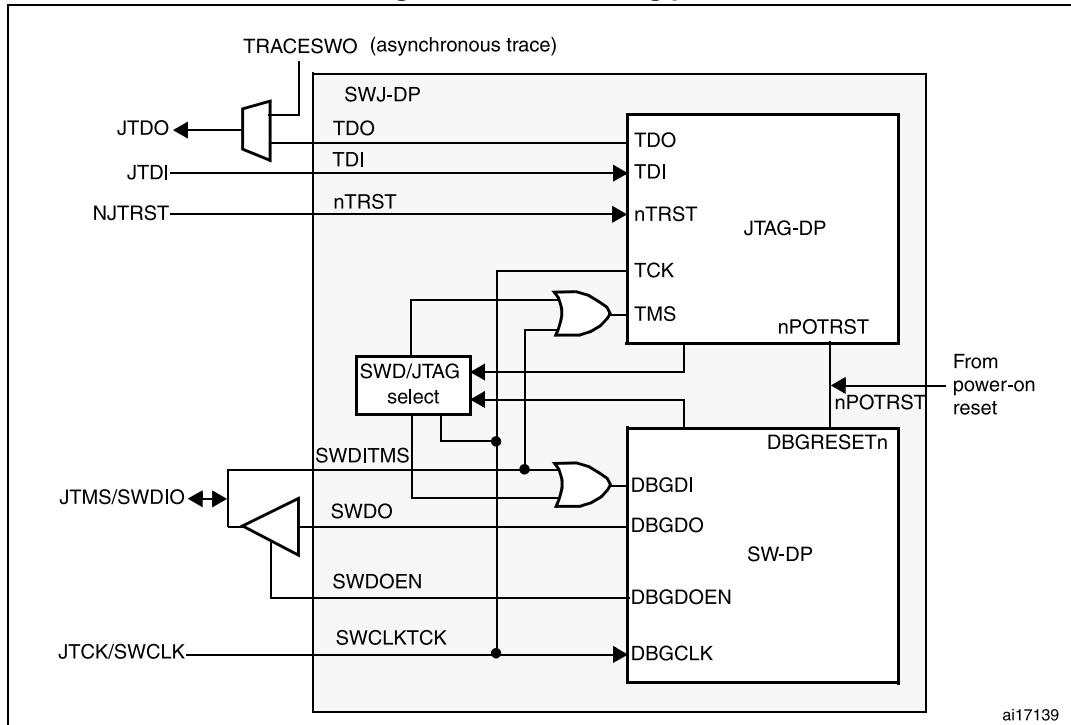
## 40.3 SWJ debug port (serial wire and JTAG)

The core of the STM32F75xxx and STM32F74xxx 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 533. SWJ debug port



*Figure 533* 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.

#### 40.3.1 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

#### 40.4 Pinout and debug port pins

The STM32F75xxx and STM32F74xxx MCUs are available in various packages with different numbers of available pins. As a result, some functionality related to pin availability (TPIU parallel output interface) may differ between packages.

#### 40.4.1 SWJ debug port pins

Five pins are used as outputs from the STM32F75xxx and STM32F74xxx for the SWJ-DP as *alternate functions* of general-purpose I/Os. These pins are available on all packages.

**Table 293. 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

#### 40.4.2 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 STM32F75xxx and STM32F74xxx MCUs offer the possibility of disabling some or all of the SWJ-DP ports and so, of releasing (in gray in the table below) the associated pins for general-purpose IO (GPIO) usage. For more details on how to disable SWJ-DP port pins, refer to [Section 6.3.2: I/O pin alternate function multiplexer and mapping](#).

**Table 294. 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		

#### 40.4.3 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
- JTDO: 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
- JTDO: 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.*

#### 40.4.4 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.*

### 40.5 STM32F75xxx and STM32F74xxx JTAG Debug Port connection

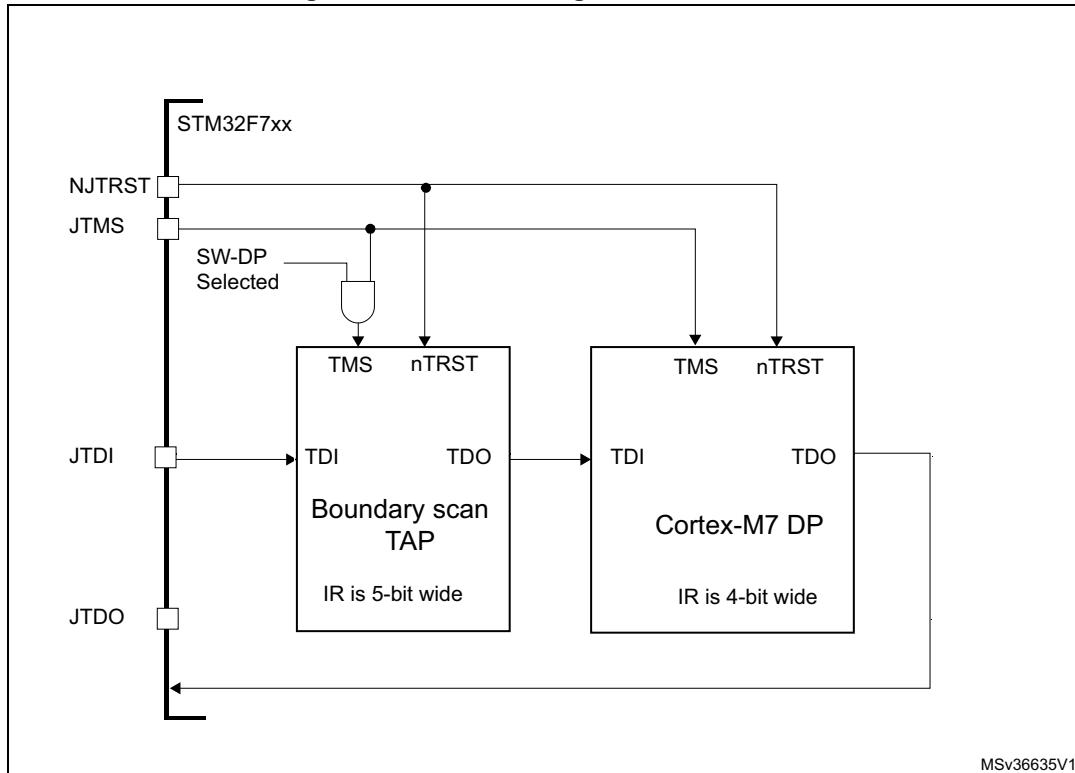
The STM32F75xxx and STM32F74xxx MCUs integrate two serially connected JTAG Debug Ports, the boundary scan Debug Port (IR is 5-bit wide) and the Cortex®-M7 with FPU Debug Port (IR is 4-bit wide).

To access the Debug Port of the Cortex®-M7 with FPU for debug purposes:

1. First, it is necessary to shift the BYPASS instruction of the boundary scan Debug Port.
2. Then, for each IR shift, the scan chain contains 9 bits (=5+4) and the unused Debug Port instruction must be shifted in using the BYPASS instruction.
3. For each data shift, the unused Debug Port, which is in BYPASS mode, adds 1 extra data bit in the data scan chain.

*Note:* **Important:** Once Serial-Wire is selected using the dedicated Arm® JTAG sequence, the boundary scan Debug Port is automatically disabled (JTMS forced high).

Figure 534. JTAG Debug Port connections



## 40.6 ID codes and locking mechanism

There are several ID codes inside the STM32F75xxx and STM32F74xxx 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.

### 40.6.1 MCU device ID code

The STM32F75xxx and STM32F74xxx 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 40.16 on page 1676](#)). This code is accessible using the JTAG debug port (4 to 5 pins) or the SW debug port (two pins) or by the user software. It is even accessible while the MCU is under system reset.

Only the DEV\_ID(11:0) should be used for identification by the debugger/programmer tools.

#### DBGMCU\_IDCODE

Address: 0xE004 2000

Only 32-bits access supported. Read-only.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REV_ID															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	DEV_ID											
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **REV\_ID[15:0]**: Revision identifier

This field indicates the revision of the device:

0x1000 = Revision A

0x1001 = Revision Z and 1

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:0 **DEV\_ID[11:0]**: Device identifier

The device ID is 0x449.

### 40.6.2 Boundary scan Debug Port

#### JTAG ID code

The Debug Port of the STM32F75xxx and STM32F74xxx BSC (boundary scan) integrates a JTAG ID code equal to 0x06449041.

### 40.6.3 Cortex®-M7 with FPU Debug Port

The Debug Port of the Arm® Cortex®-M7 with FPU integrates a JTAG ID code. This ID code is the Arm® default one and has not been modified. This code is only accessible by the JTAG Debug Port.

This code is 0x5BA00477 (corresponds to Cortex®-M7 with FPU, see [Section 40.2: Reference Arm® documentation](#)).

#### 40.6.4 Cortex®-M7 with FPU JEDEC-106 ID code

The Arm® Cortex®-M7 with FPU integrates a JEDEC-106 ID code. It is located in the 4KB ROM table mapped on the internal PPB bus at address 0xE00FF000\_0xE00FFFFF.

This code is accessible by the JTAG Debug Port (4 to 5 pins) or by the SW Debug Port (two pins) or by the user software.

### 40.7 JTAG debug port

A standard JTAG state machine is implemented with a 4-bit instruction register (IR) and five data registers (for full details, refer to the Cortex®-M7 with FPU technical *reference manual (TRM)*, for references, see [Section 40.2: Reference Arm® documentation](#)).

**Table 295. JTAG debug port data registers**

IR(3:0)	Data register	Details
1111	BYPASS [1 bit]	-
1110	IDCODE [32 bits]	ID CODE 0x06449041 (Arm® Cortex®-M7 with FPU ID Code)
1010	DPACC [35 bits]	Debug port access register This initiates a debug port and allows access to a debug port register. – When transferring data IN: Bits 34:3 = DATA[31:0] = 32-bit data to transfer for a write request Bits 2:1 = A[3:2] = 2-bit address of a debug port register. Bit 0 = RnW = Read request (1) or write request (0). – When transferring data OUT: Bits 34:3 = DATA[31:0] = 32-bit data which is read following a read request Bits 2:0 = ACK[2:0] = 3-bit Acknowledge: 010 = OK/FAULT 001 = WAIT OTHER = reserved Refer to <a href="#">Table 296</a> for a description of the A(3:2) bits

**Table 295. 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 296. 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)

## 40.8 SW debug port

### 40.8.1 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.

### 40.8.2 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 297. 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 296</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®-M7 with FPU *TRM* for a detailed description of DPACC and APACC registers.

The packet request is always followed by the turnaround time (default 1 bit) where neither the host nor target drive the line.

**Table 298. 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 299. 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.

#### 40.8.3 SW-DP state machine (reset, idle states, ID code)

The State Machine of the SW-DP has an internal ID code which identifies the SW-DP. It follows the JEP-106 standard. This ID code is the default Arm® one and is set to **0x5BA02477** (corresponding to Cortex®-M7 with FPU).

- Note:** Note that the SW-DP state machine is inactive until the target reads this ID code.
- The SW-DP state machine is in RESET STATE either after power-on reset, or after the DP has switched from JTAG to SWD or after the line is high for more than 50 cycles
  - The SW-DP state machine is in IDLE STATE if the line is low for at least two cycles after RESET state.
  - After RESET state, it is **mandatory** to first enter into an IDLE state AND to perform a READ access of the DP-SW ID CODE register. Otherwise, the target will issue a FAULT acknowledge response on another transactions.

Further details of the SW-DP state machine can be found in the *Cortex®-M7 with FPU TRM* and the *CoreSight Components Technical Reference Manual*.

