

## Description

The Atmel SAM4E series of Flash microcontrollers is based on the high-performance 32-bit ARM® Cortex®-M4 RISC processor and includes a floating point unit (FPU). It operates at a maximum speed of 120 MHz and features up to 1024 Kbytes of Flash, 2 Kbytes of cache memory and up to 128 Kbytes of SRAM.

The SAM4E offers a rich set of advanced connectivity peripherals including 10/100 Mbps Ethernet MAC supporting IEEE 1588 and dual CAN. With a single-precision FPU, advanced analog features, as well as a full set of timing and control functions, the SAM4E is the ideal solution for industrial automation, home and building control, machine-to-machine communications, automotive aftermarket and energy management applications.

The peripheral set includes a full-speed USB device port with embedded transceiver, a 10/100 Mbps Ethernet MAC supporting IEEE 1588, a high-speed MCI for SDIO/SD/MMC, an external bus interface featuring a static memory controller providing connection to SRAM, PSRAM, NOR Flash, LCD module and NAND Flash, a parallel I/O capture mode for camera interface, hardware acceleration for AES256, two USARTs, two UARTs, two TWIs, three SPIs, as well as a 4-channel PWM, nine general-purpose 16-bit timers (with stepper motor and quadrature decoder logic support), one RTC, two analog front end interfaces (16-bit ADC, DAC, MUX and PGA), one 12-bit DAC (2-ch) and an analog comparator.

# 1. Features

- Core
  - ARM® Cortex®-M4 with 2 Kbytes Cache running at up to 120 MHz<sup>(1)</sup>
  - Memory Protection Unit (MPU)
  - DSP Instruction
  - Floating Point Unit (FPU)
  - Thumb®-2 Instruction Set
- Memories
  - Up to 1024 Kbytes Embedded Flash
  - 128 Kbytes Embedded SRAM
  - 16 Kbytes ROM with Embedded Boot Loader Routines (UART) and IAP Routines
  - Static Memory Controller (SMC): SRAM, NOR, NAND Support.
  - NAND Flash Controller.
- System
  - Embedded Voltage Regulator for Single Supply Operation
  - Power-on-Reset (POR), Brown-out Detector (BOD) and Dual Watchdog for Safe Operation
  - Quartz or Ceramic Resonator Oscillators: 3 to 20 MHz Main Power with Failure Detection and Optional Low-power 32.768 kHz for RTC or Device Clock
  - RTC with Gregorian and Persian Calendar Mode, Waveform Generation in Low-power Modes
  - RTC Clock Calibration Circuitry for 32.768 kHz Crystal Frequency Compensation
  - High Precision 4/8/12 MHz Factory Trimmed Internal RC Oscillator with 4 MHz Default Frequency for Device Startup. In-application Trimming Access for Frequency Adjustment
  - Slow Clock Internal RC Oscillator as Permanent Low-power Mode Device Clock
  - One PLL up to 240 MHz for Device Clock and for USB
  - Temperature Sensor
  - Up to 2 Peripheral DMA Controller with up to 33 Channels (PDC)
  - One 4-channel DMA Controller
- Low-power Modes
  - Sleep and Backup Modes
  - Ultra Low-power RTC
- Peripherals
  - Two USARTs with USART1 (ISO7816, IrDA®, RS-485, SPI, Manchester and Modem Modes)
  - USB 2.0 Device: Full Speed (12 Mbits), 2668 byte FIFO, up to 8 Endpoints. On-chip Transceiver
  - Two 2-wire UARTs
  - Two Two-wire Interfaces (TWI)
  - High-speed Multimedia Card Interface (SDIO/SD Card/MMC)
  - One Master/Slave Serial Peripheral Interface (SPI) with Chip Select Signals
  - Three 3-Channel 32-bit Timer/Counter with Capture, Waveform, Compare and PWM Mode. Quadrature Decoder Logic and 2-bit Gray Up/Down Counter for Stepper Motor
  - 32-bit Real-time Timer and RTC with Calendar and Alarm Features
  - One Ethernet MAC (EMAC) 10/100 Mbps in MII mode only with Dedicated DMA and Support for IEEE1588, Wake-on-LAN
  - Two CAN Controllers with eight Mailboxes
  - 4-channel 16-bit PWM with Complementary Output, Fault Input, 12-bit Dead Time Generator Counter for Motor Control
  - Real-time Event Management

- Cryptography
  - AES 256-bit Key Algorithm compliant with FIPS Publication 197
- Analog
  - AFE (Analog Front End): 2x16-bit ADC, up to 24-channels, Differential Input Mode, Programmable Gain Stage, Auto Calibration and Automatic Offset Correction
  - One 2-channel 12-bit 1 Msps DAC
  - One Analog Comparator with Flexible Input Selection, Selectable Input Hysteresis
- I/O
  - Up to 117 I/O Lines with External Interrupt Capability (Edge or Level Sensitivity), Debouncing, Glitch Filtering and On-die Series Resistor Termination
  - Bidirectional Pad, Analog I/O, Programmable Pull-up/Pull-down
  - Five 32-bit Parallel Input/Output Controllers, Peripheral DMA Assisted Parallel Capture Mode
- Packages
  - 144-ball LFBGA, 10x10 mm, pitch 0.8 mm
  - 100-ball TFBGA, 9x9 mm, pitch 0.8 mm
  - 144-lead LQFP, 20x20 mm, pitch 0.5 mm
  - 100-lead LQFP, 14x14 mm, pitch 0.5 mm

Note: 1. 120 MHz: -40/+105°C, VDDCORE = 1.2V or using internal voltage regulator

## 1.1 Configuration Summary

The SAM4E series devices differ in memory size, package and features. [Table 1-1](#) summarizes the configurations of the device family.

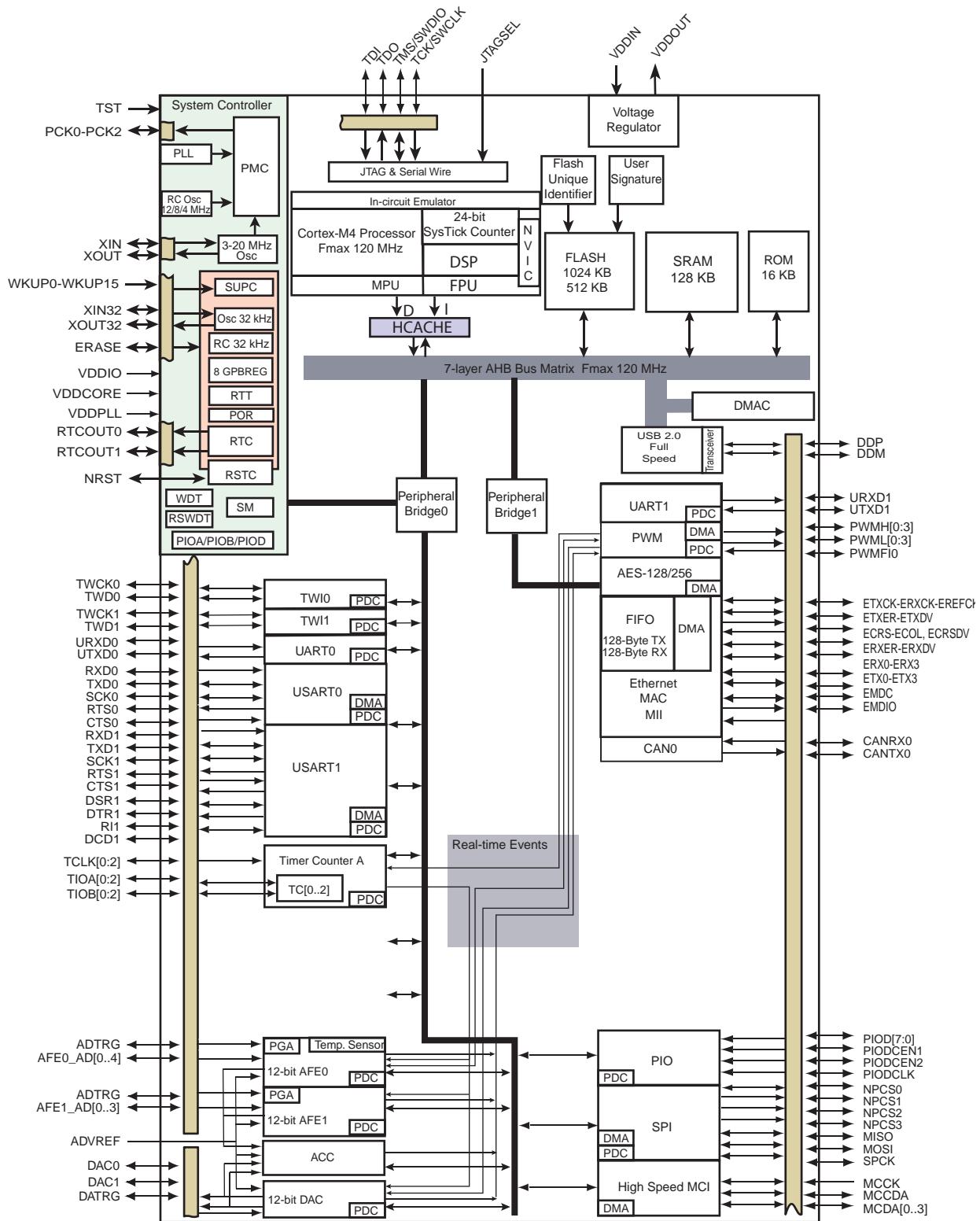
**Table 1-1.** Configuration Summary

Feature	SAM4E16E	SAM4E8E	SAM4E16C	SAM4E8C
<b>Flash</b>	1024 Kbytes	512 Kbytes	1024 Kbytes	512 Kbytes
<b>SRAM</b>	128 Kbytes		128 Kbytes	
<b>CMCC</b>	2 Kbytes		2 Kbytes	
<b>Package</b>	LFBGA 144 LQFP 144		TFBGA 100 LQFP 100	
<b>Number of PIOs</b>	117		79	
<b>External Bus Interface</b>	8-bit Data, 4 Chip Selects, 24-bit Address		-	
<b>Analog Front End (AFE0 \ AFE1)</b>	Up to 16 bits <sup>(1)</sup> 16 ch.\ 8 ch <sup>(2)</sup>		Up to 16 bits <sup>(1)</sup> 6 ch.\ 4ch <sup>(3)</sup>	
<b>EMAC</b>	YES		YES	
<b>CAN</b>	2		1	
<b>12-bit DAC</b>	2 ch.		2 ch.	
<b>Timer</b>	9 <sup>(4)</sup>		3 <sup>(5)</sup>	
<b>PDC Channels</b>	24 +9		21 +9	
<b>USART/ UART</b>	2/2 <sup>(6)</sup>		2/2 <sup>(6)</sup>	
<b>USB</b>	Full Speed		Full Speed	
<b>HSMCI</b>	1 port 4 bits		1 port 4 bits	
<b>TWI</b>	2		2	