#### 40.8.4 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.

#### 40.8.5 SW-DP registers

Access to these registers are initiated when APnDP=0

Table 300. SW-DP registers

A(3:2)	R/W	CTRLSEL bit of SELECT register	Register	Notes
00	Read	-	IDCODE	The manufacturer code is not set to ST code. 0x5BA02477 (identifies the SW-DP)
00	Write	-	ABORT	-

**Table 300. 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

#### 40.8.6 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

## 40.9 AHB-AP (AHB access port) - valid for both JTAG-DP and SW-DP

### Features:

- System access is independent of the processor status.
- Either SW-DP or JTAG-DP accesses AHB-AP.
- The AHB-AP is an AHB master into the Bus Matrix. Consequently, it can access all the data buses (Dcode Bus, System Bus, internal and external PPB bus) but the ICode bus.
- Bitband transactions are supported.
- AHB-AP transactions bypass the FPB.

The address of the 32-bits AHP-AP registers are 6-bits wide (up to 64 words or 256 bytes) and consists of:

- c) Bits [7:4] = the bits [7:4] APBANKSEL of the DP SELECT register
- d) Bits [3:2] = the 2 address bits of A(3:2) of the 35-bit packet request for SW-DP.

The AHB-AP of the Cortex®-M7 with FPU includes 9 x 32-bits registers:

**Table 301. Cortex®-M7 with FPU AHB-AP registers**

Address offset	Register name	Notes
0x00	AHB-AP Control and Status Word	Configures and controls transfers through the AHB interface (size, hprot, status on current transfer, address increment type)
0x04	AHB-AP Transfer Address	-
0x0C	AHB-AP Data Read/Write	-
0x10	AHB-AP Banked Data 0	
0x14	AHB-AP Banked Data 1	
0x18	AHB-AP Banked Data 2	
0x1C	AHB-AP Banked Data 3	
0xF8	AHB-AP Debug ROM Address	Base Address of the debug interface
0xFC	AHB-AP ID Register	-

Refer to the *Cortex®-M7 with FPU TRM* for further details.

## 40.10 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 302. 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®-M7 with FPU TRM* for further details.

To Halt on reset, it is necessary to:

- enable the bit0 (VC\_CORRESET) of the Debug and Exception Monitor Control Register
- enable the bit0 (C\_DEBUGEN) of the Debug Halting Control and Status Register.

## 40.11 Capability of the debugger host to connect under system reset

The reset system of the STM32F75xxx and STM32F74xxx MCU comprises the following reset sources:

- POR (power-on reset) which asserts a RESET at each power-up.
- Internal watchdog reset
- Software reset
- External reset

The Cortex®-M7 with FPU differentiates the reset of the debug part (generally PORRESETn) and the other one (SYSRESETn)

This way, it is possible for the debugger to connect under System Reset, programming the Core Debug Registers to halt the core when fetching the reset vector. Then the host can release the system reset and the core will immediately halt without having executed any instructions. In addition, it is possible to program any debug features under System Reset.

*Note:* *It is highly recommended for the debugger host to connect (set a breakpoint in the reset vector) under system reset.*

## 40.12 FPB (Flash patch breakpoint)

Typically in Cortex-M architecture the FPB unit allows to:

- implement hardware breakpoints
- patch code and data from code space to system space. This feature gives the possibility to correct software bugs located in the Code Memory Space.

Where the use of a Software Patch or a Hardware Breakpoint is exclusive.

But there are some major changes in Pelican FPB:

- Flash patching is no more supported (there is no FP\_REMAP register)
- All comparators are for instruction addresses (up to 8 instruction breakpoints)
- Programmer's model for breakpoint comparators is enhanced to allow hardware breakpoint in full address range.

## 40.13 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

## 40.14 ITM (instrumentation trace macrocell)

### 40.14.1 General description

The ITM is an application-driven trace source that supports *printf* style debugging to trace *Operating System* (OS) and application events, and emits diagnostic system information. The ITM emits trace information as packets which can be generated as:

- **Software trace.** Software can write directly to the ITM stimulus registers to emit packets.
- **Hardware trace.** The DWT generates these packets, and the ITM emits them.
- **Time stamping.** Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp. The Cortex®-M7 with FPU clock or the bit clock rate of the *Serial Wire Viewer* (SWV) output clocks the counter.

The packets emitted by the ITM are output to the TPIU (Trace Port Interface Unit). The formatter of the TPIU adds some extra packets (refer to TPIU) and then output the complete packets sequence to the debugger host.

The bit TRCEN of the Debug Exception and Monitor Control Register must be enabled before you program or use the ITM.

### 40.14.2 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 303. 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 40.17.2: TRACE pin assignment](#) and [Section 40.16.3: 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)

## 40.15 ETM (Embedded trace macrocell)

### 40.15.1 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 four 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 40.13: 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 40.17: Pelican TPIU \(trace port interface unit\)](#)) and then outputs the complete packet sequence to the debugger host.

*Note:* *N.B: Cortex-M7 ETM is compliant with Arm ETM architecture v4, which programming model is not backward compatible with Cortex-M4 ETM one (ETM architecture v3.5).*

### 40.15.2 Signal protocol, packet types

This part is described in the chapter 6 ETM v4 architecture specification (IHI0064B).

### 40.15.3 Main ETM registers

For more information on registers refer to the Pelican ETM technical reference manual (DDI0494-2a) and the ETM v4 architecture specification (IHI0064B).

### 40.15.4 Configuration example

To output a simple value to the TPIU:

- Configure trace I/Os: enable TRACE\_CLKINEN in the STM32F75xxx and STM32F74xxx debug configuration register (DBGMCU\_CR).
- Write @ E000EDFC 01000000; SCS: set TRCENA, otherwise trace registers are not accessible.
- Write @ E00400F0 00000000; TPIU: select SYNC PORT Mode
- Write @ E0040004 00000008; TPIU: select TPIU PORT SIZE=4
- Write @ E0001020 002002CA; WT: PC MATCH Comparator (PC=0x2002CA)
- Write @ E0001024 00000000; DWT: No mask apply on comparator
- Write @ E0001028 00000008; DWT: ETM trig on PC on match

ETM:

- Write @ E0041004 00000000; Disable ETM
- Read @ E004100C 00000003; ETM should be in Idle state
- Write @ E0041040 00000002; Instruction trace source ID = 0x2
- Write @ E0041080 00000001; Resource for ViewInst enabling event is “always TRUE”
- Write @ E004108C 000000FF; Processor comparator selection for Start: pc\_match0 (=>DWT match)
- Write @ E0041004 00000001; Enable ETM

## 40.16 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

### 40.16.1 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.

#### 40.16.2 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.

For timers having complementary outputs, when the counter is stopped (DBG\_TIMx\_STOP=1), the outputs are disabled (as if the MOE bit was reset) for safety purposes.

#### 40.16.3 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.

#### 40.16.4 DBGMCU\_CR register

Address: 0xE004 2004

Only 32-bit access supported

POR Reset: 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TRACE_MODE [1:0]	TRACE_CLKINEN	Res.	Res.	DBG_STAND BY	DBG_STOP	DBG_SLEEP								
								rw	rw	rw		rw	rw	rw	

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:5 **TRACE\_MODE[1:0]** and **TRACE\_CLKINEN**: Trace clock and pin assignment control

- With TRACE\_CLKINEN=0:  
TRACE\_MODE=xx: TRACE output disabled (both synchronous and asynchronous)
- With TRACE\_CLKINEN=1:
  - TRACE\_MODE[1:0]=00: Asynchronous trace interface enabled at pad level (TRACESWO available only on TDO pad when using Serial Wire mode) / Synchronous trace interface enabled
  - TRACE\_MODE[1:0] different to =00: Asynchronous trace interface disabled / Synchronous trace interface enabled

Bits 4:3 Reserved, must be kept at reset value.

**Bit 2 **DBG\_STANDBY**: Debug Standby mode**

0: (FCLK=Off, HCLK=Off) The whole digital part is unpowered.

From software point of view, exiting from Standby is identical than fetching reset vector (except a few status bit indicated that the MCU is resuming from Standby)

1: (FCLK=On, HCLK=On) In this case, the digital part is not unpowered and FCLK and HCLK are provided by the internal RC oscillator which remains active. In addition, the MCU generate a system reset during Standby mode so that exiting from Standby is identical than fetching from reset

**Bit 1 **DBG\_STOP**: Debug Stop mode**

0: (FCLK=Off, HCLK=Off) In STOP mode, the clock controller disables all clocks (including HCLK and FCLK). When exiting from STOP mode, the clock configuration is identical to the one after RESET (CPU clocked by the 8 MHz internal RC oscillator (HSI)). Consequently, the software must reprogram the clock controller to enable the PLL, the Xtal, etc.

1: (FCLK=On, HCLK=On) In this case, when entering STOP mode, FCLK and HCLK are provided by the internal RC oscillator which remains active in STOP mode. When exiting STOP mode, the software must reprogram the clock controller to enable the PLL, the Xtal, etc. (in the same way it would do in case of **DBG\_STOP=0**)

**Bit 0 **DBG\_SLEEP**: Debug Sleep mode**

0: (FCLK=On, HCLK=Off) In Sleep mode, FCLK is clocked by the system clock as previously configured by the software while HCLK is disabled.

In Sleep mode, the clock controller configuration is not reset and remains in the previously programmed state. Consequently, when exiting from Sleep mode, the software does not need to reconfigure the clock controller.

1: (FCLK=On, HCLK=On) In this case, when entering Sleep mode, HCLK is fed by the same clock that is provided to FCLK (system clock as previously configured by the software).

#### 40.16.5 Debug MCU APB1 freeze register (**DBGMCU\_APB1\_FZ**)

The **DBGMCU\_APB1\_FZ** register is used to configure the MCU under DEBUG. It concerns APB2 peripherals. It is mapped on the external PPB bus at address 0xE004 2008

It is asynchronously reset by the POR (and not the system reset). It can be written by the debugger under system reset.

Address: 0xE004 2008

Only 32-bit access is supported.

Power-on-reset (POR): 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	DBG_CAN2_STOP	DBG_CAN1_STOP	DBG_I2C4_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	DBG_LPTIM1_STOP	DBG_TIM14_STOP	DBG_TIM13_STOP	DBG_TIM12_STOP	DBG_TIM7_STOP	DBG_TIM6_STOP	DBG_TIM5_STOP	DBG_TIM4_STOP	DBG_TIM3_STOP	DBG_TIM2_STOP
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:27 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\_I2C4\_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 **DBG\_LPTIM1\_STOP:** LPTIM1 counter stopped when core is halted  
 0: The clock of LPTIM1 counter is fed even if the core is halted  
 1: The clock of LPTIM1 counter is stopped when the core is halted
- Bits 8:0 **DBG\_TIMx\_STOP:** TIMx counter stopped when core is halted ( $x=2..7, 12..14$ )  
 0: The clock of the involved Timer Counter is fed even if the core is halted  
 1: The clock of the involved Timer counter is stopped and the outputs are disabled when the core is halted

#### 40.16.6 Debug MCU APB2 Freeze register (DBGMCU\_APB2\_FZ)

The DBGMCU\_APB2\_FZ register is used to configure the MCU under Debug. It concerns APB2 peripherals.

This register is mapped on the external PPB bus at address 0xE004 200C

It is asynchronously reset by the POR (and not the system reset). It can be written by the debugger under system reset.

Address: 0xE004 200C

Only 32-bit access is supported.

POR: 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	DBG_TIM11_STOP	DBG_TIM10_STOP	DBG_TIM9_STOP												
													rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DBG_TIM8_STOP	DBG_TIM1_STOP													
													rw	rw	

Bits 31:19 Reserved, must be kept at reset value.

- Bits 18:16 **DBG\_TIMx\_STOP:** TIMx counter stopped when core is halted ( $x=9..11$ )  
 0: The clock of the involved Timer Counter is fed even if the core is halted  
 1: The clock of the involved Timer counter is stopped and the outputs are disabled 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 and the output is disabled 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 and the output is disabled when the core is halted

## 40.17 Pelican TPIU (trace port interface unit)

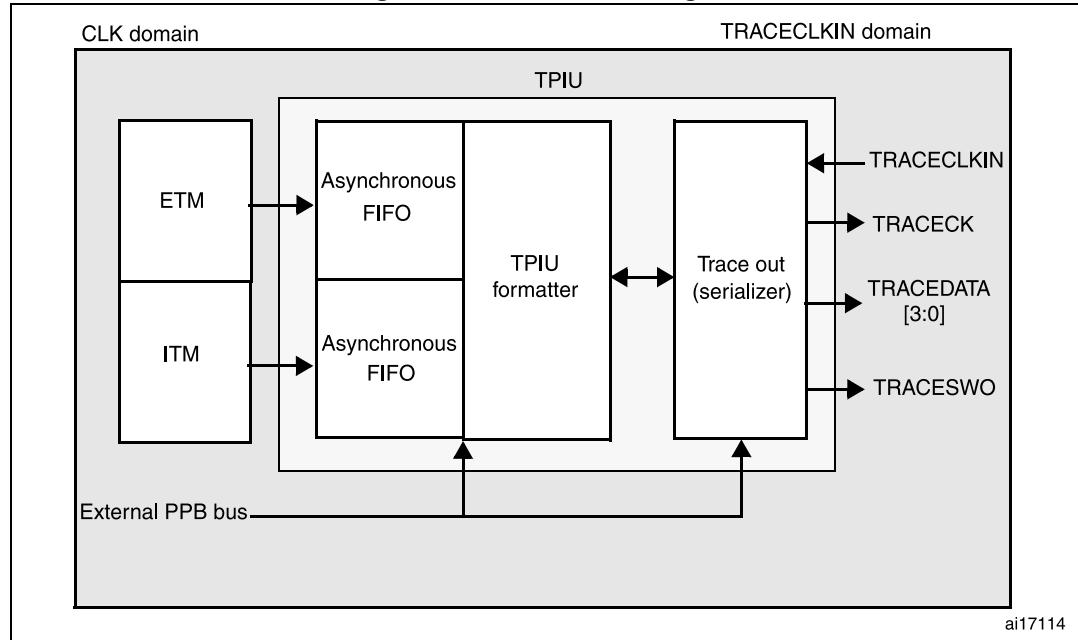
### 40.17.1 Introduction

The TPIU acts as a bridge between the on-chip trace data from the ITM, the ETM and the external trace capture device.

The output data stream encapsulates the trace source ID, that is then captured by a *trace port analyzer* (TPA).

The core embeds a simple TPIU, especially designed for low-cost debug (consisting of a special version of the CoreSight TPIU).

Figure 535. TPIU block diagram



### 40.17.2 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 304. Asynchronous TRACE pin assignment**

TPIU pin name	Trace synchronous mode	
	Type	Description
TRACESWO	O	TRACE Async Data Output

- Synchronous mode

The synchronous mode requires from 2 to 6 extra pins depending on the data trace size and is only available in the larger packages. In addition it is available in JTAG mode and in Serial Wire mode and provides better bandwidth output capabilities than asynchronous trace.

**Table 305. Synchronous TRACE pin assignment**

TPIU pin name	Trace synchronous mode	
	Type	Description
TRACECK	O	TRACE Clock
TRACED[3:0]	O	TRACE Sync Data Outputs Can be 1, 2 or 4.

#### TPIU TRACE pin assignment

By default, these pins are NOT assigned. They can be assigned by setting the TRACE\_CLKINEN and TRACE\_MODE bits in the **MCU Debug component configuration register**. This configuration has to be done by the debugger host.

In addition, the number of pins to assign depends on the trace configuration (asynchronous or synchronous).

- Asynchronous mode:** 1 extra pin is needed
- Synchronous mode:** from 2 to 5 extra pins are needed depending on the size of the data trace port register (1, 2 or 4):
  - TRACECK
  - TRACED(0) if port size is configured to 1, 2 or 4
  - TRACED(1) if port size is configured to 2 or 4
  - TRACED(2) if port size is configured to 4
  - TRACED(3) if port size is configured to 4

To assign the TRACE pin, the debugger host must program the bits TRACE\_CLKINEN and TRACE\_MODE[1:0] of the Debug MCU configuration register (DBGMCU\_CR). By default the TRACE pins are not assigned.

This register is mapped on the external PPB and is reset by the PORESET (and not by the SYSTEM reset). It can be written by the debugger under SYSTEM reset.

**Table 306. Flexible TRACE pin assignment**

DBGMCU_CR register		Pins assigned for:	TRACE IO pin assigned					
			JTDO/ TRACESWO	TRACECK	TRACED[0]	TRACED[1]	TRACED[2]	TRACED[3]
0	XX	No Trace (default state)	Released <sup>(1)</sup>			-		
1	00	Asynchronous Trace	TRACESWO	-	-	Released (usable as GPIO)		
1	different of 00	Synchronous Trace 1 bit <sup>(2)</sup>	Released <sup>(1)</sup>	TRACECK	TRACED[0]	-	-	-
1	different of 00	Synchronous Trace 2 bit <sup>(2)</sup>		TRACECK	TRACED[0]	TRACED[1]	-	-
1	different of 00	Synchronous Trace 4 bit <sup>(2)</sup>		TRACECK	TRACED[0]	TRACED[1]	TRACED[2]	TRACED[3]

1. When Serial Wire mode is used, it is released. But when JTAG is used, it is assigned to JTDO.

2. Selected with Bit[3:0] Current port size from TPIU register.

**Note:** By default, the TRACECLKIN input clock of the TPIU is tied to GND. It is assigned to HCLK two clock cycles after the bit TRACE\_CLKINEN has been set.

The debugger must then program the Trace Mode by writing the PROTOCOL[1:0] bits in the SPP\_R (Selected Pin Protocol) register of the TPIU.

- PROTOCOL=00: Trace Port Mode (synchronous)
- PROTOCOL=01 or 10: Serial Wire (Manchester or NRZ) Mode (asynchronous mode). Default state is 01

It then also configures the TRACE port size by writing the bits [3:0] in the CPSPS\_R (Current Sync Port Size Register) of the TPIU:

- 0x1 for 1 pin (default state)
- 0x2 for 2 pins
- 0x8 for 4 pins

### 40.17.3 TPIU formatter

The purpose of this formatter is to build 128 bit frames containing trace data from, potentially, both the ETM and the ITM, and to allow at a trace analyzer level a correlation between trace packets and emitters.

The formatter protocol outputs data in 16-byte frames:

- seven bytes of data
- eight bytes of mixed-use bytes consisting of:
  - 1 bit (LSB) to indicate it is a DATA byte ('0) or an ID byte ('1).
  - 7 bits (MSB) which can be data or change of source ID trace.
- one byte of auxiliary bits where each bit corresponds to one of the eight mixed-use bytes:
  - if the corresponding byte was a data, this bit gives bit0 of the data.
  - if the corresponding byte was an ID change, this bit indicates when that ID change takes effect.

**Note:** Refer to the Arm® CoreSight Architecture Specification v1.0 (Arm® IHI 0029B) for further information

### 40.17.4 TPIU frame synchronization packets

The TPIU can generate two types of synchronization packets:

- The Frame Synchronization packet (or Full Word Synchronization packet)  
It consists of the word: 0x7F\_FF\_FF\_FF (LSB emitted first). This sequence can not occur at any other time provided that the ID source code 0x7F has not been used.  
It is output periodically **between** frames.  
In continuous mode, the TPA must discard all these frames once a synchronization frame has been found.
- The Half-Word Synchronization packet  
It consists of the half word: 0x7F\_FF (LSB emitted first).  
It is output periodically **between or within** frames.  
These packets are only generated in continuous mode and enable the TPA to detect that the TRACE port is in IDLE mode (no TRACE to be captured). When detected by the TPA, it must be discarded.

### 40.17.5 Transmission of the synchronization frame packet

There is no Synchronization Counter register implemented in the TPIU of the core. Consequently, the synchronization trigger can only be generated by the **DWT**. Refer to the registers DWT Control Register (bits SYNCTAP[11:10]) and the DWT Current PC Sampler Cycle Count Register.

The TPIU Frame synchronization packet (0x7F\_FF\_FF\_FF) is emitted:

- after each TPIU reset release. This reset is synchronously released with the rising edge of the TRACECLKIN clock. This means that this packet is transmitted when the

TRACE\_CLKINEN bit in the DBGMCU\_CFG register is set. In this case, the word 0x7F\_FF\_FF\_FF is not followed by any formatted packet.

- at each DWT trigger (assuming DWT has been previously configured). Two cases occur:
  - If the bit SYNENA of the ITM is reset, only the word 0x7F\_FF\_FF\_FF is emitted without any formatted stream which follows.
  - If the bit SYNENA of the ITM is set, then the ITM synchronization packets will follow (0x80\_00\_00\_00\_00\_00), formatted by the TPIU (trace source ID added).

#### 40.17.6 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.

#### 40.17.7 Asynchronous mode

This is a low cost alternative to output the trace using only 1 pin: this is the asynchronous output pin TRACESWO. Obviously there is a limited bandwidth.

Single IO trace mode is typically suitable for ITM trace output. Also, formatter is disabled in case of asynchronous trace, so merging of ETM and ITM trace streams is not possible.

TRACESWO is multiplexed with JTDO when using the SW-DP pin. This way, this functionality is available in all STM32F75xxx and STM32F74xxx 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%.

#### 40.17.8 TRACECLKIN connection inside the STM32F75xxx and STM32F74xxx

In the STM32F75xxx and STM32F74xxx, 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 STM32F75xxx and STM32F74xxx MCUs is the internal RC oscillator. Its frequency under reset is different from the one after reset release. This is because the RC calibration is the default one under system reset and is updated at each system reset release.*

*Consequently, the trace port analyzer (TPA) should not enable the trace (with the TRACE\_CLKINEN bit) under system reset, because a Synchronization Frame Packet will be issued with a different bit time than trace packets which will be transmitted after reset release.*

### 40.17.9 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 307. Important TPIU registers**

Address	Register	Description
0xE0040004	Current port size	Allows the trace port size to be selected: Bit 0: Port size = 1 Bit 1: Port size = 2 Bit 2: Port size = 3, not supported Bit 3: Port Size = 4 Only 1 bit must be set. By default, the port size is one bit. (0x00000001)
0xE00400F0	Selected pin protocol	Allows the Trace Port Protocol to be selected: Bit1:0= 00: Sync Trace Port Mode 01: Serial Wire Output - manchester (default value) 10: Serial Wire Output - NRZ 11: reserved
0xE0040304	Formatter and flush control	Bits 31-9 = always '0 Bit 8 = TrigIn = always '1 to indicate that triggers are indicated Bits 7-4 = always 0 Bits 3-2 = always 0 Bit 1 = EnFCont. In Sync Trace mode (Select_Pin_Protocol register bit1:0=00), this bit is forced to '1: the formatter is automatically enabled in continuous mode. In asynchronous mode (Select_Pin_Protocol register bit1:0 <> 00), this bit can be written to activate or not the formatter. Bit 0 = always 0 The resulting default value is 0x102 <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).
0xE0040300	Formatter and flush status	Not used in Cortex®-M7 with FPU, always read as 0x00000008

### 40.17.10 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

## 40.18 DBG register map

The following table summarizes the Debug registers.

**Table 308. DBG register map and reset values**

Addr.	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0xE004 2000	<b>DBGMCU_IDCODE</b>	Res.																																	
0xE004 2004	<b>DBGMCU_CR</b>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
0xE004 2008	<b>DBGMCU_APB1_FZ</b>	Res.																																	
0xE004 200C	<b>DBGMCU_APB2_FZ</b>	Res.																																	
	Reset value <sup>(1)</sup>																																		
	Reset value																																		
	Reset value																																		
	Reset value																																		

1. The reset value is product dependent. For more information, refer to [Section 40.6.1: MCU device ID code](#).

Refer to [Section 2.2 on page 69](#) for the register boundary addresses.