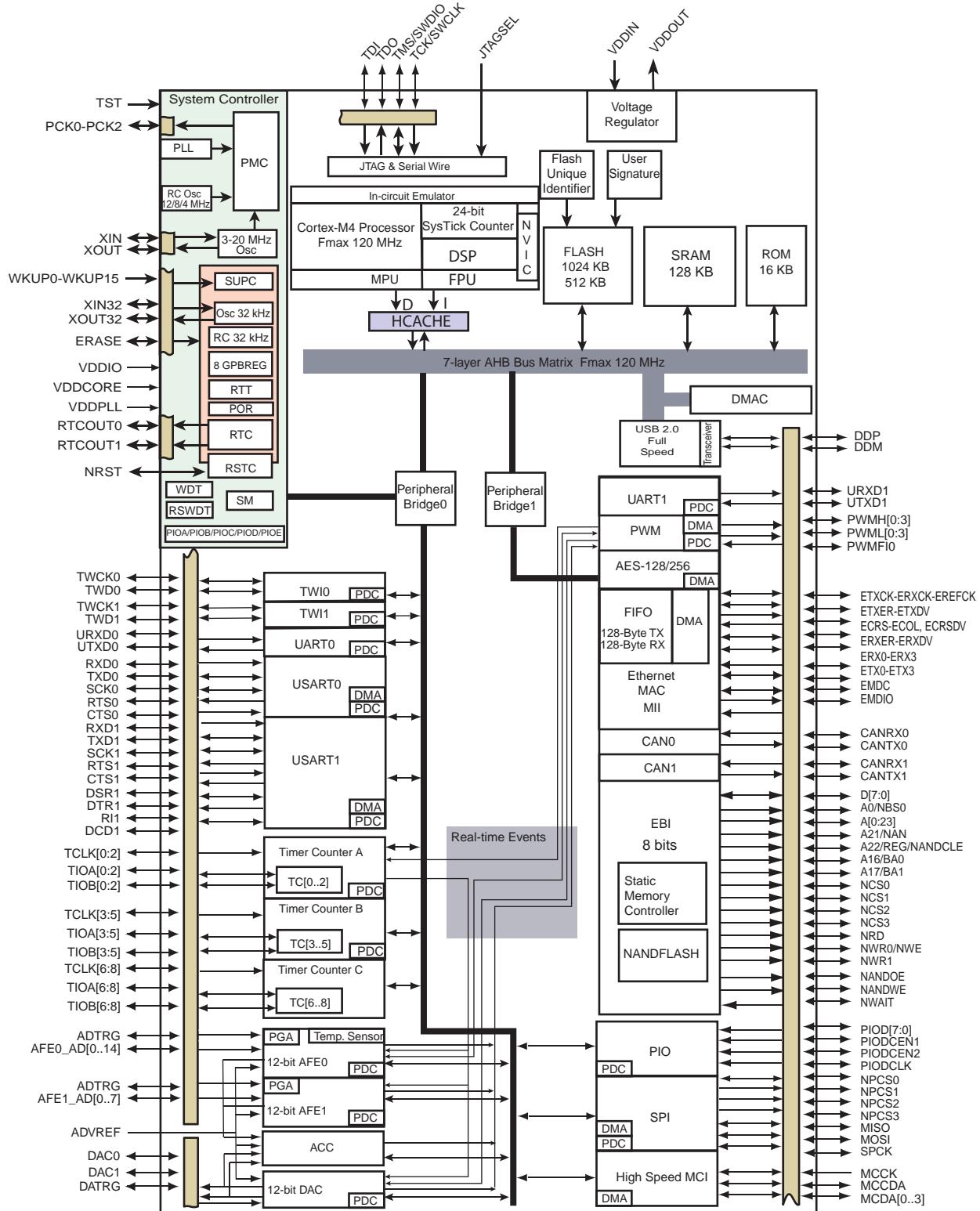
- Notes:
- ADC is 12-bit, up to 16 bits with averaging.  
For details, please refer to the “Electrical Characteristics” section of this datasheet.
  - AFE0 is 16 channels and AFE1 is 8 channels. The total number of AFE channels is 24.  
One channel is reserved for the internal temperature sensor.
  - AFE0 is 6 channels and AFE1 is 4 channels. The total number of AFE channels is 10.  
One channel is reserved for the internal temperature sensor.
  - 9 TC channels are accessible through PIO.
  - 3 TC channels are accessible through PIO.
  - Full Modem support on USART1.

## 2. Block Diagram

Figure 2-1. SAM4E 100-pin Block Diagram



**Figure 2-2. SAM4E 144-pin Block Diagram**



### 3. Signal Description

Table 3-1 gives details on signal names classified by peripheral.

Table 3-1. Signal Description List

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
<b>Power Supplies</b>					
VDDIO	Peripherals I/O Lines Power Supply	Power			1.62V to 3.6V
VDDIN	Voltage Regulator Input, DAC and Analog Comparator Power Supply	Power			1.62V to 3.6V <sup>(1)</sup>
VDDOUT	Voltage Regulator Output	Power			1.2V Output
VDDPLL	Oscillator and PLL Power Supply	Power			1.08 V to 1.32V
VDDCORE	Power the core, the embedded memories and the peripherals	Power			1.08V to 1.32V
GND	Ground	Ground			
<b>Clocks, Oscillators and PLLs</b>					
XIN	Main Oscillator Input	Input		VDDIO	Reset State: - PIO Input - Internal Pull-up disabled - Schmitt Trigger enabled <sup>(2)</sup>
XOUT	Main Oscillator Output	Output			
XIN32	Slow Clock Oscillator Input	Input			
XOUT32	Slow Clock Oscillator Output	Output			
PCK0 - PCK2	Programmable Clock Output	Output			Reset State: - PIO Input - Internal Pull-up enabled - Schmitt Trigger enabled <sup>(2)</sup>
<b>Real-time Clock</b>					
RTCOUT0	Programmable RTC waveform output	Output		VDDIO	Reset State: - PIO Input
RTCOUT1	Programmable RTC waveform output	Output			- Internal Pull-up enabled - Schmitt Trigger enabled <sup>(2)</sup>
<b>Serial Wire/JTAG Debug Port - SWJ-DP</b>					
TCK/SWCLK	Test Clock/Serial Wire Clock	Input		VDDIO	Reset State: - SWJ-DP Mode
TDI	Test Data In	Input			- Internal Pull-up disabled <sup>(3)</sup>
TDO/TRACESWO	Test Data Out / Trace Asynchronous Data Out	Output			- Schmitt Trigger enabled <sup>(2)</sup>
TMS/SWDIO	Test Mode Select /Serial Wire Input/Output	Input / I/O			
JTAGSEL	JTAG Selection	Input	High		Permanent Internal Pull-down
<b>Flash Memory</b>					
ERASE	Flash and NVM Configuration Bits Erase Command	Input	High	VDDIO	Reset State: - Erase Input - Internal Pull-down enabled - Schmitt Trigger enabled <sup>(2)</sup>

Table 3-1. Signal Description List

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
<b>Reset/Test</b>					
NRST	Synchronous Microcontroller Reset	I/O	Low	VDDIO	Permanent Internal Pull-up
TST	Test Select	Input			Permanent Internal Pull-down
<b>Universal Asynchronous Receiver Transceiver - UARTx</b>					
URXDx	UART Receive Data	Input			
UTXDx	UART Transmit Data	Output			
<b>PIO Controller - PIOA - PIOB - PIOC- PIOD - PIOE</b>					
PA0 - PA31	Parallel IO Controller A	I/O		VDDIO	Reset State: - PIO or System IOs <sup>(4)</sup> - Internal Pull-up enabled - Schmitt Trigger enabled <sup>(2)</sup>
PB0 - PB14	Parallel IO Controller B	I/O			
PC0 - PC31	Parallel IO Controller C	I/O			
PD0 - PD31	Parallel IO Controller D	I/O			
PE0 - PE5	Parallel IO Controller E	I/O			Reset State: - PIO or System IOs <sup>(4)</sup> - Internal Pull-up enabled - Schmitt Trigger enabled <sup>(2)</sup>
<b>PIO Controller - Parallel Capture Mode</b>					
PIODC0-PIODC7	Parallel Capture Mode Data	Input		VDDIO	
PIODCCLK	Parallel Capture Mode Clock	Input			
PIODCEN1-2	Parallel Capture Mode Enable	Input			
<b>High Speed Multimedia Card Interface - HSMCI</b>					
MCCK	Multimedia Card Clock	I/O			
MCCDA	Multimedia Card Slot A Command	I/O			
MCDA0 - MCDA3	Multimedia Card Slot A Data	I/O			
<b>Universal Synchronous Asynchronous Receiver Transmitter USARTx</b>					
SCKx	USARTx Serial Clock	I/O			
TXDx	USARTx Transmit Data	I/O			
RXDx	USARTx Receive Data	Input			
RTSx	USARTx Request To Send	Output			
CTSx	USARTx Clear To Send	Input			
DTR1	USART1 Data Terminal Ready	I/O			
DSR1	USART1 Data Set Ready	Input			
DCD1	USART1 Data Carrier Detect	Output			
RI1	USART1 Ring Indicator	Input			
<b>Timer/Counter - TC</b>					
TCLKx	TC Channel x External Clock Input	Input			
TIOAx	TC Channel x I/O Line A	I/O			
TIOBx	TC Channel x I/O Line B	I/O			

Table 3-1. Signal Description List

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
<b>Serial Peripheral Interface - SPI</b>					
MISO	Master In Slave Out	I/O			
MOSI	Master Out Slave In	I/O			
SPCK	SPI Serial Clock	I/O			
SPI_NPCS0	SPI Peripheral Chip Select 0	I/O	Low		
SPI_NPCS1 - SPI_NPCS3	SPI Peripheral Chip Select	Output	Low		
<b>Two-Wire Interface - TWIx</b>					
TWDx	TWIx Two-wire Serial Data	I/O			
TWCKx	TWIx Two-wire Serial Clock	I/O			
<b>Analog</b>					
ADVREF	ADC, DAC and Analog Comparator Reference	Analog			
<b>12-bit Analog-Front-End - AFEx</b>					
AFE0_ADO-AFE0_AD14	Analog Inputs	Analog, Digital			
AFE1_ADO-AFE1_AD7	Analog Inputs	Analog, Digital			
ADTRG	Trigger	Input		VDDIO	
<b>12-bit Digital-to-Analog Converter - DAC</b>					
DAC0 - DAC1	Analog output	Analog, Digital			
DACTRGS	DAC Trigger	Input		VDDIO	
<b>Fast Flash Programming Interface - FFPI</b>					
PGMEN0-PGMEN2	Programming Enabling	Input		VDDIO	
PGMM0-PGMM3	Programming Mode	Input		VDDIO	
PGMD0-PGMD15	Programming Data	I/O			
PGMRDY	Programming Ready	Output	High		
PGMNVALID	Data Direction	Output	Low		
PGMNOE	Programming Read	Input	Low		
PGMCK	Programming Clock	Input			
PGMNCMD	Programming Command	Input	Low		
<b>External Bus Interface</b>					
D0 - D7	Data Bus	I/O			
A0 - A23	Address Bus	Output			
NWAIT	External Wait Signal	Input	Low		
<b>Static Memory Controller - SMC</b>					
NCS0 - NCS3	Chip Select Lines	Output	Low		
NRD	Read Signal	Output	Low		

**Table 3-1. Signal Description List**

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
NWE	Write Enable	Output	Low		
<b>NAND Flash Logic</b>					
NANDOE	NAND Flash Output Enable	Output	Low		
NANDWE	NAND Flash Write Enable	Output	Low		
<b>Pulse Width Modulation Controller- PWMC</b>					
PWMH	PWM Waveform Output High for channel x	Output			
PWML	PWM Waveform Output Low for channel x	Output			Only output in complementary mode when dead time insertion is enabled.
PWMFI0	PWM Fault Input	Input			
<b>Ethernet MAC 10/100 -EMAC</b>					
GREFCK	Reference Clock	Input			
GTXCK	Transmit Clock	Input			
GRXCK	Receive Clock	Input			
GTXEN	Transmit Enable	Output			
GTX0 - GTX3	Transmit Data	Output			
GTXER	Transmit Coding Error	Output			
GRXDV	Receive Data Valid	Input			
GRX0 - GRX3	Receive Data	Input			
GRXER	Receive Error	Input			
GCRS	Carrier Sense	Input			
GCOL	Collision Detected	Input			
GMDC	Management Data Clock	Output			
GMDIO	Management Data Input/Output	I/O			
<b>Controller Area Network-CAN (x=[0:1])</b>					
CANRXx	CAN Receive	Input			
CANTTx	CAN Transmit	Output			
<b>USB Full Speed Device</b>					
DDM	DDM USB Full Speed Data -	Analog, Digital			Reset State: - USB Mode - Internal Pull-down <sup>(1)</sup>
DDP	DDP USB Full Speed Data +				Reset State: - USB Mode - Internal Pull-down <sup>(1)</sup>

- Notes:
1. See "Typical Powering Schematics" section of this datasheet for restrictions on voltage range of Analog Cells.
  2. Schmitt Triggers can be disabled through PIO registers.
  3. TDO pin is set in input mode when the Cortex-M4 Core is not in debug mode. Thus the internal pull-up corresponding to this PIO line must be enabled to avoid current consumption due to floating input.
  4. Some PIO lines are shared with System I/Os.

## 4. Package and Pinout

The SAM4E is available in TFBGA100, LFBGA144, LQFP100, and LQFP144 and packages described in the “SAM4E Mechanical Characteristics” section of this datasheet.

### 4.1 100-ball TFBGA Package and Pinout

#### 4.1.1 100-ball TFBGA Package Outline

The 100-ball TFBGA package has a 0.8 mm ball pitch and respects Green Standards. Refer to [Section 47.1 “100-ball TFBGA Package Drawing”](#) for details.

#### 4.1.2 100-ball TFBGA Pinout

**Table 4-1. SAM4E 100-ball TFBGA Pinout**

A1	PB9	C6	PD29	F1	PA19/PGMD11	H6	PA14/PGMD6
A2	PB8	C7	PA30	F2	PA20/PGMD12	H7	PA25
A3	PB14	C8	PB5	F3	PD23	H8	PA27
A4	PB10	C9	PD10	F4	GNDIO	H9	PA5/PGMN1
A5	PD4	C10	PA1/PGMRDY	F5	GNDCORE	H10	PA4/PGMN0
A6	PD7	D1	ADVREF	F6	GNDIO	J1	PA21/PGMD13
A7	PA31	D2	PD1	F7	TST	J2	PA7/PGMN3
A8	PA6/PGMN2	D3	GNDCORE	F8	PB12	J3	PA22/PGMD14
A9	PA28	D4	GNDCORE	F9	PA3/PGMNVALID	J4	PD22
A10	JTAGSEL	D5	PD5	F10	PD14	J5	PA16/PGMD8
B1	PD31	D6	VDDCORE	G1	PA17/PGMD9	J6	PA15/PGMD7
B2	PB13	D7	VDDCORE	G2	PA18/PGMD10	J7	PD28
B3	VDDPLL	D8	PA0/PGMNCMD	G3	PD26	J8	PA11/PGMD3
B4	PB11	D9	PD11	G4	PD24	J9	PA9/PGMD1
B5	PD3	D10	PA2/PGMNOE	G5	PA13/PGMD5	J10	PD17
B6	PD6	E1	PB0	G6	VDDCORE	K1	PD30
B7	PD8	E2	PB1	G7	VDDIO	K2	PA8/PGMD0
B8	PD9	E3	PD2	G8	PB6	K3	PD20
B9	PB4	E4	GNDANA	G9	PD16	K4	PD19
B10	PD15	E5	VDDIO	G10	NRST	K5	PA23/PGMD15
C1	PD0	E6	VDDIO	H1	PB2	K6	PD18
C2	VDDIN	E7	GNDIO	H2	PB3	K7	PA24
C3	VDDOUT	E8	PD13	H3	PD25	K8	PA26
C4	GNDPLL	E9	PB7	H4	PD27	K9	PA10/PGMD2
C5	PA29	E10	PD12	H5	PD21	K10	PA12/PGMD4

## 4.2 144-ball LFBGA Package and Pinout

### 4.2.1 144-ball LFBGA Package Outline

The 144-ball LFBGA package has a 0.8 mm ball pitch and respects Green Standards. Refer to [Section 47.2 “144-ball LFBGA Package Drawing”](#) for details.

### 4.2.2 144-ball LFBGA Pinout

**Table 4-2. SAM4E 144-ball LFBGA Pinout**

A1	PE1	D1	ADVREF	G1	PC15	K1	PE4
A2	PB9	D2	GNDANA	G2	PC13	K2	PA21/PGMD13
A3	PB8	D3	PD31	G3	PB1	K3	PA22/PGMD14
A4	PB11	D4	PD0	G4	GNDIO	K4	PC2
A5	PD2	D5	GNDPLL	G5	GNDIO	K5	PA16/PGMD8
A6	PA29	D6	PD4	G6	GNDIO6	K6	PA14/PGMD6
A7	PC21	D7	PD5	G7	GNDCORE	K7	PC6
A8	PD6	D8	PC19	G8	VDDIO	K8	PA25
A9	PC20	D9	PD9	G9	PD13	K9	PD20
A10	PA30	D10	PD29	G10	PD12	K10	PD28
A11	PD15	D11	PC16	G11	PC9	K11	PD16
A12	PB4	D12	PA1/PGMRDY	G12	PB12	K12	PA4/PGMN0
B1	PE2	E1	PC31	H1	PA19/PGMD11	L1	PE5
B2	PB13	E2	PC27	H2	PA18/PGMD10	L2	PA7/PGMN3
B3	VDDPLL	E3	PE3	H3	PA20/PGMD12	L3	PC3
B4	PB10	E4	PC0	H4	PB0	L4	PA23/PGMD15
B5	PD1	E5	GNDCORE	H5	VDDCORE	L5	PA15/PGMD7
B6	PC24	E6	GNDCORE	H6	VDDIO	L6	PD26
B7	PD3	E7	VDDIO	H7	VDDIO	L7	PA24
B8	PD7	E8	VDDCORE	H8	VDDCORE	L8	PC5
B9	PA6/PGMN2	E9	PD8	H9	PD21	L9	PA10/PGMD2
B10	PC18	E10	PC14	H10	PD14	L10	PA12/PGMD4
B11	JTAGSEL	E11	PD11	H11	TEST	L11	PD17
B12	PC17	E12	PA2/PGMNOE	H12	NRST	L12	PC28
C1	VDDIN	F1	PC30	J1	PA17/PGMD9	M1	PD30
C2	PE0	F2	PC26	J2	PB2	M1	PA8/PGMD0
C3	VDDOUT	F3	PC29	J3	PB3	M3	PA13/PGMD5
C4	PB14	F4	PC12	J4	PC1	M4	PC7
C5	PC25	F5	GNDIO	J5	PC4	M5	PD25
C6	PC23	F6	GNDIO	J6	PD27	M6	PD24
C7	PC22	F7	GNDCORE	J7	VDDCORE	M7	PD23
C8	PA31	F8	VDDIO	J8	PA26	M8	PD22
C9	PA28	F9	PB7	J9	PA11/PGMD3	M9	PD19
C10	PB5	F10	PC10	J10	PA27	M10	PD18
C11	PA0/PGMNCMD	F11	PC11	J11	PB6	M11	PA5/PGMN1
C12	PD10	F12	PA3/PGMNVALID	J12	PC8	M12	PA9/PGMD1

## 4.3 100-lead LQFP Package and Pinout

### 4.3.1 100-lead LQFP Package Outline

The 100-lead LQFP package has a 0.5 mm ball pitch and respects Green Standards. Please refer to [Section 47.3 “100-lead LQFP Package Drawing”](#) for details.