## 41 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 STM32F75xxx and STM32F74xxx microcontrollers.

### 41.1 Unique device ID register (96 bits)

The unique device identifier is ideally suited:

- for use as serial numbers (for example USB string serial numbers or other end applications)
- for use as security keys in order to increase the security of code in Flash memory while using and combining this unique ID with software cryptographic primitives and protocols before programming the internal Flash memory
- to activate secure boot processes, etc.

The 96-bit unique device identifier provides a reference number which is unique for any device and in any context. These bits can never be altered by the user.

The 96-bit unique device identifier can also be read in single bytes/half-words/words in different ways and then be concatenated using a custom algorithm.

**Base address: 0x1FF0 F420**

Address offset: 0x00

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UID[31:0]																															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bits 31:0 **UID[31:0]**: X and Y coordinates on the wafer

Address offset: 0x04

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UID[63:48]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UID[47:32]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:8 **UID[63:40]**: LOT\_NUM[23:0]

Lot number (ASCII encoded).

Bots 7:0 **UID[39:32]**: WAF\_NUM[7:0]

Wafer number (8-bit unsigned number).

Address offset: 0x08

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UID[95:80]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UID[79:64]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **UID[95:64]**: LOT\_NUM[55:24]

Lot number (ASCII encoded).

## 41.2 Flash size

Base address: 0x1FF0 F442

Address offset: 0x00

Read only = 0xXXXX where X is factory-programmed

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F_SIZE															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 F\_ID(15:0): Flash memory size

This bitfield indicates the size of the device Flash memory expressed in Kbytes.

As an example, 0x0400 corresponds to 1024 Kbytes.

### 41.3 Package data register

Base address: 0x1FF0 F7E0

Address offset: 0x00

Read only = 0xXXXX where X is factory-programmed

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	PKG[2:0]			Res.							
					rw	rw	rw								

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:8 PKG[2:0]: Package type

0x1xx: LQFP208 and TFBGA216 package

0x011: LQFP176 and UFBGA176 package

0x010: WLCSP143 and LQFP144 package

0x001: LQFP100 package

0x000: Reserved

Bits 7:0 Reserved, must be kept at reset value.

## 42 Important security notice

The STMicroelectronics group of companies (ST) places a high value on product security, which is why the ST product(s) identified in this documentation may be certified by various security certification bodies and/or may implement our own security measures as set forth herein. However, no level of security certification and/or built-in security measures can guarantee that ST products are resistant to all forms of attacks. As such, it is the responsibility of each of ST's customers to determine if the level of security provided in an ST product meets the customer needs both in relation to the ST product alone, as well as when combined with other components and/or software for the customer end product or application. In particular, take note that:

- ST products may have been certified by one or more security certification bodies, such as Platform Security Architecture ([www.psacertified.org](http://www.psacertified.org)) and/or Security Evaluation standard for IoT Platforms ([www.trustcb.com](http://www.trustcb.com)). For details concerning whether the ST product(s) referenced herein have received security certification along with the level and current status of such certification, either visit the relevant certification standards website or go to the relevant product page on [www.st.com](http://www.st.com) for the most up to date information. As the status and/or level of security certification for an ST product can change from time to time, customers should re-check security certification status/level as needed. If an ST product is not shown to be certified under a particular security standard, customers should not assume it is certified.
- Certification bodies have the right to evaluate, grant and revoke security certification in relation to ST products. These certification bodies are therefore independently responsible for granting or revoking security certification for an ST product, and ST does not take any responsibility for mistakes, evaluations, assessments, testing, or other activity carried out by the certification body with respect to any ST product.
- Industry-based cryptographic algorithms (such as AES, DES, or MD5) and other open standard technologies which may be used in conjunction with an ST product are based on standards which were not developed by ST. ST does not take responsibility for any flaws in such cryptographic algorithms or open technologies or for any methods which have been or may be developed to bypass, decrypt or crack such algorithms or technologies.
- While robust security testing may be done, no level of certification can absolutely guarantee protections against all attacks, including, for example, against advanced attacks which have not been tested for, against new or unidentified forms of attack, or against any form of attack when using an ST product outside of its specification or intended use, or in conjunction with other components or software which are used by customer to create their end product or application. ST is not responsible for resistance against such attacks. As such, regardless of the incorporated security features and/or any information or support that may be provided by ST, each customer is solely responsible for determining if the level of attacks tested for meets their needs, both in relation to the ST product alone and when incorporated into a customer end product or application.
- All security features of ST products (inclusive of any hardware, software, documentation, and the like), including but not limited to any enhanced security features added by ST, are provided on an "AS IS" BASIS. AS SUCH, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ST DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, unless the applicable written and signed contract terms specifically provide otherwise.

## 43 Revision history

**Table 309. Document revision history**

Date	Revision	Changes
21-May-2015	1	<p>Initial release.</p>
21-Jul-2015	2	<p><b>Updated FMC Section:</b>            Updated <i>Table 83: Programmable NAND flash access parameters</i> memory setup time, memory hold max value.            Updated c7amba_fmc_V2_UserSpec:            - Updated <i>Section : Common memory space timing register (FMC_PMEM)</i> MEMSEY, MEMHOLD, MEMHIZ registers.            - Updated <i>Section : Attribute memory space timing register (FMC_PATT)</i> ATTSET, ATTHOLD, ATTHIZ register.            - Updated <i>Table 89: FMC register map and reset values</i> and the bitfield description adding [x:x].            and FMC_SDCR2 Bit 13 to bit 31 reserved.            - Updated <i>Section : SRAM/NOR-flash chip-select control register for bank x (FMC_BCRx)</i> CPSIZE[2:0] bits description adding 011 configuration.</p> <p><b>Updated SYSARCHI section:</b>            - Updated <i>Section 2.1.8: DMA memory bus</i> and <i>Section 2.1.9: DMA peripheral bus</i> adding ‘internal Flash memory’.            - Updated <i>Section 2.1.6: CPU AHBS bus</i>.</p> <p><b>Updated Flash memory section:</b>            - Updated <i>Section 3.3.1: Flash memory organization</i>.            - Updated <i>Section 3.3.8: Flash Interrupts</i> replacing FLASH_SR by FLASH_CR register.            Updated c7amba_ioport_UserSpec removing GPIO port bit reset register (GPIOx_BRR) (x = A..K) in the bit field and in <i>Table 23: GPIO register map and reset values</i>.            Updated c7amba_spi2s1_v3_x_UserSpec <i>Section 32.1: Introduction</i> hiding “full duplex and”.</p> <p><b>Updated LTDC section:</b>            - All registers updated, adding [x:x] in the register bit description, register map and putting bold character format.            - Updated <i>Section 18.4.2: Layer programmable parameters</i>.            - Updated <i>Section 18.7.4: LTDC Total Width Configuration Register (LTDC_TWCR)</i> TOTALW[11:0] bit field.            - Updated <i>Section 18.7.5: LTDC Global Control Register (LTDC_GCR)</i> bit 29 description.</p> <p><b>Updated USART section:</b>            - Updated note in <i>Section 31.5.13: USART Smartcard mode</i> about the RTO counter start.            - Updated <i>Table 178: USART interrupt requests</i> removing error in the line receiver timeout error.</p>

Table 309. Document revision history (continued)

Date	Revision	Changes
21-Jul-2015	2 (continued)	<p><b>Updated PWR section:</b> Updated: <i>Table 15, Table 16, Table 18 and Table 19</i> adding 'No interrupt (for WFI) or event (for WFE) is pending'.</p> <p><b>Update RTC section:</b></p> <ul style="list-style-type: none"> <li>- Updated <i>Section : Programming the wakeup timer</i> point 3: Program the wakeup auto-reload value.</li> <li>- Updated <i>Section 29.6.4: RTC initialization and status register (RTC_ISR)</i> bit 2 'WUTWF: wakeup timer write flag' description.</li> </ul> <p><b>Updated DCMI section:</b> Updated <i>Section 17.1: DCMI introduction</i>. Output mode not supported.</p>
03-Nov-2015	3	<p><b>Updated RCC (Reset and Clock Control) section:</b></p> <ul style="list-style-type: none"> <li>- Updated <i>Section 5.3.23: RCC PLLI2S configuration register (RCC_PLLI2SCFGR)</i> PLLI2S[2:0] by PLLI2SR[2:0].</li> <li>- Updated <i>Section 5.3.2: RCC PLL configuration register (RCC_PLLCFGR)</i> PLLN[8:0].</li> <li>- Updated <i>Section 5.3.23: RCC PLLI2S configuration register (RCC_PLLI2SCFGR)</i> PLLI2SN[8:0].</li> <li>- Updated <i>Section 5.3.24: RCC PLLSAI configuration register (RCC_PLLSAICFGR)</i> PLLSAIN[8:0].</li> </ul> <p><b>Updated RTC2 section:</b> Updated <i>Figure 284: RTC block diagram</i>.</p> <p><b>Updated GPIO section:</b> Updated <i>Section 6.4.3: GPIO port output speed register (GPIOx_OSPEEDR)</i> (<math>x = A</math> to <math>K</math>) OSPEEDRy[1:0] definition.</p> <p><b>Updated general-purpose timer:</b></p> <ul style="list-style-type: none"> <li>- Updated <i>Table 154: TIMx internal trigger connection</i> TIM2_ITR1 and TIM3_ITR2.</li> <li>- Updated <i>Section 23.4.19: TIM2 option register (TIM2_OR)</i> PTP by ETH_PTP.</li> <li>- Updated <i>Section 23.3.13: One-pulse mode</i> replacing by "Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx_CCMR1 register".</li> <li>- Removed TIM3_OR register in the bit shield and in the register map.</li> </ul> <p><b>Updated Lptimer1 section:</b></p> <ul style="list-style-type: none"> <li>- Updated <i>Section 26.3: LPTIM implementation</i> removing LPTIM2 in the description, removing LPTIM2 option register (LPTIM2_OR) in the bit field and in the register map.</li> <li>- Updated <i>Table 163: STM32F75xxx and STM32F74xxx LPTIM features</i> removing LPTIM2 column.</li> </ul> <p><b>Updated ethernet section:</b> Updated <i>Section 38.3: Ethernet pins on page 1519</i> removing the column port at the alternate function mapping table and adding reference to the mapping of the datasheet.</p> <p><b>Updated Icdtft section:</b> Updated <i>Section 18.7.26: LTDC register map</i> fixing AHBP[11:0], AAV[11:0], TOTALW[11:0] and HSW[11:0] at 12 bits.</p>

Table 309. Document revision history (continued)