### 4.3.2 100-lead LQFP Pinout

**Table 4-3. SAM4E 100-lead LQFP Pinout**

1	PD0	26	PA22/PGMD14	51	PD28	76	PD29
2	PD31	27	PA13/PGMD5	52	PA5/PGMN1	77	PB5
3	GND	28	VDDIO	53	PD17	78	PD9
4	VDDOUT	29	GND	54	PA9/PGMD1	79	PA28
5	VDDIN	30	PA16/PGMD8	55	PA4/PGMN0	80	PD8
6	GND	31	PA23/PGMD15	56	PD16	81	PA6/PGMN2
7	GND	32	PD27	57	PB6	82	PA30
8	GND	33	PA15/PGMD7	58	NRST	83	PA31
9	ADVREF	34	PA14/PGMD6	59	PD14	84	PD7
10	GND	35	PD25	60	TST	85	PD6
11	PB1	36	PD26	61	PB12	86	VDDCORE
12	PB0	37	PD24	62	PD13	87	PD5
13	PA20/PGMD12	38	PA24	63	PB7	88	PD4
14	PA19/PGMD11	39	PD23	64	PA3/PGMNVALID	89	PD3
15	PA18/PGMD10	40	PA25	65	PD12	90	PA29
16	PA17/PGMD9	41	PD22	66	PA2/PGMNOE	91	PD2
17	PB2	42	PA26	67	GND	92	PD1
18	VDDCORE	43	PD21	68	VDDIO	93	VDDIO
19	VDDIO	44	PA11/PGMD3	69	PD11	94	PB10
20	PB3	45	PD20	70	PA1/PGMRDY	95	PB11
21	PA21/PGMD13	46	PA10/PGMD2	71	PD10	96	VDDPLL
22	VDDCORE	47	PD19	72	PA0	97	PB14
23	PD30	48	PA12/PGMD4	73	JTAGSEL	98	PB8
24	PA7/PGMN3	49	PD18	74	PB4	99	PB9
25	PA8/PGMD0	50	PA27	75	PD15	100	PB13

## 4.4 144-lead LQFP Package and Pinout

### 4.4.1 144-lead LQFP Package Outline

The 144-lead LQFP package has a 0.5 mm ball pitch and respects Green Standards. Please refer to [Section 47.4 “144-lead LQFP Package Drawing”](#) for details.

### 4.4.2 144-lead LQFP Pinout

**Table 4-4. SAM4E 144-lead LQFP Pinout**

1	PD0	37	PA22/PGMD14	73	PA5/PGMN1	109	PB5
2	PD31	38	PC1	74	PD17	110	PD9
3	VDDOUT	39	PC2	75	PA9/PGMD1	111	PC18
4	PE0	40	PC3	76	PC28	112	PA28
5	VDDIN	41	PC4	77	PA4/PGMN0	113	PD8
6	PE1	42	PA13/PGMD5	78	PD16	114	PA6/PGMN2
7	PE2	43	VDDIO	79	PB6	115	GND
8	GND	44	GND	80	VDDIO	116	PA30
9	ADVREFP	45	PA16/PGMD8	81	VDDCORE	117	PC19
10	PE3	46	PA23/PGMD15	82	PC8	118	PA31
11	PC0	47	PD27	83	NRST	119	PD7
12	PC27	48	PC7	84	PD14	120	PC20
13	PC26	49	PA15/PGMD7	85	TEST	121	PD6
14	PC31	50	VDDCORE	86	PC9	122	PC21
15	PC30	51	PA14/PGMD6	87	PB12	123	VDDCORE
16	PC29	52	PD25	88	PD13	124	PC22
17	PC12	53	PD26	89	PB7	125	PD5
18	PC15	54	PC6	90	PC10	126	PD4
19	PC13	55	PD24	91	PA3	127	PC23
20	PB1	56	PA24	92	PD12	128	PD3
21	PB0	57	PD23	93	PA2	129	PA29
22	PA20/PGMD12	58	PC5	94	PC11	130	PC24
23	PA19/PGMD11	59	PA25	95	GND	131	PD2
24	PA18/PGMD10	60	PD22	96	VDDIO	132	PD1
25	PA17/PGMD9	61	GND	97	PC14	133	PC25
26	PB2	62	PA26	98	PD11	134	VDDIO
27	PE4	63	PD21	99	PA1	135	GND
28	PE5	64	PA11/PGMD3	100	PC16	136	PB10
29	VDDCORE	65	PD20	101	PD10	137	PB11
30	VDDIO	66	PA10/PGMD2	102	PA0	138	GND
31	PB3	67	PD19	103	PC17	139	VDDPLL
32	PA21/PGMD13	68	PA12/PGMD4	104	JTAGSEL	140	PB14
33	VDDCORE	69	PD18	105	PB4	141	PB8
34	PD30	70	PA27	106	PD15	142	PB9
35	PA7/PGMN3	71	PD28	107	VDDCORE	143	VDDIO
36	PA8/PGMD0	72	VDDIO	108	PD29	144	PB13

## 5. Power Considerations

### 5.1 Power Supplies

The SAM4E has several types of power supply pins:

- VDDCORE pins: power the core, the first flash rail, the embedded memories and the peripherals.  
Voltage ranges from 1.08V to 1.32V.
- VDDIO pins: power the peripheral I/O lines (Input/Output Buffers), the second flash rail, the backup part, the USB transceiver, 32 kHz crystal oscillator and oscillator pads.  
Voltage ranges from 1.62V to 3.6V.
- VDDIN pins: voltage regulator input, DAC and Analog Comparator power supply.  
Voltage ranges from 1.62V to 3.6V.
- VDDPLL pin: powers the PLL, the Fast RC and the 3 to 20 MHz oscillator.  
Voltage ranges from 1.08V to 1.32V.

### 5.2 Voltage Regulator

The SAM4E embeds a voltage regulator that is managed by the Supply Controller.

This internal regulator is designed to supply the internal core of SAM4E It features two operating modes:

- In normal mode, the voltage regulator consumes less than 500  $\mu$ A static current and draws 80 mA of output current. Internal adaptive biasing adjusts the regulator quiescent current depending on the required load current. In wait mode quiescent current is only 5  $\mu$ A.
- In backup mode, the voltage regulator consumes less than 1.5  $\mu$ A while its output (VDDOUT) is driven internally to GND. The default output voltage is 1.20V and the start-up time to reach normal mode is less than 300  $\mu$ s.

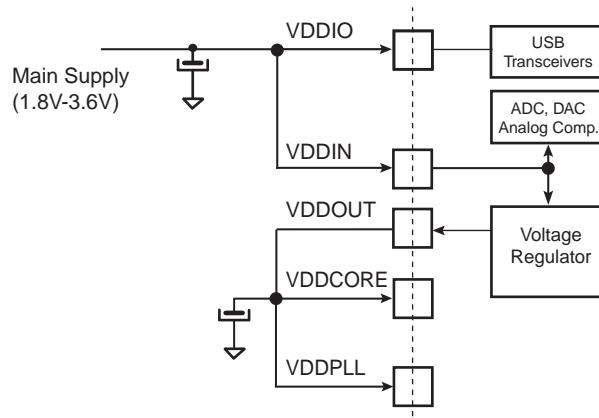
For adequate input and output power supply decoupling/bypassing, refer to the “Voltage Regulator” section in the “Electrical Characteristics” section of the datasheet.

### 5.3 Typical Powering Schematics

The SAM4E supports a 1.62V-3.6V single supply mode. The internal regulator input is connected to the source and its output feeds VDDCORE. [Figure 5-1](#) below shows the power schematics.

As VDDIN powers the voltage regulator, the DAC and the analog comparator, when the user does not want to use the embedded voltage regulator, it can be disabled by software via the SUPC (note that this is different from backup mode).

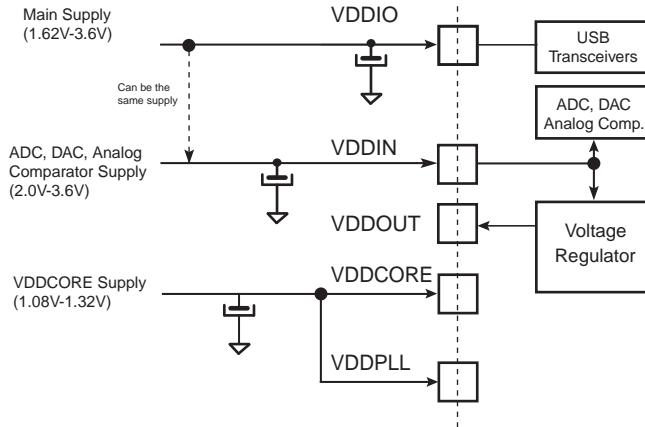
**Figure 5-1. Single Supply**



Note: Restrictions:

- For USB, VDDIO needs to be greater than 3.0V
- For AFE, VDDIN needs to be greater than 2.0V
- For DAC, VDDIN needs to be greater than 2.4V

**Figure 5-2. Core Externally Supplied**



Note: Restrictions:

- For USB, VDDIO needs to be greater than 3.0V.
- For AFE, VDDIN needs to be greater than 2.0V
- For DAC, VDDIN needs to be greater than 2.4V

## 5.4 Active Mode

Active mode is the normal running mode with the core clock running from the fast RC oscillator, the main crystal oscillator or the PLLA. The power management controller can be used to adapt the frequency and to disable the peripheral clocks.

## 5.5 Low-power Modes

The SAM4E has the following low-power modes: backup mode, wait mode and sleep mode.

Note: The Wait For Event instruction (WFE) of the Cortex-M4 core can be used to enter any of the low-power modes, however, this may add complexity in the design of application state machines. This is due to the fact that the WFE instruction goes along with an event flag of the Cortex core (cannot be managed by the software application). The event flag can be set by interrupts, a debug event or an event signal from another processor. Since it is possible for an interrupt to occur just before the execution of WFE, WFE takes into account events that happened in the past. As a result, WFE prevents the device from entering wait mode if an interrupt event has occurred.

Atmel has made provision to avoid using the WFE instruction. The workarounds to ease application design are as follows:

- For backup mode, switch off the voltage regulator and configure the VROFF bit in the Supply Controller Control Register (SUPC\_CR).
- For wait mode, configure the WAITMODE bit in the PMC Clock Generator Main Oscillator Register of the Power Management Controller (PMC)
- For sleep mode, use the Wait For Interrupt (WFI) instruction.

Complete information is available in [Table 5-1 “Low-power Mode Configuration Summary”](#).