Date	Revision	Changes
03-Nov-2015	3 (continued)	<p><b>Updated SPI2S section:</b></p> <ul style="list-style-type: none"> <li>- Updated Section 32.5.2: Communications between one master and one slave and Section 32.5.3: Standard multislave communication figures 340, 341, 342 and 343.</li> <li>- Notes updated and added below the figures.</li> <li>- Added Section 32.5.4: Multimaster communication.</li> </ul> <p><b>Updated FMC section:</b></p> <ul style="list-style-type: none"> <li>- Updated Section 13.5.3: SDRAM address mapping.</li> <li>- Updated Section 13.8.6: FMC register map.</li> <li>- Changed figures for Mode1, ModeA, Mode2, ModeB, ModeC and Muxed write access waveforms (from FSMC to FMC).</li> <li>- Updated Section 13.6.4: NOR flash/PSRAM controller asynchronous transactions.</li> <li>- Updated Section : FIFO status and interrupt register (FMC_SR) adding note for Bit0 (IRS) and Bit2 (IFS).</li> <li>- Updated Section : SRAM/NOR-flash chip-select timing register for bank x (FMC_BTRx).</li> </ul> <p>Updated Section : SRAM/NOR-flash write timing registers x (FMC_BWTRx).</p> <p><b>Updated MMAP section:</b></p> <ul style="list-style-type: none"> <li>- Updated Section 2.5: Boot configuration embedded bootloader description.</li> </ul> <p><b>Updated I2C2 section:</b></p> <ul style="list-style-type: none"> <li>- Updated : I2C timings and Figure 301: Setup and hold timings.</li> </ul> <p><b>Update USB_OTG section:</b></p> <ul style="list-style-type: none"> <li>- Updated Section 37: USB on-the-go full-speed/high-speed (OTG_FS/OTG_HS).</li> </ul>
21-Dec-2015	4	<p><b>Updated USART section:</b></p> <p>Replaced nCTS by CTS - nRTS by RTS - SCLK by CK everywhere in the document.</p> <p>Replaced "w" by "rc_w1" in Section 31.8.9: Interrupt flag clear register (USARTx_ICR).</p> <p>Updated RTOF bit description in USARTx_ISR.</p> <p><b>Update Low-power timer section:</b></p> <p>Removed LPTIM1 option register (LPTIM1_OR).</p> <p><b>Updated advanced control timer section:</b></p> <p>Updated Section 22.3.16: Using the break function description.</p> <p><b>Updated BxCan section:</b> Dual can IP implemented.</p> <p><b>Updated USB section:</b></p> <p>Updated Section : Choosing the value of TRDT in OTG_GUSBCFG.</p> <p>Updated TRDT bit description in Section 37.15.4: OTG USB configuration register (OTG_GUSBCFG).</p> <p>Added Table 234: TRDT values (FS) and Table 235: TRDT values (HS).</p> <p>Updated Section 37.15.4: OTG USB configuration register (OTG_GUSBCFG) PHYSEL bit "r" to "rw" and PHYSEL description for USB OTG_HS.</p>

Table 309. Document revision history (continued)

Date	Revision	Changes
21-Dec-2015	4 (continued)	<p><b>Updated RTC section:</b> Updated reference to TAMPTS bit in <i>Section 29.3.13: Time-stamp function.</i></p> <p><b>Updated ADC section:</b> Added note in <i>Section 15.13.7: ADC watchdog higher threshold register (ADC_HTR)</i> and <i>Section 15.13.8: ADC watchdog lower threshold register (ADC_LTR)</i>.</p> <p><b>Updated LTDC section:</b> Updated <i>Section 18.3.2: LTDC reset and clocks.</i></p> <p><b>Updated SDMMC section:</b> Updated limit from 48 to 50 MHz in <i>Section 35.1: SDMMC main features</i>, <i>Section 35.3: SDMMC functional description</i>, <i>Section 35.8.1: SDMMC power control register (SDMMC_POWER)</i>, <i>Section 35.8.2: SDMMC clock control register (SDMMC_CLKCR)</i> and in <i>Section 35.8.4: SDMMC command register (SDMMC_CMD)</i>.</p> <p><b>Updated general purpose timer section:</b> Updated reset value to 0xFFFF in <i>Section 23.4.15: TIMx auto-reload register (TIMx_ARR)(x = 2 to 5)</i>. Updated CC1IF bit description in <i>Section 23.4.5: TIMx status register (TIMx_SR)(x = 2 to 5)</i>.</p> <p>Added <i>Section 23.3.14: Retriggerable one pulse mode.</i></p> <p><b>Update PWR power controller section:</b> Updated <i>Section 4.4.2: PWR power control/status register (PWR_CSR1)</i> 'EIWUP' bit 8 introduced in the register field and register map. Updated <i>Table 18</i> and <i>Table 19</i> PPDS bit is cleared in Power Control register (PWR_CR1). Updated <i>Table 19: Standby mode entry and exit</i> WUIF bit is cleared in PWR_CSR1, EIWUP bit is set in PWR_CSR1. Updated <i>Section : RTC alternate functions to wake up the device from the Standby mode</i> adding new "c" bullet for the three cases.</p> <p><b>Updated USART section:</b> Updated <i>Section 31.8.3: Control register 3 (USARTx_CR3)</i> 'ONEBIT' bit 11 description adding a note.</p> <p><b>Updated Electronic signature section:</b> Updated <i>Section 41.1: Unique device ID register (96 bits)</i> UID[31:0] bits used for X,Y coordinates on the wafer, Lot number and Wafer number.</p> <p><b>Updated interrupts section:</b> Updated <i>Section 11.9.1: Interrupt mask register (EXTI_IMR)</i> MRx to IMx bit description in the bit field and register map. Updated <i>Section 11.9.2: Event mask register (EXTI_EMR)</i> MRx to EMx bit description in the bit field and register map.</p>

**Table 309. Document revision history (continued)**

Date	Revision	Changes
21-Dec-2015	4 (continued)	<p><b>Updated FMC section:</b>            Updated BUSTURN bit description in <i>Section : SRAM/NOR-flash chip-select timing register for bank x (FMC_BTRx)</i>.            Updated <i>Section 13.8.3: SDRAM controller functional description</i>.            Updated <i>Figure 53: NAND flash controller waveforms for common memory access</i> adding note 2.            Updated <i>Section 13.7.5: NAND flash prewait functionality</i>.            Updated <i>Common memory space timing register (FMC_PMEM)</i> MEMHOLD[7:0] description.            Updated <i>Attribute memory space timing register (FMC_PATT)</i> ATTHOLD[7:0] description.            Updated <i>Section 13.4: AHB interface</i>.            Updated <i>Table 63: FMC_BCRx bitfields (mode 1)</i> bit 10 and 11 description.</p> <p><b>Updated Flash memory section:</b>            Updated <i>Section 3.5: FLASH memory protection</i> read protection (RDP) description and note added.</p>
26-Apr-2016	5	<p><b>Updated PWR section:</b>            – Updated <i>Table 14: Features over all modes</i> cryptographic processor (CRYP), USB OTG_FS, USB OTG_HS and Ethernet rows.</p> <p><b>Updated DMA section:</b>            – Updated <i>Section 8.5.5: DMA stream x configuration register (DMA_SxCR)</i> bit 18 “DBM or reserved” by “DBM” and “rw or r” by “rw”.</p> <p><b>Updated DMA2D1 section:</b>            – Updated <i>Section 9.3.12: DMA2D transfer control (start, suspend, abort and completion)</i> putting “automatic CLUT transfers can also be aborted or suspended by using the ABORT or the SUSP bit of the DMA2D_CR register”.            – Updated <i>Section 9.5.8: DMA2D foreground PFC control register (DMA2D_FGPCCR)</i> START bit property in the table by “rs”.            – Updated <i>Section 9.5.10: DMA2D background PFC control register (DMA2D_BGPCCR)</i> START bit property in the table by “rs”. and START bit description putting “data transfer or automatic foreground CLUT transfer”.</p> <p><b>Updated RTC section:</b>            – Added case of RTC clocked by LSE in <i>Section 29.3.9: Resetting the RTC</i>.</p> <p><b>Updated RCC section:</b>            – Updated <i>Section 5.2.8: RTC/AWU clock</i> adding “the RTC remains clocked and functional under system reset” when the RTC clock is LSE.            – Updated <i>Section 5.3.26: RCC dedicated clocks configuration register (DCKCFGR2)</i> and <i>Section 5.3.27: RCC register map</i> bit28 renamed SDMMC1SEL.</p> <p><b>Updated HASH section:</b>            – Updated <i>Figure 143: Message data swapping feature</i>.</p>

Table 309. Document revision history (continued)

Date	Revision	Changes
26-Apr-2016	5 (continued)	<p><b>Updated TIMER section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 25.4.7: <i>TIMx prescaler (TIMx_PSC)(x = 6 to 7)</i> PSC[15:0] bits description.</li> <li>– Updated Section 24.4.10: <i>TIMx prescaler (TIMx_PSC)(x = 9, 12)</i> PSC[15:0] bits description.</li> <li>– Updated Section 24.5.1: <i>TIMx control register 1 (TIMx_CR1)(x = 10, 11, 13, 14)</i> adding OPM bit-field.</li> <li>– Updated Section 24.5.13: <i>TIM10/TIM11/TIM13/TIM14 register map</i> adding OPM bit.</li> <li>– Updated Section 22.4.5: <i>TIMx status register (TIMx_SR)(x = 1, 8)</i> and Section 22.4.27: <i>TIM8 register map</i> CC5IF and CC6IF bit names.</li> <li>– Updated Section 26: <i>Low-power timer (LPTIM)</i> changing register name LPTIMx_regname in LPTIM_regname.</li> </ul> <p><b>Updated FMC section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section : <i>SRAM/NOR-flash chip-select timing register for bank x (FMC_BTRx)</i> busturn bits description.</li> <li>– Updated Figure 47: <i>Muxed write access waveforms NWE signal negative edge.</i></li> <li>– Updated Figure 53: <i>NAND flash controller waveforms for common memory access</i> replacing ‘MEMxHIZ’ by ‘MEMxHIZ+1’.</li> <li>– Updated Section : <i>Common memory space timing register (FMC_PMEM)</i> MEMHOLD[7:0] and Section : <i>Attribute memory space timing register (FMC_PATT)</i> ATTHOLD[7:0] replacing 257 HCLK by 256 HCLK.</li> <li>– Updated Section : <i>SDRAM control register x (FMC_SDCRx)</i> adding RPIPE[1:0] description.</li> </ul> <p><b>Updated I2C2 section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 48.4.5: <i>I2C initialization</i>, Section 30.4.9: <i>I2C controller mode</i> and Section 30.7.5: <i>Timing register (I2C_TIMINGR)</i>.</li> </ul> <p><b>Updated cover</b> adding the PM0253 programming manual in the related document list.</p> <p><b>Updated system configuration:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 2.1.12: <i>LCD-TFT controller DMA bus</i> description.</li> </ul> <p><b>Updated USART section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 31: <i>Universal synchronous/asynchronous receiver transmitter (USART/UART)</i> changing register name USARTx_regname in USART_regname.</li> </ul>

Table 309. Document revision history (continued)

Date	Revision	Changes
26-Apr-2016	5 (continued)	<p><b>Updated LDC-TFT section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 40: LCD-TFT display controller (LTDC)</i>.</li> <li>– Updated <i>Section 40.7.15: LTDC layer x window horizontal position configuration register (LTDC_LxWHPCR)</i> removing “all values within this range are allowed” and updating the WHSPPOS[11:0] and WHSTPOS[11:0] bit description.</li> <li>– Updated <i>Section 40.7.15: LTDC layer x window horizontal position configuration register (LTDC_LxWHPCR)</i> removing “all values within this range are allowed” and updating the WVSPPOS[10:0] and WVSTPOS[10:0] bit description.</li> <li>– Updated <i>Table 116: LTDC pins and signal interface</i> modifying ‘Data Enable’ by ‘Not Data Enable’.</li> </ul> <p><b>Updated Power Controller (PWR) section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 4.1.4: Voltage regulator</i> removing low voltage mode in under-drive mode.</li> <li>– Updated <i>Section : Entering low-power mode</i> removing low voltage.</li> </ul> <p><b>Updated Flash memory section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 3.3.7: Flash programming sequences</i> adding note about the FLASH_CR register.</li> <li>– Updated <i>Section 3.3.7: Flash programming sequences</i> adding a note in <i>Section : Standard programming</i>.</li> <li>– Updated <i>Section : Modifying user option bytes</i> adding a note about the FLASH_OPTCR register.</li> </ul>

Table 309. Document revision history (continued)

Date	Revision	Changes
05-Oct-2016	6	<p><b>Updated RCC section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 5.3.3: <i>RCC clock configuration register (RCC_CFGR)</i> two cautions replacing 90 MHz by 108 MHz and 45 MHz by 54 MHz.</li> <li>– Updated Section 5.3.2: <i>RCC PLL configuration register (RCC_PLLCFGR)</i> PLLP[1:0] bit description replacing '180 MHz' by '216 MHz'.</li> </ul> <p><b>Updated RTC section:</b></p> <ul style="list-style-type: none"> <li>– Updated Figure 284: <i>RTC block diagram</i> RTC_WUTR 'ck_spre' input.</li> <li>– Updated ADD1H and SUB1H bit descriptions in Section 29.6.3: <i>RTC control register (RTC_CR)</i>.</li> <li>– Updated Section : <i>RTC backup registers</i> and Section 29.6.20: <i>RTC backup registers (RTC_BKPxR)</i> RTC_BKPxR registers cannot be reset when the Flash readout protection is disabled.</li> <li>– Added caution at the end of Section 29.6.3: <i>RTC control register (RTC_CR)</i>.</li> <li>– Updated caution at the end of Section 29.6.16: <i>RTC tamper configuration register (RTC_TAMPCR)</i>.</li> </ul> <p><b>Updated FMC section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section : <i>SDRAM control register x (FMC_SDCRx)</i> replacing 'KCK_FMC' by 'HCLK' in RPIPE[1:0] description.</li> <li>– Updated Section : <i>SRAM/NOR-flash chip-select timing register for bank x (FMC_BTRx)</i> and Section : <i>SRAM/NOR-flash write timing registers x (FMC_BWTRx)</i> BUSTURN[3:0] bit description replacing 'FMC' by 'HCLK' clock cycle.</li> </ul> <p><b>Updated LCD-TFT section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 40.4.2: <i>Layer programmable parameters</i> modifying configuring 'WHSTPOS[10:0] and WHSPPOS[10:0]' by 'WVSTPOS[10:0] and WVSPPOS[10:0]' in the LTDC_LxWVPCR register.</li> <li>– Updated Figure 421: <i>Layer window programmable parameters</i>.</li> <li>– Updated Section : <i>Example of synchronous timings configuration LTDC_SSCR register</i>.</li> </ul> <p><b>Updated USART section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 31.8.8: <i>USART interrupt and status register (USART_ISR)</i> RWU bit available independently of the wakeup from stop feature availability.</li> <li>– Updated Section 31.8.12: <i>USART register map</i>.</li> <li>– Updated Table 199: <i>Error calculation for programmed baud rates at fCK = 216 MHz in both cases of oversampling by 8 (OVER8 = 1)</i>.</li> </ul> <p><b>Updated I2C2 section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 30.4.9: <i>I2C controller mode note in Master communication initialization (address phase) paragraph</i>.</li> <li>– Updated Section 30.7.2: <i>Control register 2 (I2C_CR2)</i> completing the note of bit 13 'START' description.</li> </ul>

Table 309. Document revision history (continued)

Date	Revision	Changes
05-Oct-2016	6 (continued)	<p><b>Updated DEBUG section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 40.16.2: <i>Debug support for timers, watchdog, bxCAN and I<sup>2</sup>C</i> adding a paragraph for timers having complementary outputs.</li> <li>– Updated Section 40.16.5: <i>Debug MCU APB1 freeze register (DBGMCU_APB1_FZ)</i> and Section 40.16.6: <i>Debug MCU APB2 Freeze register (DBGMCU_APB2_FZ)</i> DBG_TIMx_STOP bit description.</li> </ul> <p><b>Updated TIMER section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 22.3.27: <i>DMA burst mode</i> adding note ‘reserved registers can be written with a null value’.</li> <li>– Updated Section 22.3.28: <i>Debug mode</i> adding ‘for safety purposes’ paragraph.</li> <li>– Updated Table 150: <i>Output control bits for complementary OCx and OCxN channels with break feature</i> in Section 22: <i>Advanced-control timers (TIM1/TIM8)</i>.</li> <li>– Updated Section 23.4.16: <i>TIMx capture/compare register 1 (TIMx_CCR1)(x = 2 to 5)</i>, Section 23.4.17: <i>TIMx capture/compare register 2 (TIMx_CCR2)(x = 2 to 5)</i>, Section 23.4.18: <i>TIMx capture/compare register 3 (TIMx_CCR3)(x = 2 to 5)</i> and Section 23.4.19: <i>TIMx capture/compare register 4 (TIMx_CCR4)(x = 2 to 5)</i> bit description adding ‘the CCRx register is read-only and cannot be written’ and putting rw/r for all the bits.</li> </ul> <p><b>Updated ADC section:</b></p> <p>Updated all ‘CCx’ by ‘CHx’ references and removing ‘event’ in:</p> <ul style="list-style-type: none"> <li>– <i>Figure 70: Single ADC block diagram</i>.</li> <li>– <i>Table 96: External trigger for regular channels</i>.</li> <li>– <i>Table 97: External trigger for injected channels</i>.</li> <li>– <i>Section 15.13.3: ADC control register 2 (ADC_CR2) EXTSEL[3:0] and JEXTSEL[3:0]</i> bit description.</li> </ul>

**Table 309. Document revision history (continued)**

Date	Revision	Changes
07-Mar-2018	7	<p>Updated Arm word and added logo.</p> <p><b>Updated USB section:</b></p> <ul style="list-style-type: none"> <li>– Complete re-mastering of the section</li> <li>– Added <i>Section 37.4.2: OTG_FS/OTG_HS pin and internal signals.</i></li> <li>– Updated <i>Section 37.15.16: OTG core ID register (OTG_CID).</i></li> <li>– Updated <i>Section 37.15.41: OTG all endpoints interrupt mask register (OTG_DAINTMSK)</i> replacing bit 18 by bit 19 in OEPMSK bit description.</li> </ul> <p><b>Updated memory organization section</b></p> <p>Added <i>Figure 2: Memory map.</i></p> <p><b>Updated SPDIFRX section:</b></p> <ul style="list-style-type: none"> <li>– All reference to spdifrx_ck_symb removed.</li> <li>– Updated <i>Figure 413: SPDIFRX block diagram.</i></li> <li>– Removed ‘symbol clock generation’ paragraph.</li> </ul> <p><b>Updated general-purpose timer section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 24.5.12: TIM11 option register 1 (TIM11_OR)</i> TI1_RMP[1:0] bits 1/0 description by SPDIFRX_FRAME_SYNC .... SPDIF frames.</li> <li>– Updated <i>Section 23.4.3: TIMx slave mode control register (TIMx_SMCR)(x = 2 to 5)</i> Bits 6:4 TS: Trigger selection description removing ‘reserved’.</li> <li>– Updated <i>Section : SDRAM control register x (FMC_SDCRx)</i> bits [1:0] ‘01’ description for SPDIFRX_FRAME_SYNC’.</li> <li>– Added <i>Section 24.3.12: Retriggerable one pulse mode (TIM12 only).</i></li> <li>– Added <i>Section 24.3.13: UIF bit remapping.</i></li> <li>– Added <i>Section 24.3.15: Timer input XOR function.</i></li> </ul> <p><b>Updated RCC section</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Figure 13: Clock tree OTG_HS_SCL renamed by OTG_HS_ULPI_CK.</i></li> <li>– Updated <i>Section 5.3.21: RCC clock control &amp; status register (RCC_CSR)</i> bit RMVF put in read/write.</li> <li>– Updated <i>Section 5.3.27: RCC register map PADRSTF</i> in PINRSTF.</li> <li>– Updated <i>Section 5.3.20: RCC backup domain control register (RCC_BDCR)</i> adding LSEDRV[1:0] in the description.</li> <li>– Updated <i>Section 5.3.14: RCC APB2 peripheral clock enable register (RCC_APB2ENR)</i> ADC1EN, ADC2EN and ADC3EN are enabled when bit set to ‘1’.</li> </ul> <p><b>Updated Document conventions section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 1.2: List of abbreviations for registers</i></li> </ul> <p><b>Updated PWR section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 4.1.5: Battery backup domain note</i> removing ‘only one I/O at a time can be used as an output’ sentence.</li> <li>– Updated <i>Section 4.4.2: PWR power control/status register (PWR_CSR1)</i> bits[19:18] UDRDY[1:0] description.</li> <li>– Updated <i>Section 4.1.3: Battery backup domain step 3 of ‘Access to the backup SRAM’ paragraph.</i></li> </ul>

Table 309. Document revision history (continued)

Date	Revision	Changes
07-Mar-2018	7 (continued)	<p><b>Updated Flash memory section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 3.7.7: <i>Flash option control register (FLASH_OPTCR1)</i> and Section 3.7.8: <i>Flash interface register map</i> reset value at ‘0x0040 0000’.</li> </ul> <p><b>Updated FMC section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section : <i>SDRAM control register x (FMC_SDCRx)</i> replacing ‘KCK_FMC’ by ‘HCLK’ in RPIPE[1:0] description.080’.</li> <li>– Updated SDRAM controller read cycle: CAS latency = 3, RPIPE delay = 0 to 2: Five data.</li> </ul> <p><b>Updated RTC section::</b></p> <ul style="list-style-type: none"> <li>– Updated Section 29.6.3: <i>RTC control register (RTC_CR)</i> WUTE bit description adding note.</li> <li>– Updated Figure 298: <i>RTC block diagram</i> WUCKSEL dividers for /2,4,8,16.</li> </ul> <p><b>Updated ETHERNET section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section : <i>Ethernet MAC MII address register (ETH_MACMIIAR)</i> bits[4:2] clock range of setting 100 is valid for 150-216 MHz.</li> </ul> <p><b>Updated QUADSPI section:</b></p> <ul style="list-style-type: none"> <li>– Added Table 90: <i>QUADSPI pins</i>.</li> <li>– Updated Section 14.5.1: <i>QUADSPI control register (QUADSPI_CR)</i> PRESCALER[7:0] bit description.</li> </ul> <p><b>Updated DCMI section:</b></p> <ul style="list-style-type: none"> <li>– Section <i>DCMI pins</i> merged with Section 17.4.3: <i>DCMI physical interface</i>.</li> <li>– External signals HSYNC, VSYNC and PIXCLK standardized to DCMI_HSYNC, DCMI_VSYNC and DCIM_PIXCLK in the whole document.</li> </ul> <p><b>Updated LTDC section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 40.3.3: <i>LTDC reset and clocks</i>.</li> <li>– Updated CFBP bitfield size in LTDC_LxCFBLR.</li> </ul> <p><b>Updated RNG section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 19: <i>True random number generator (RNG)</i>.</li> </ul> <p><b>Updated CRYP section:</b></p> <ul style="list-style-type: none"> <li>– Updated Section 20: <i>Cryptographic processor (CRYP)</i>.</li> </ul> <p><b>Updated HASH section.</b></p> <ul style="list-style-type: none"> <li>– Updated Section 21: <i>Hash processor (HASH)</i>.</li> </ul> <p><b>Updated LPTIM section:</b></p> <ul style="list-style-type: none"> <li>– Added Section 26.4.2: <i>LPTIM trigger mapping</i>.</li> <li>– Added Table 168: <i>Interrupt events</i>.</li> </ul>

**Table 309. Document revision history (continued)**

Date	Revision	Changes
07-Mar-2018	7 (continued)	<p><b>Updated I2C2 section:</b></p> <ul style="list-style-type: none"><li>– Updated <i>Figure 301: Setup and hold timings</i>.</li></ul> <p><b>Updated Serial audio section:</b></p> <ul style="list-style-type: none"><li>– Updated <i>Section 33.5.18: SAI register map MCJDIV by MCKDIV bit name</i>.</li><li>– Added <i>Section 33.3.2: SAI pins and internal signals</i>.</li></ul> <p><b>Updated USART section:</b></p> <ul style="list-style-type: none"><li>– Updated <i>Table 197: STM32F75xxx and STM32F74xxx USART features</i> adding note 2.</li></ul> <p><b>Updated SPI2S section:</b></p> <ul style="list-style-type: none"><li>– Removed text note ‘When the SPI is in slave mode ....the clock is stable in steady state’.</li><li>– Updated <i>Section 32.5.9: Data transmission and reception procedures</i> cross reference ‘communication using DMA’.</li><li>– Updated <i>Section 32.7.2: I2S full duplex</i></li><li>– Updated <i>Section 32.1: Introduction</i>.</li><li>– Updated <i>Section 32.9.9: SPIx_I2S prescaler register (SPIx_I2SPR)</i> removing example.</li></ul> <p><b>Updated SDMMC section:</b></p> <ul style="list-style-type: none"><li>– Updated <i>Section 35.8.8: SDMMC data length register (SDMMC_DLEN)</i> note.</li></ul>