### 5.5.1 Backup Mode

The purpose of backup mode is to achieve the lowest power consumption possible in a system which is performing periodic wake-ups to perform tasks but not requiring fast startup time. Total current consumption is 1  $\mu$ A typical (VDDIO = 1.8 V to 25°).

The Supply Controller, zero-power power-on reset, RTT, RTC, backup registers and 32 kHz oscillator (RC or crystal oscillator selected by software in the Supply Controller) are running. The regulator and the core supply are off.

The SAM4E can be awakened from this mode using the WKUP0-15 pins, the supply monitor (SM), the RTT or RTC wake-up event.

Backup mode is entered by writing a 1 to the VROFF bit of the Supply Controller Control register (SUPC\_CR) (A key is needed to write the VROFF bit, refer to the Supply Controller SUPC section of the product datasheet) and with the

SLEEPDEEP bit in the Cortex-M4 System Control register set to 1. (See the power management description in the ARM Cortex-M4 Processor section of the product datasheet).

To enter backup mode using the VROFF bit:

- Write a 1 to the VROFF bit of SUPC\_CR.

To enter backup mode using the WFE instruction:

- Write a 1 to the SLEEPDEEP bit of the Cortex-M4 processor.
- Execute the WFE instruction of the processor.

In both cases, exit from backup mode happens if one of the following enable wake up events occurs:

- WKUPEN0-15 pins (level transition, configurable debouncing)
- Supply Monitor alarm
- RTC alarm
- RTT alarm

### 5.5.2 Wait Mode

The purpose of wait mode is to achieve very low power consumption while maintaining the whole device in a powered state for a startup time of less than 10 µs. Current consumption in wait mode is typically 32 µA (total current consumption) if the internal voltage regulator is used.

In this mode, the clocks of the core, peripherals and memories are stopped. However, the core, peripherals and memories power supplies are still powered. From this mode, a fast start up is available.

This mode is entered by setting the WAITMODE bit to 1 in the CKGR\_MOR register in conjunction with FLPM = 0 or FLPM = 1 bits of the PMC\_FSMR register or by the WFE instruction.

The Cortex-M4 is able to handle external or internal events in order to wake-up the core. This is done by configuring the external lines WUP0-15 as fast startup wake-up pins (refer to [Section 5.7 “Fast Start-up”](#)). RTC or RTT Alarm and USB wake-up events can be used to wake up the CPU.

To enter wait mode with WAITMODE bit:

- Select the 4/8/12 MHz fast RC oscillator as Main Clock.
- Set the FLPM field in the PMC Fast Startup Mode Register (PMC\_FSMR).
- Set Flash Wait State at 0.
- Set the WAITMODE bit = 1 in PMC Main Oscillator Register (CKGR\_MOR).
- Wait for Master Clock Ready MCKRDY = 1 in the PMC Status Register (PMC\_SR).

To enter wait mode with WFE:

- Select the 4/8/12 MHz fast RC oscillator as Main Clock.
- Set the FLPM field in the PMC Fast Startup Mode Register (PMC\_FSMR).
- Set Flash Wait State at 0.
- Set the LPM bit in the PMC Fast Startup Mode Register (PMC\_FSMR).
- Execute the Wait-For-Event (WFE) instruction of the processor.

In both cases, depending on the value of the field Flash Low Power Mode (FLPM), the Flash enters three different modes:

- FLPM = 0 in Standby mode (Low consumption)
- FLPM = 1 in Deep power-down mode (Extra low consumption)
- FLPM = 2 in Idle mode. Memory ready for Read access

[Table 5-1](#) summarizes the power consumption, wake-up time and system state in wait mode.

### 5.5.3 Sleep Mode

The purpose of sleep mode is to optimize power consumption of the device versus response time. In this mode, only the core clock is stopped. The peripheral clocks can be enabled. The current consumption in this mode is application dependent.

This mode is entered via WFI or WFE instructions.

The processor can be awakened from an interrupt if the WFI instruction of the Cortex-M4 is used or from an event if the WFE instruction is used.

### 5.5.4 Low-power Mode Summary Table

The modes detailed above are the main low-power modes. Each part can be set to on or off separately and wake-up sources can be individually configured. [Table 5-1](#) below provides the configuration summary of the low-power modes.

**Table 5-1. Low-power Mode Configuration Summary**

Mode	SUPC, 32 kHz Oscillator, RTC, RTT Backup Registers, POR (Backup Region)	Regulator	Core Memory Peripherals	Mode Entry	Potential Wake-Up Sources	Core at Wake-Up	PIO State while in Low- Power Mode	PIO State at Wake Up	Consumption <sup>(2) (3)</sup>	Wake-up Time <sup>(1)</sup>
Backup Mode	ON	OFF	OFF (Not powered)	VROFF = 1 or WFE + SLEEPDEEP = 1	WUP0-15 pins SM alarm RTC alarm RTT alarm	Reset	Previous state saved	PIOA & PIOB & PIOC & PIOD & PIOE Inputs with pull-ups	1 µA typ <sup>(4)</sup>	< 1 ms
Wait Mode w/Flash in Standby Mode	ON	ON	Powered (Not clocked)	WAITMODE = 1 + FLPM = 0 or WFE + SLEEPDEEP = 0 + LPM = 1 + FLPM = 0	Any Event from: Fast startup through WUP0-15 RTC alarm RTT alarm USB wake-up	Clocked back	Previous state saved	Unchanged	56 µA <sup>(5)</sup>	10 us
Wait Mode w/Flash in Deep Power-down Mode	ON	ON	Powered (Not clocked)	WAITMODE = 1 + FLPM = 1 or WFE + SLEEPDEEP = 0 + LPM = 1 + FLPM = 1	Any Event from: Fast startup through WUP0-15 RTC alarm RTT alarm USB wake-up	Clocked back	Previous state saved	Unchanged	46.6 µA	< 100 us
Sleep Mode	ON	ON	Powered <sup>(7)</sup> / (N ot clocked)	WFE or WFI + SLEEPDEEP = 0 + LPM = 0	Entry mode =WFI Interrupt Only; Entry mode =WFE Any Enabled Interrupt and/or Any Event from: Fast start-up through WUP0-15 RTC alarm RTT alarm USB wake-up	Clocked back	Previous state saved	Unchanged	(6)	(6)

Notes:

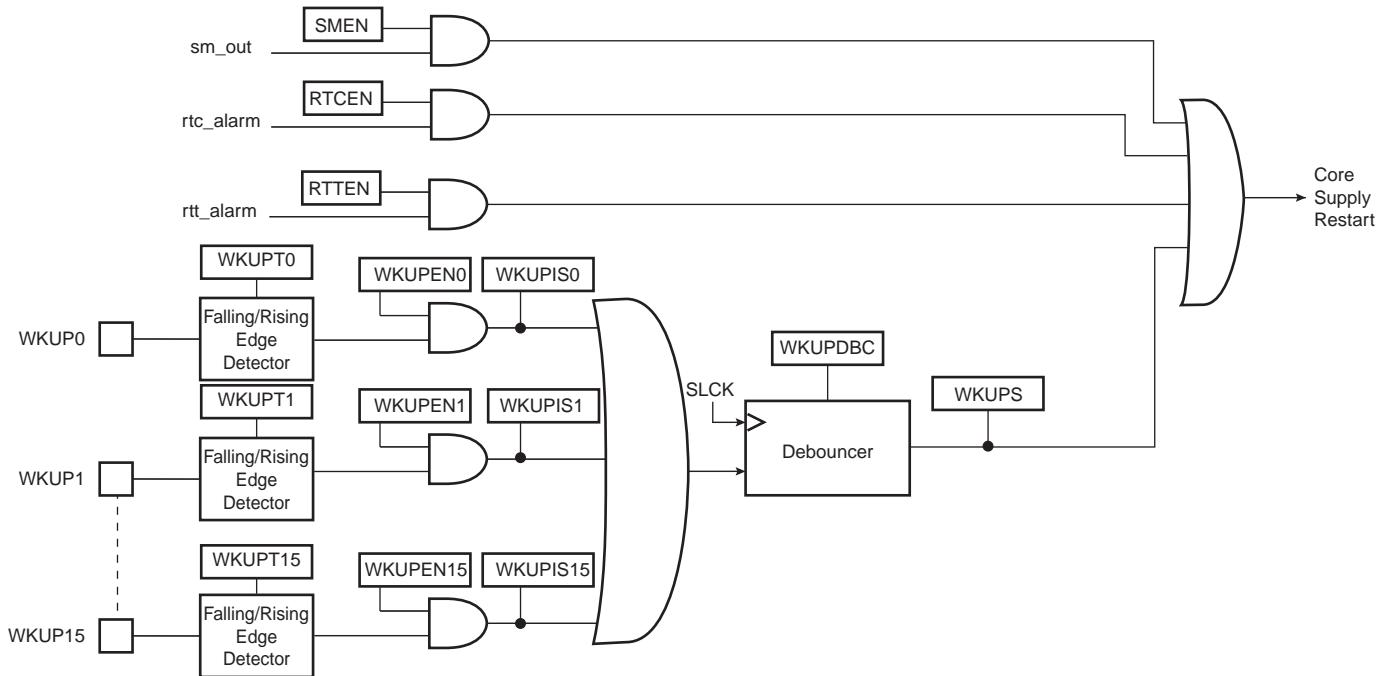
- When considering wake-up time, the time required to start the PLL is not taken into account. Once started, the device works with the 4/8/12 MHz fast RC oscillator. The user has to add the PLL start-up time if it is needed in the system. The wake-up time is defined as the time taken for wake up until the first instruction is fetched.

- The external loads on PIOs are not taken into account in the calculation.
- Supply Monitor current consumption is not included.
- Total consumption 1µA typ to 1.8V on VDDCORE to 25°.
- Power consumption on VDDCORE. For total current consumption, please refer to the "Electrical Characteristics" section of this datasheet.
- Depends on MCK frequency.
- In this mode the core is supplied and not clocked but some peripherals can be clocked.

## 5.6 Wake-up Sources

The wake-up events allow the device to exit the backup mode. When a wake-up event is detected, the Supply Controller performs a sequence which automatically reenables the core power supply and the SRAM power supply, if they are not already enabled.