Table 309. Document revision history (continued)

Date	Revision	Changes
25-Jun-2018	8	<p><b>Updated system architecture section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Figure 1: System architecture for STM32F75xxx and STM32F74xxx devices.</i></li> </ul> <p><b>Updated embedded Flash memory section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 3.3.1: Flash memory organization</i> and added <i>Table 4: STM32F750xx Flash memory organization.</i></li> </ul> <p><b>Updated LCD-TFT controller section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 40: LCD-TFT display controller (LTDC)</i> “STM32F756xx and STM32F46xx” by “STM32F756xx, STM32F750xx and STM32F46xx”.</li> </ul> <p><b>Updated CRYP section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 20: Cryptographic processor (CRYP)</i> adding introduction for the whole STM32F756xx and STM32F750xx devices.</li> </ul> <p><b>Updated HASH section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 21: Hash processor (HASH)</i> “STM32F756xx” by “STM32F756xx and STM32F750xx”.</li> </ul> <p><b>Updated RCC section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 41.3: Package data register 0x1FF0 7BF0</i> instead of “0x1FFF 7BF0”.</li> </ul> <p><b>Updated RCC section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 5.3.25: RCC dedicated clocks configuration register (RCC_DCKCFGR1)</i> name.</li> </ul> <p><b>Updated I2C section:</b></p> <p><i>Updated Section 48.4.1: I2C block diagram:</i></p> <ul style="list-style-type: none"> <li>– Removed ‘For I2C I/Os supporting 20mA ... refer to section: I2C implementation’ paragraph.</li> <li>– removed ‘this independent clock source .... refer to RCC for more details’ paragraph.</li> </ul> <p><i>Updated Section 48.4.5: I2C initialization</i> removing the reference to RCC.</p> <p><i>Updated Section 30.4.9: I2C controller mode master communication initialization (address phase) note.</i></p> <p><i>Updated Section 30.6: I2C interrupts:</i></p> <ul style="list-style-type: none"> <li>– Updated <i>Table 195: I2C interrupt requests</i> according to new IP guideline (acronym column).</li> <li>– Removed figure: I2C interrupt mapping diagram.</li> <li>– Removed ‘depending on the product implementation ..refer to section EXTI’ paragraph.</li> </ul> <p><i>Updated Section 30.9.2: I2C control register 2 (I2C_CR2)</i> START bit 13 description and note.</p> <p><b>Updated Debug support section:</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 40.6.1: MCU device ID code</i> by ‘0x1001 = Revision Z and 1’.</li> </ul>

**Table 309. Document revision history (continued)**

Date	Revision	Changes
03-Feb-2025	9	<p>Small text updates in the document.</p> <p>Updated:</p> <ul style="list-style-type: none"><li>– <i>Exiting low-power mode</i></li><li>– <i>Section 3.3.5: Maximum program/erase parallelism</i></li><li>– PTP clock source in <i>Figure 13: Clock tree</i></li><li>– Typo in <i>Section 6.4: GPIO registers</i></li><li>– <i>Section 3.5.1: Read protection (RDP)</i></li><li>– <i>Table 25: DMA1 request mapping</i></li><li>– <i>Section 15.10: Temperature sensor</i></li><li>– <i>Section 22.4.5: TIMx status register (TIMx_SR)(x = 1, 8)</i></li><li>– Base address for package data register in <i>Section 41: Device electronic signature</i></li><li>– Base address in <i>Section 41.3: Package data register</i></li></ul> <p>Added <i>Section 42: Important security notice</i></p> <p>Master and slave terms in <i>Section 30: Inter-integrated circuit interface (I2C)</i> replaced with controller and target, respectively.</p>

# Index

## A

ADC_CCR .....	454
ADC_CDR .....	457
ADC_CR1 .....	444
ADC_CR2 .....	446
ADC_CSR .....	453
ADC_DR .....	453
ADC_HTR .....	449
ADC_JDRx .....	452
ADC_JOFRx .....	448
ADC_JSQR .....	452
ADC_LTR .....	449
ADC_SMPR1 .....	447
ADC_SMPR2 .....	448
ADC_SQR1 .....	450
ADC_SQR2 .....	450
ADC_SQR3 .....	451
ADC_SR .....	443

## C

CAN_BTR .....	1308
CAN_ESR .....	1307
CAN_FA1R .....	1317
CAN_FFA1R .....	1316
CAN_FiRx .....	1317
CAN_FM1R .....	1315
CAN_FMR .....	1315
CAN_FS1R .....	1316
CAN_IER .....	1306
CAN_MCR .....	1299
CAN_MSR .....	1301
CAN_RDHxR .....	1314
CAN_RDLxR .....	1314
CAN_RDTxR .....	1313
CAN_RF0R .....	1305
CAN_RF1R .....	1305
CAN_RIxR .....	1312
CAN_TDHzR .....	1312
CAN_TDLxR .....	1311
CAN_TDTxR .....	1311
CAN_TIxR .....	1310
CAN_TSR .....	1302
CEC_CFGR .....	1649
CEC_CR .....	1648
CEC_IER .....	1653
CEC_ISR .....	1651
CEC_RXDR .....	1651

CEC_TXDR .....	1651
CRC_CR .....	306
CRC_DR .....	305
CRC_IDR .....	305
CRC_INIT .....	307
CRC_POL .....	307
CRYP_CR .....	599
CRYP_DIN .....	602
CRYP_DMACR .....	603
CRYP_DOUT .....	602
CRYP_IMSCR .....	604
CRYP_IV0LR .....	610
CRYP_IV0RR .....	610
CRYP_IV1LR .....	611
CRYP_IV1RR .....	611
CRYP_K0LR .....	606
CRYP_K0RR .....	606
CRYP_K1LR .....	607
CRYP_K1RR .....	607
CRYP_K2LR .....	608
CRYP_K2RR .....	608
CRYP_K3LR .....	609
CRYP_K3RR .....	609
CRYP_MISR .....	605
CRYP_RISR .....	604
CRYP_SR .....	601

## D

DAC_CR .....	472
DAC_DHR12L1 .....	476
DAC_DHR12L2 .....	477
DAC_DHR12LD .....	478
DAC_DHR12R1 .....	475
DAC_DHR12R2 .....	477
DAC_DHR12RD .....	478
DAC_DHR8R1 .....	476
DAC_DHR8R2 .....	477
DAC_DHR8RD .....	479
DAC_DOR1 .....	479
DAC_DOR2 .....	479
DAC_SR .....	480
DAC_SWTRIGR .....	475
DBGMCU_APB2_FZ .....	1679, 1681
DBGMCU_CR .....	1678
DBGMCU_IDCODE .....	1663
DCMI_CR .....	493
DCMI_CWSIZE .....	502
DCMI_CWSTRT .....	501

DCMI_DR .....	502	ETH_DMARPDR .....	1621
DCMI_ESCR .....	500	ETH_DMARSWTR .....	1631
DCMI_ESUR .....	500	ETH_DMASR .....	1622
DCMI_ICR .....	499	ETH_DMATDLAR .....	1622
DCMI_IER .....	497	ETH_DMATPDR .....	1620
DCMI_MIS .....	498	ETH_MACA0HR .....	1600
DCMI_RIS .....	496	ETH_MACA0LR .....	1601
DCMI_SR .....	496	ETH_MACA1HR .....	1601
DMA_HIFCR .....	250	ETH_MACA1LR .....	1602
DMA_HISR .....	249	ETH_MACA2HR .....	1602
DMA_LIFCR .....	250	ETH_MACA2LR .....	1603
DMA_LISR .....	248	ETH_MACA3HR .....	1604
DMA_SxCR .....	251	ETH_MACA3LR .....	1604
DMA_SxFCR .....	256	ETH_MACCR .....	1586
DMA_SxM0AR .....	255	ETH_MACDBGR .....	1598
DMA_SxM1AR .....	255	ETH_MACFCR .....	1593
DMA_SxNDTR .....	254	ETH_MACFFR .....	1589
DMA_SxPAR .....	255	ETH_MACHTHR .....	1590
DMA2D_AMTCR .....	286	ETH_MACHTLR .....	1591
DMA2D_BGCLUTx .....	287	ETH_MACIMR .....	1600
DMA2D_BGCMAR .....	281	ETH_MACIIAR .....	1591
DMA2D_BGCOLR .....	280	ETH_MACIIDR .....	1592
DMA2D_BGMAR .....	276	ETH_MACPMCSR .....	1597
DMA2D_BGOR .....	276	ETH_MACRWUFFR .....	1595
DMA2D_BGPFCCR .....	279	ETH_MACSR .....	1599
DMA2D_CR .....	272	ETH_MACVLANTR .....	1594
DMA2D_FGCLUTx .....	287	ETH_MMCCR .....	1606
DMA2D_FGCMAR .....	281	ETH_MMCRFAECR .....	1610
DMA2D_FGCOLR .....	278	ETH_MMCRFCECR .....	1610
DMA2D_FGMAR .....	275	ETH_MMCRGUFCR .....	1611
DMA2D_FGOR .....	275	ETH_MMCRIMR .....	1608
DMA2D_FGPFCCR .....	277	ETH_MMCRIR .....	1606
DMA2D_IFCR .....	274	ETH_MMCTGFCCR .....	1610
DMA2D_ISR .....	274	ETH_MMCTGFMSCCR .....	1609
DMA2D_LWR .....	286	ETH_MMCTGFSSCR .....	1609
DMA2D_NLR .....	285	ETH_MMCTIMR .....	1608
DMA2D_OCOLR .....	282-284	ETH_MMCTIR .....	1607
DMA2D_OMAR .....	284	ETH_PTPPPSCR .....	1618
DMA2D_OOR .....	285	ETH_PTPSSIR .....	1613
DMA2D_OPFCCR .....	282	ETH_PTPTSAR .....	1616
		ETH_PTPTSCR .....	1611
		ETH_PTPTSHR .....	1614
		ETH_PTPTSHUR .....	1615
		ETH_PTPTSLR .....	1614
		ETH_PTPTSLUR .....	1616
		ETH_PTPTSSR .....	1617
		ETH_PTPTTHR .....	1617
		ETH_PTPTTLLR .....	1617
		EXTI_EMR .....	298
		EXTI_FTSR .....	299
		EXTI_IMR .....	298
		EXTI_PR .....	300

**E**

ETH_DMABMR .....	1619
ETH_DMACHRBAR .....	1633
ETH_DMACHRDR .....	1632
ETH_DMACHTBAR .....	1632
ETH_DMACHTDR .....	1631
ETH_DMAIER .....	1628
ETH_DMAMFBOCR .....	1630
ETH_DMAOMR .....	1625
ETH_DMARDLAR .....	1621

EXTI_RTSR .....	299
EXTI_SWIER .....	300

**F**

FLITF_FCR .....	95
FLITF_FKEYR .....	93
FLITF_FOPTCR .....	96-97
FLITF_FOPTKEYR .....	93
FLITF_FSR .....	94
FMC_BCRx .....	347
FMC_BTRx .....	349
FMC_BWTRx .....	352
FMC_ECCR .....	364
FMC_PATT .....	363
FMC_PCR .....	359
FMC_PMEM .....	362
FMC_SDCMR .....	379
FMC_SDCRx .....	376
FMC_SDRTR .....	380
FMC_SDSR .....	382
FMC_SDTRx .....	377
FMC_SR .....	361

**G**

GPIOx_AFRH .....	216
GPIOx_AFRL .....	215
GPIOx_BSRR .....	213
GPIOx_IDR .....	213
GPIOx_LCKR .....	214
GPIOx_MODER .....	211
GPIOx_ODR .....	213
GPIOx_OSPEEDR .....	212
GPIOx_OTYPER .....	211
GPIOx_PUPDR .....	212

**H**

HASH_CR .....	626
HASH_CSRx .....	633
HASH_DIN .....	628
HASH_HRAx .....	630
HASH_HRx .....	630-631
HASH_IMR .....	631
HASH_SR .....	632
HASH_STR .....	629

**I**

I2C_CR1 .....	995
I2C_CR2 .....	998
I2C_ICR .....	1006

I2C_ISR .....	1004
I2C_OAR1 .....	1000
I2C_OAR2 .....	1001
I2C_PECR .....	1007
I2C_RXDR .....	1007
I2C_TIMEOUTR .....	1003
I2C_TIMINGR .....	1002
I2C_TXDR .....	1008
IWDG_KR .....	890
IWDG_PR .....	891
IWDG_RLR .....	892
IWDG_SR .....	893
IWDG_WINR .....	894

**L**

LPTIM_ARR .....	885
LPTIM_CFGR .....	880
LPTIM_CMP .....	884
LPTIM_CNT .....	885
LPTIM_CR .....	883
LPTIM_ICR .....	879
LPTIM_IER .....	879
LPTIM_ISR .....	878
LTDC_AWCR .....	519
LTDC_BCCR .....	522
LTDC_BPCR .....	518
LTDC_CDSR .....	525
LTDC_CPSR .....	525
LTDC_GCR .....	520
LTDC_ICR .....	524
LTDC_IER .....	522
LTDC_ISR .....	523
LTDC_LIPCR .....	524
LTDC_LxBFCR .....	531
LTDC_LxCACR .....	530
LTDC_LxCFBAR .....	532
LTDC_LxCFBLNR .....	533
LTDC_LxCFBLR .....	532
LTDC_LxCKCR .....	529
LTDC_LxCLUTWR .....	533
LTDC_LxCR .....	526
LTDC_LxDCCR .....	530
LTDC_LxPFCR .....	529
LTDC_LxWHPCR .....	527
LTDC_LxWVPCR .....	528
LTDC_SRCR .....	521
LTDC_SSCR .....	516
LTDC_TWCR .....	519

**O**

OTG_CID .....	1386
---------------	------

OTG_DAINT	1413
OTG_DAINTMSK	1413
OTG_DCFG	1405
OTG_DCTL	1407
OTG_DEACHINT	1417
OTG_DEACHINTMSK	1417
OTG_DIEPCTL0	1420
OTG_DIEPCTLx	1422
OTG_DIEPDMAx	1427
OTG_DIEPEMPMSK	1416
OTG_DIEPINTx	1424
OTG_DIEPMSK	1410
OTG_DIEPTSIZ0	1426
OTG_DIEPTSIZx	1428
OTG_DIEPTXF0	1383
OTG_DIEPTXFx	1390
OTG_DOEPCTL0	1429
OTG_DOEPCTLx	1434
OTG_DOEPDMAx	1433
OTG_DOEPINTx	1430
OTG_DOEPMISK	1411
OTG_DOEPTSIZ0	1432
OTG_DOEPTSIZx	1436
OTG_DSTS	1409
OTG_DTHRCTL	1415
OTG_DTXFSTSx	1427
OTG_DVBUSDIS	1414
OTG_DVBUSPULSE	1414
OTG_GAHBCFG	1362
OTG_GCCFG	1385
OTG_GINTMSK	1375
OTG_GINTSTS	1370
OTG_GLPMCFG	1386
OTG_GOTGCTL	1357
OTG_GOTGINT	1360
OTG_GRSTCTL	1367
OTG_GRXFSIZ	1382
OTG_GRXSTSP	1380-1381
OTG_GRXSTSR	1378-1379
OTG_GUSBCFG	1364
OTG_HAINT	1394
OTG_HAINTPMSK	1395
OTG_HCCHARx	1398
OTG_HCDMAx	1404
OTG_HCFG	1391
OTG_HCINTMSKx	1402
OTG_HCINTx	1400
OTG_HCSPLTx	1399
OTG_HCTSIZx	1403
OTG_HFIR	1392
OTG_HFNUM	1393
OTG_HNPTXFSIZ	1383
OTG_HNPTXSTS	1384
OTG_HPRT	1396
OTG_HPTXFSIZ	1390
OTG_HPTXSTS	1393
OTG_HS_DIEPEACHMSK1	1418
OTG_HS_DOEPEACHMSK1	1419
OTG_PCGCCTL	1437
<b>P</b>	
PWR_CR	124
PWR_CSR	126, 128, 130
<b>Q</b>	
QUADSPI_ABR	408
QUADSPI_AR	408
QUADSPI_CCR	406
QUADSPI_CR	400
QUADSPI_DCR	403
QUADSPI_DLR	405
QUADSPI_DR	409
QUADSPI_FCR	405
QUADSPI_LPTR	411
QUADSPI_PIR	410
QUADSPI_PSMAR	410
QUADSPI_PSMKR	409
QUADSPI_SR	404
<b>R</b>	
RCC_AHB1ENR	165
RCC_AHB1LPENR	175
RCC_AHB1RSTR	155
RCC_AHB2ENR	167
RCC_AHB2LPENR	177
RCC_AHB2RSTR	158
RCC_AHB3ENR	168
RCC_AHB3LPENR	178
RCC_AHB3RSTR	159
RCC_APB1ENR	168
RCC_APB1LPENR	179
RCC_APB1RSTR	159
RCC_APB2ENR	172
RCC_APB2LPENR	183
RCC_APB2RSTR	163
RCC_BDCR	185
RCC_CFGR	150
RCC_CIR	152
RCC_CR	145
RCC_CSR	186
RCC_PLLCFGR	148, 189, 192
RCC_SSCGR	188

RNG_CR .....	544	SDMMC_POWER .....	1261
RNG_DR .....	546	SDMMC_RESPCMD .....	1265
RNG_SR .....	545	SDMMC_RESPx .....	1265
RTC_ALRMAR .....	931	SDMMC_STA .....	1270
RTC_ALRMASSR .....	942	SPDIFRX_CR .....	1209
RTC_ALRMBMR .....	932	SPDIFRX_CSR .....	1217
RTC_ALRBSSR .....	943	SPDIFRX_DIR .....	1217
RTC_BKPxR .....	944	SPDIFRX_FMT0_DR .....	1215
RTC_CALR .....	938	SPDIFRX_FMT1_DR .....	1215
RTC_CR .....	923	SPDIFRX_FMT2_DR .....	1216
RTC_DR .....	922	SPDIFRX_IFCR .....	1214
RTC_ISR .....	926	SPDIFRX_IMR .....	1211
RTC_OR .....	944	SPDIFRX_SR .....	1212
RTC_PRER .....	929	SPIx_CR1 .....	1121
RTC_SHIFTR .....	934	SPIx_CR2 .....	1123
RTC_SSR .....	933	SPIx_CRCPR .....	1127
RTC_TAMPCR .....	939	SPIx_DR .....	1127
RTC_TR .....	921	SPIx_I2SCFGR .....	1128
RTC_TSDR .....	936	SPIx_I2SPR .....	1130
RTC_TSSSR .....	937	SPIx_RXCRCR .....	1127
RTC_TSTR .....	935	SPIx_SR .....	1125
RTC_WPR .....	933	SPIx_TXCRCR .....	1128
RTC_WUTR .....	930	SYSCFG_EXTICR1 .....	222
		SYSCFG_EXTICR2 .....	222
		SYSCFG_EXTICR3 .....	223
		SYSCFG_EXTICR4 .....	224
		SYSCFG_MEMRMP .....	220

**S**

SAI_ACLRFR .....	1173
SAI_ACR1 .....	1163
SAI_ACR2 .....	1166
SAI_ADR .....	1174
SAI_AFRCR .....	1168
SAI_AIM .....	1170
SAI_ASLOTR .....	1169
SAI_ASRR .....	1171
SAI_BCLRFR .....	1184
SAI_BCR1 .....	1175
SAI_BCR2 .....	1177
SAI_BDR .....	1185
SAI_BFRCR .....	1179
SAI_BIM .....	1181
SAI_BSLOTR .....	1180
SAI_BSR .....	1182
SAI_GCR .....	1163
SDMMC_ARG .....	1264
SDMMC_CLKCR .....	1261
SDMMC_DCOUNT .....	1270
SDMMC_DCTRL .....	1267
SDMMC_DLEN .....	1267
SDMMC_DTIMER .....	1266
SDMMC_FIFO .....	1276
SDMMC_ICR .....	1271
SDMMC_MASK .....	1273

SDMMC_POWER .....	1261
SDMMC_RESPCMD .....	1265
SDMMC_RESPx .....	1265
SDMMC_STA .....	1270
SPDIFRX_CR .....	1209
SPDIFRX_CSR .....	1217
SPDIFRX_DIR .....	1217
SPDIFRX_FMT0_DR .....	1215
SPDIFRX_FMT1_DR .....	1215
SPDIFRX_FMT2_DR .....	1216
SPDIFRX_IFCR .....	1214
SPDIFRX_IMR .....	1211
SPDIFRX_SR .....	1212
SPIx_CR1 .....	1121
SPIx_CR2 .....	1123
SPIx_CRCPR .....	1127
SPIx_DR .....	1127
SPIx_I2SCFGR .....	1128
SPIx_I2SPR .....	1130
SPIx_RXCRCR .....	1127
SPIx_SR .....	1125
SPIx_TXCRCR .....	1128
SYSCFG_EXTICR1 .....	222
SYSCFG_EXTICR2 .....	222
SYSCFG_EXTICR3 .....	223
SYSCFG_EXTICR4 .....	224
SYSCFG_MEMRMP .....	220

**T**

TIM2_OR .....	798
TIM5_OR .....	798
TIMx_ARR .....	715, 795, 838, 850, 864
TIMx_BDTR .....	718
TIMx_CCER .....	711, 792, 836, 848
TIMx_CCMR1	705-706, 786, 788, 833-834, 845-846
TIMx_CCMR2 .....	709-710, 790-791
TIMx_CCMR3 .....	723
TIMx_CCR1 .....	716, 795, 838, 850-851
TIMx_CCR2 .....	717, 795, 839
TIMx_CCR3 .....	717, 796
TIMx_CCR4 .....	718, 796
TIMx_CCR5 .....	724
TIMx_CCR6 .....	725
TIMx_CNT .....	715, 793-794, 837, 849, 863
TIMx_CR1 .....	694, 777, 827, 842, 860
TIMx_CR2 .....	695, 778, 862
TIMx_DCR .....	721, 797
TIMx_DIER .....	700, 783, 830, 843, 862
TIMx_DMAR .....	722, 798
TIMx_EGR .....	704, 785, 832, 844, 863

---

TIMx\_PSC ..... 715, 794, 838, 850, 864  
TIMx\_RCR ..... 716  
TIMx\_SMCR ..... 698, 780, 828  
TIMx\_SR ..... 702, 784, 830, 843, 863

**U**

USART\_BRR ..... 1062  
USART\_CR1 ..... 1053  
USART\_CR2 ..... 1056  
USART\_CR3 ..... 1059  
USART\_GTPR ..... 1063  
USART\_ICR ..... 1070  
USART\_ISR ..... 1066  
USART\_RDR ..... 1071  
USART\_RQR ..... 1065  
USART\_RTOR ..... 1064  
USART\_TDR ..... 1072

**W**

WWDG\_CFR ..... 900  
WWDG\_CR ..... 899  
WWDG\_SR ..... 900

**IMPORTANT NOTICE – READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to [www.st.com/trademarks](http://www.st.com/trademarks). All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2025 STMicroelectronics – All rights reserved