**Figure 5-3. Wake-up Source**

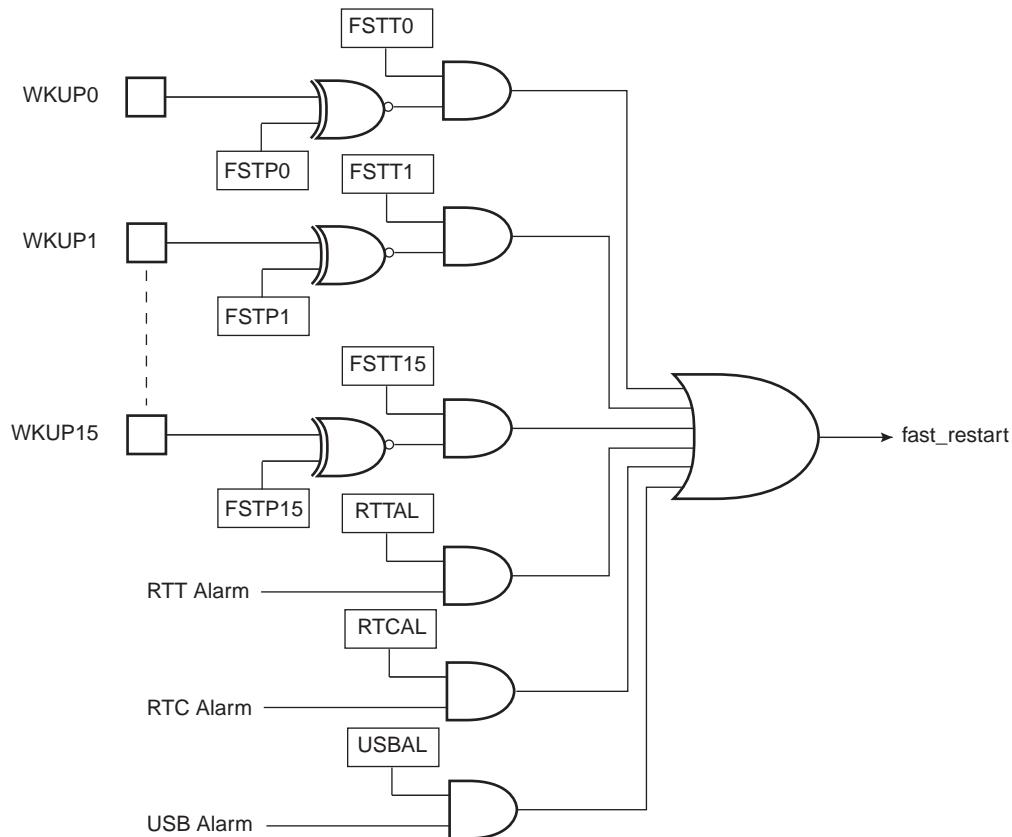


## 5.7 Fast Start-up

The SAM4E allows the processor to restart in a few microseconds while the processor is in wait mode or in sleep mode. A fast start-up can occur upon detection of a low level on one of the 19 wake-up inputs (WKUP0 to 15 + RTC + RTT + USB).

The fast restart circuitry, as shown in [Figure 5-4](#), is fully asynchronous and provides a fast start-up signal to the Power Management Controller. As soon as the fast start-up signal is asserted, the PMC automatically restarts the embedded 4/8/12 MHz Fast RC oscillator, switches the master clock on this 4 MHz clock and reenables the processor clock.

**Figure 5-4. Fast Start-up Sources**



## 6. Input/Output Lines

The SAM4E has several kinds of input/output (I/O) lines such as general-purpose I/Os (GPIO) and system I/Os. GPIOs can have alternate functionality due to multiplexing capabilities of the PIO controllers. The same PIO line can be used whether in I/O mode or by the multiplexed peripheral. System I/Os include pins such as test pins, oscillators, erase or analog inputs.

### 6.1 General-purpose I/O Lines

GPIO lines are managed by PIO Controllers. All I/Os have several input or output modes such as pull-up or pull-down, input Schmitt triggers, multi-drive (open-drain), glitch filters, debouncing or input change interrupt. Programming of these modes is performed independently for each I/O line through the PIO controller user interface. For more details, refer to the product “PIO Controller” section.

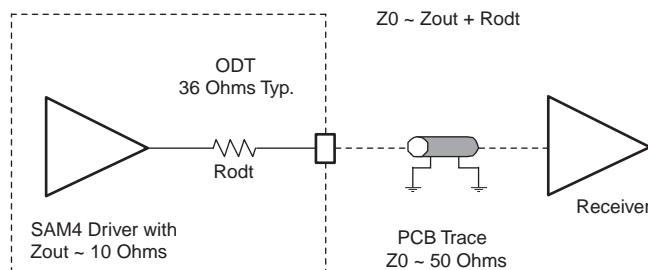
Some GPIOs can have an alternate function as analog input. When a GPIO is set in analog mode, all digital features of the I/O are disabled.

The input/output buffers of the PIO lines are supplied through VDDIO power supply rail.

The SAM4E device embeds high speed pads able. See “AC Characteristics” section of the datasheet for more details. Typical pull-up and pull-down value is 100 kΩ for all I/Os.

Each I/O line also embeds an ODT (On-Die Termination), (see [Figure 6-1](#) below). It consists of an internal series resistor termination scheme for impedance matching between the driver output (SAM4E) and the PCB trace impedance preventing signal reflection. The series resistor helps to reduce IOs switching current ( $di/dt$ ) thereby reducing in turn, EMI. It also decreases overshoot and undershoot (ringing) due to inductance of interconnect between devices or between boards. In conclusion, ODT helps diminish signal integrity issues.

**Figure 6-1. On-die Termination**



## 6.2 System I/O Lines

System I/O lines are pins used by oscillators, test mode, reset and JTAG to name but a few. Described below in [Table 6-1](#) are the SAM4E system I/O lines shared with PIO lines.

These pins are software configurable as general-purpose I/O or system pins. At start-up, the default function of these pins is always used.

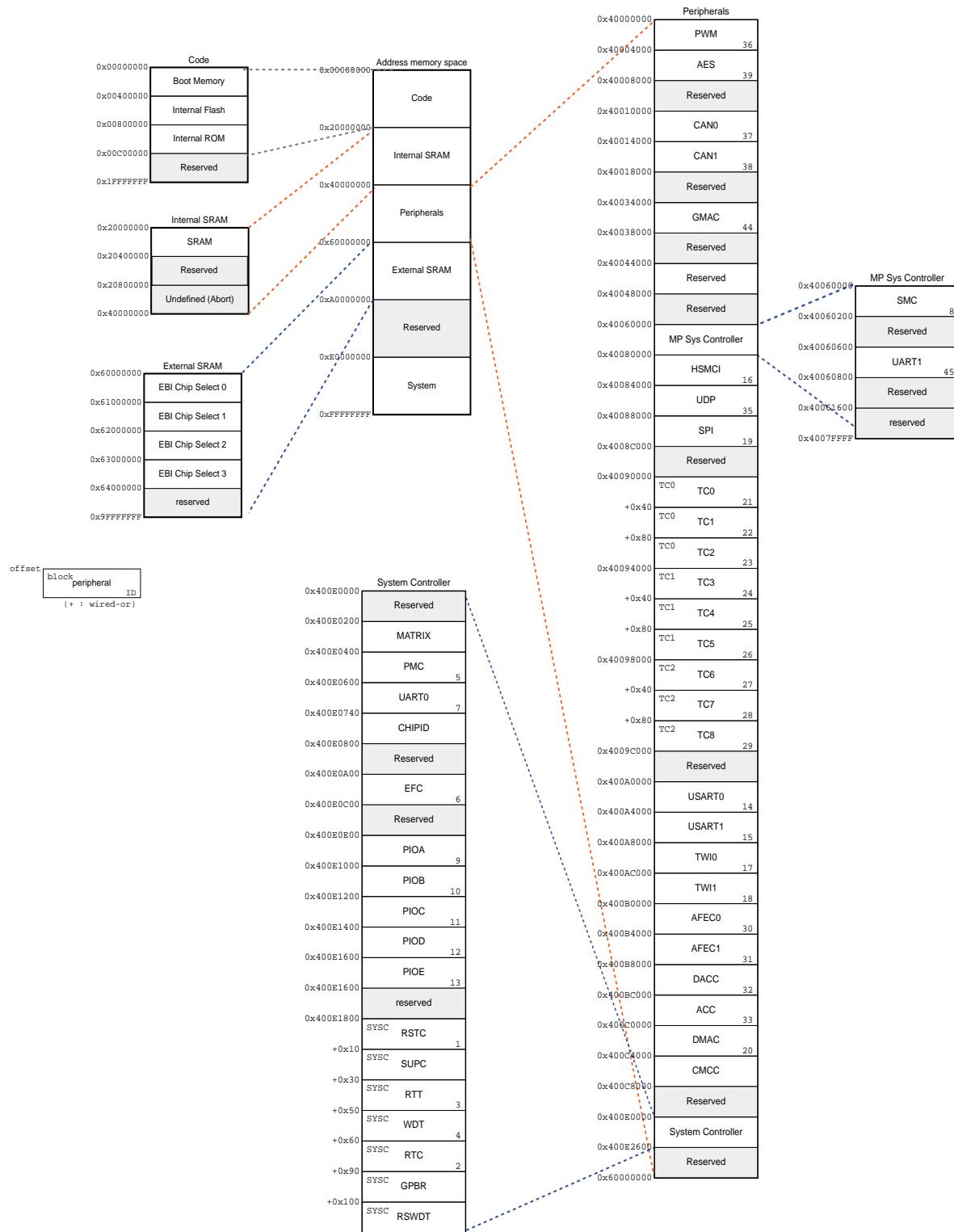
**Table 6-1. System I/O Configuration Pin List.**

SYSTEM_IO Bit Number	Default Function after Reset	Other Function	Constraints for Normal Start	Configuration
12	ERASE	PB12	Low Level at start-up <sup>(1)</sup>	In Matrix User Interface registers (Refer to the System I/O Configuration Register in the "Bus Matrix" section of this datasheet.)
7	TCK/SWCLK	PB7	-	
6	TMS/SWDIO	PB6	-	
5	TDO/TRACESWO	PB5	-	
4	TDI	PB4	-	
-	PA7	XIN32	-	
-	PA8	XOUT32	-	
-	PB9	XIN	-	
-	PB8	XOUT	-	See footnote <sup>(3)</sup> below

- Notes:
1. If PB12 is used as PIO input in user applications, a low level must be ensured at startup to prevent Flash erase before the user application sets PB12 into PIO mode,
  2. In the product Datasheet Refer to: "Slow Clock Generator" of the "Supply Controller" section.
  3. In the product Datasheet Refer to: "3 to 20 MHz Crystal Oscillator" information in the "PMC" section.

## 7. Product Mapping

Figure 7-1. SAM4E Product Mapping



## 8. Memories

### 8.1 Embedded Memories

#### 8.1.1 Internal SRAM

The SAM4E device (1024 Kbytes) embeds a total of 128 kbytes of high-speed SRAM.

The SRAM is accessible over System Cortex-M4 bus at address 0x2000\_0000.

The SRAM is in the bit band region. The bit band alias region is from 0x2200\_0000 to 0x23FF\_FFFF.

#### 8.1.2 Internal ROM

The SAM4E device embeds an internal ROM, which contains the SAM Boot Assistant (SAM-BA<sup>®</sup>), In Application Programming routines (IAP) and Fast Flash Programming Interface (FFPI).

At any time, the ROM is mapped at address 0x0080 0000.

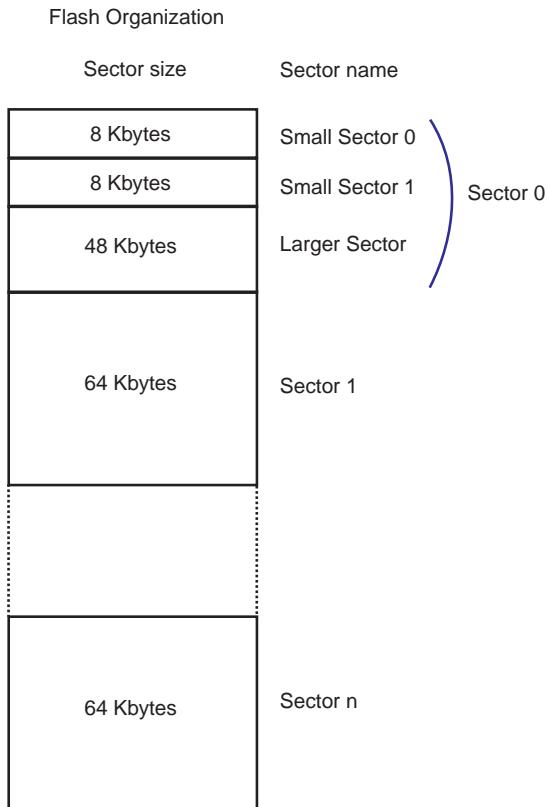
#### 8.1.3 Embedded Flash

##### 8.1.3.1 Flash Overview

The memory is organized in sectors. Each sector has a size of 64 kbytes. The first sector of 64 kbytes is divided into three smaller sectors.

The three smaller sectors are organized to consist of two sectors of 8 kbytes and one sector of 48 kbytes. Refer to [Figure 8-1, "Global Flash Organization"](#) below.

**Figure 8-1. Global Flash Organization**



Each Sector is organized in pages of 512 bytes.

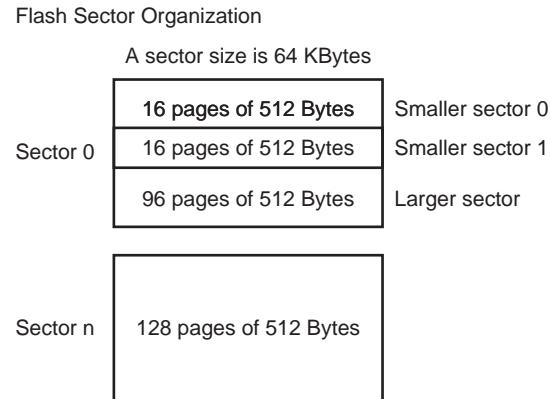
For sector 0:

- The smaller sector 0 has 16 pages of 512 bytes
- The smaller sector 1 has 16 pages of 512 bytes
- The larger sector has 96 pages of 512 bytes

From Sector 1 to n:

The rest of the array is composed of 64 kbyte sector of each 128 pages of 512 bytes. Refer to [Figure 8-2, "Flash Sector Organization"](#) below.

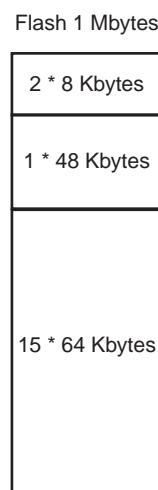
**Figure 8-2. Flash Sector Organization**



Flash size varies by product. The Flash size of SAM4E device is 1024 kbytes.

Refer to [Figure 8-3, "Flash Size"](#) below for the organization of the Flash following its size.

**Figure 8-3. Flash Size**



Erasing the memory can be performed as follows:

- On a 512-byte page inside a sector, of 8 Kbytes
- On a 4-kbyte block inside a sector of 8 Kbytes/48 Kbytes/64 Kbytes
- On a sector of 8 kbytes/48 kbytes/64 kbytes
- On chip

The memory has one additional reprogrammable page that can be used as page signature by the user. It is accessible through specific modes, for erase, write and read operations. Erase pin assertion will not erase the user signature page.

Erase memory by page is possible only in sector of 8 kbytes:

- EWP and EWPL commands can be only used in 8 kbytes sector

The write commands of the flash cannot be used under 330 kHz.

#### 8.1.3.2 Enhanced Embedded Flash Controller

The Enhanced Embedded Flash Controller manages accesses performed by the masters of the system. It enables reading the Flash and writing the write buffer. It also contains a user interface, mapped on the APB.

The Enhanced Embedded Flash Controller ensures the interface of the Flash block.

It manages the programming, erasing, locking and unlocking sequences of the Flash using a full set of commands.

One of the commands returns the embedded Flash descriptor definition that informs the system about the Flash organization, thus making the software generic.

#### 8.1.3.3 Flash Speed

The user needs to set the number of wait states depending on the frequency used:

For more details, refer to the “AC Characteristics” section of the product “Electrical Characteristics”.

Target for the Flash speed at 0 wait state: 24 MHz.

#### 8.1.3.4 Lock Regions

Several lock bits are used to protect write and erase operations on lock regions. A lock region is composed of several consecutive pages, and each lock region has its associated lock bit.

**Table 8-1. Lock bit number**

Product	Number of lock bits	Lock region size
SAM4E	128	8 Kbytes

If a locked region's erase or program command occurs, the command is aborted and the EEFC triggers an interrupt.

The lock bits are software programmable through the EEFC User Interface. The command “Set Lock Bit” enables the protection. The command “Clear Lock Bit” unlocks the lock region.

Asserting the ERASE pin clears the lock bits, thus unlocking the entire Flash.

#### 8.1.3.5 Security Bit Feature

The SAM4E device features a security bit, based on a specific general-purpose NVM bit (GPNVM bit 0). When the security is enabled, any access to the Flash, SRAM, core registers and internal peripherals either through the ICE interface or through the Fast Flash Programming Interface, is forbidden. This ensures the confidentiality of the code programmed in the Flash.

This security bit can only be enabled through the command “Set General Purpose NVM Bit 0” of the EEFC User Interface. Disabling the security bit can only be achieved by asserting the ERASE pin at 1, and after a full Flash erase is performed. When the security bit is deactivated, all accesses to the Flash, SRAM, Core registers, Internal Peripherals are permitted.

It is important to note that the assertion of the ERASE pin should always be longer than 200 ms.

As the ERASE pin integrates a permanent pull-down, it can be left unconnected during normal operation. However, it is safer to connect it directly to GND for the final application.

#### 8.1.3.6 Calibration Bits

NVM bits are used to calibrate the brownout detector and the voltage regulator. These bits are factory configured and cannot be changed by the user. The ERASE pin has no effect on the calibration bits.

#### 8.1.3.7 Unique Identifier

Each device integrates its own 128-bit unique identifier. These bits are factory configured and cannot be changed by the user. The ERASE pin has no effect on the unique identifier.

#### 8.1.3.8 User Signature

Each part contains a user signature of 512 bytes. It can be used by the user to store user information, such as trimming, keys, etc., that the customer does not want to be erased by asserting the ERASE pin or by software ERASE command. Read, write and erase of this area is allowed.

#### 8.1.3.9 Fast Flash Programming Interface

The Fast Flash Programming Interface allows programming the device through a multiplexed fully-handshaked parallel port. It allows gang programming with market-standard industrial programmers.

The FFPI supports read, page program, page erase, full erase, lock, unlock and protect commands.

The Fast Flash Programming Interface is enabled and the Fast Programming mode is entered when TST and PA0 and PA1 are tied low.

#### 8.1.3.10 SAM-BA Boot

The SAM-BA boot is a default boot program which provides an easy way to program in-situ the on-chip Flash memory.

The SAM-BA boot assistant supports serial communication via the UART

The SAM-BA boot provides an interface with SAM-BA Graphic User Interface (GUI).

The SAM-BA boot is in ROM and is mapped in Flash at address 0x0 when GPNVM bit 1 is set to 0.

#### 8.1.3.11 GPNVM Bits

The SAM4E device features two GPNVM bits. These bits can be cleared or set respectively through the commands "Clear GPNVM Bit" and "Set GPNVM Bit" of the EEFC User Interface.

The Flash of SAM4E is composed of 1024 kbytes in a single bank.

**Table 8-2. General-purpose Non-volatile Memory Bits**

GPNVMBit[#]	Function
0	Security bit
1	Boot mode selection

#### 8.1.4 Boot Strategies

The system always boots at address 0x0. To ensure maximum boot possibilities, the memory layout can be changed via GPNVM.

A general-purpose NVM (GPNVM) bit is used to boot either on the ROM (default) or from the Flash.

The GPNVM bit can be cleared or set respectively through the commands "Clear General-purpose NVM Bit" and "Set General-purpose NVM Bit" of the EEFC User Interface.

Setting GPNVM Bit 1 selects the boot from the Flash, clearing it selects the boot from the ROM. Asserting ERASE clears the GPNVM Bit 1 and thus selects the boot from the ROM by default.

## 8.2 External Memories

The SAM4E device features one External Bus Interface to provide an interface to a wide range of external memories and to any parallel peripheral.

## 8.3 Cortex-M Cache Controller (CMCC)

The SAM4E device features one cache memory and his controller which improve code execution when the code runs out of Code section (memory from 0x0 to 0x2000\_0000).

The Cache controller handles both command instructions and data, it is an unified cache:

- L1 data cache size set to 2 Kbytes
- L1 cache line is 16 Bytes
- L1 cache integrates 32 bits bus master interface
- Unified 4-way set associative cache architecture

## 9. Real-time Event Management

The events generated by peripherals are designed to be directly routed to peripherals managing/using these events without processor intervention. Peripherals receiving events contain logic by which to select the one required.

### 9.1 Embedded Characteristics

- Timers, PWM, IO peripherals generate event triggers which are directly routed to event managers such as AFEC or DACC, for example, to start measurement/conversion without processor intervention.
- UART, USART, SPI, TWI, PWM, HSMCI, AES, AFEC, DACC, PIO, TIMER (capture mode) also generate event triggers directly connected to Peripheral DMA Controller (PDC) for data transfer without processor intervention.
- Parallel capture logic is directly embedded in PIO and generates trigger event to Peripheral DMA Controller to capture data without processor intervention.
- PWM security events (faults) are in combinational form and directly routed from event generators (ADC, ACC, PMC, TIMER) to PWM module.
- PWM output comparators generate events directly connected to TIMER.
- PMC security event (clock failure detection) can be programmed to switch the MCK on reliable main RC internal clock without processor intervention.

## 9.2 Real-time Event Mapping List

**Table 9-1.** Real-time Event Mapping List

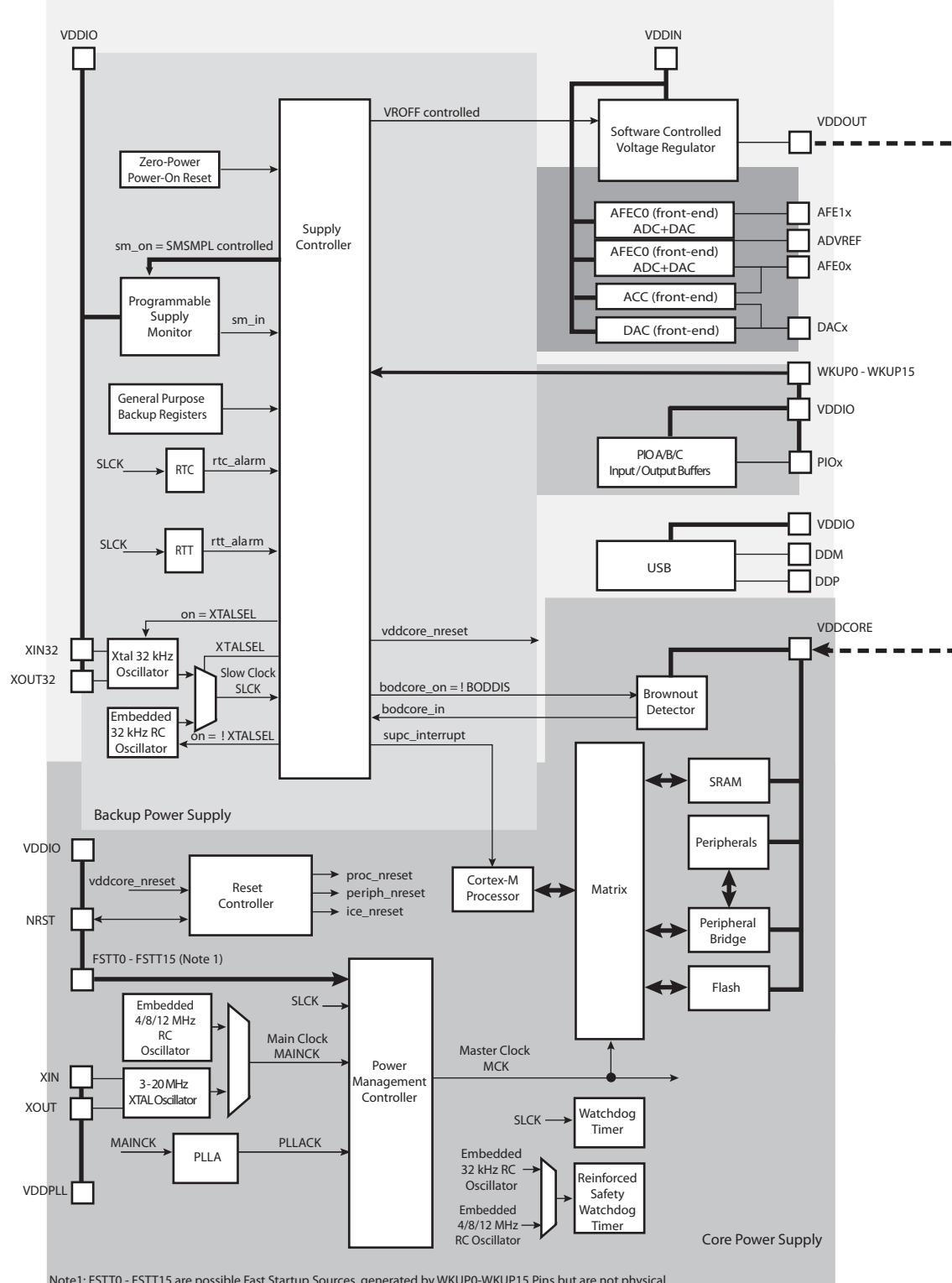
Event Generator	Event Manager	Function
IO (WKUP0/1)	General-purpose Backup Register (GPBR)	Security / Immediate GPBR clear (asynchronous) on Tamper detection through WKUP0/1 IO pins.
Power Management Controller (PMC)	PMC	Safety / Automatic Switch to Reliable Main RC oscillator in case of Main Crystal Clock Failure
PMC	Pulse Width Modulation (PWM)	Safety / Puts the PWM Outputs in Safe Mode (Main Crystal Clock Failure Detection)
Analog Comparator Controller (ACC)	PWM	Safety / Puts the PWM Outputs in Safe Mode (Overcurrent sensor, ...)
Analog-Front-End-Controller (AFEC0/1)	PWM	Safety / Puts the PWM Outputs in Safe Mode (Overspeed, Overcurrent detection ...)
Timer Counter (TC)	PWM	Safety / Puts the PWM Outputs in Safe Mode (Overspeed detection through TIMER Quadrature Decoder)
IO	PWM	Safety / Puts the PWM Outputs in Safe Mode (General-purpose Fault Inputs)
IO	Parallel Capture (PC)	PC is embedded in PIO (Capture Image from Sensor directly to System Memory)
IO (ADTRG)	AFEC	Trigger for measurement. Selection in ADC module
TC Output 0	AFEC	Trigger for measurement. Selection in ADC module
TC Output 1	AFEC	Trigger for measurement. Selection in ADC module
TC Output 2	AFEC	Trigger for measurement. Selection in ADC module
PWM Output Compare Line 0	TC Input (A/B) 0	Allows delay measurement between PWM outputs and TC inputs externally connected to power transistor bridge driver.
PWM Output Compare Line 1	TC Input (A/B) 1	Allows delay measurement between PWM outputs and TC inputs externally connected to power transistor bridge driver
PWM Output Compare Line 2	TC Input (A/B) 2	Allows delay measurement between PWM outputs and TC inputs externally connected to power transistor bridge driver
PWM Event Line 0	AFEC	Trigger for measurement. PWM contains a programmable delay for this trigger.
PWM Event Line 1	AFEC	Trigger for measurement. PWM contains a programmable delay for this trigger.
IO (DATRG)	DACC (Digital-Analog Converter Controller)	Trigger for conversion. Selection in DAC module
TC Output 0	DACC	Trigger for conversion. Selection in DAC module
TC Output 1	DACC	Trigger for conversion. Selection in DAC module
TC Output 2	DACC	Trigger for conversion. Selection in DAC module
PWM Event Line 0	DACC	Trigger for conversion. Selection in DAC module
PWM Event Line 1	DACC	Trigger for conversion. Selection in DAC module

## 10. System Controller

The System Controller is a set of peripherals, which allow handling of key elements of the system, such as power, resets, clocks, time, interrupts, watchdog, etc.

See the system controller block diagram in [Figure 10-1 on page 31](#).

**Figure 10-1. System Controller Block Diagram**



## 10.1 System Controller and Peripherals Mapping

Please refer to [Section 7-1 “SAM4E Product Mapping” on page 23](#).

## 10.2 Power-on-Reset, Brownout and Supply Monitor

The SAM4E device embeds three features to monitor, warn and/or reset the chip:

- Power-on-Reset on VDDIO
- Brownout Detector on VDDCORE
- Supply Monitor on VDDIO

### 10.2.1 Power-on-Reset

The Power-on-Reset monitors VDDIO. It is always activated and monitors voltage at start up but also during power down. If VDDIO goes below the threshold voltage, the entire chip is reset. For more information, refer to the “Electrical Characteristics” section of the datasheet.

### 10.2.2 Brownout Detector on VDDCORE

The Brownout Detector monitors VDDCORE. It is active by default. It can be deactivated by software through the Supply Controller (SUPC\_MR). It is especially recommended to disable it during low-power modes such as wait or sleep modes.

If VDDCORE goes below the threshold voltage, the reset of the core is asserted. For more information, refer to the “Supply Controller (SUPC)” and “Electrical Characteristics” sections of the datasheet.

### 10.2.3 Supply Monitor on VDDIO

The Supply Monitor monitors VDDIO. It is not active by default. It can be activated by software and is fully programmable with 16 steps for the threshold (between 1.6V to 3.4V). It is controlled by the Supply Controller (SUPC). A sample mode is possible. It allows to divide the supply monitor power consumption by a factor of up to 2048. For more information, refer to the SUPC and Electrical Characteristics sections of the datasheet.

## 10.3 Reset Controller

The Reset Controller is based on a Power-on-Reset cell, and a Supply Monitor on VDDCORE.

The Reset Controller is capable to return to the software the source of the last reset, either a general reset, a wake-up reset, a software reset, a user reset or a watchdog reset.

The Reset Controller controls the internal resets of the system and the NRST pin input/output. It is capable to shape a reset signal for the external devices, simplifying to a minimum connection of a push-button on the NRST pin to implement a manual reset.

The configuration of the Reset Controller is saved as supplied on VDDIO.

- Multiplexing of four peripheral functions per I/O Line
- For each I/O Line (whether assigned to a peripheral or used as general-purpose I/O)
  - Input change interrupt
  - Configurable peripheral event generator
  - Programmable glitch filter
  - Programmable debouncing filter
  - Multi-drive option enables driving in open drain
  - Programmable pull-up on each I/O line
  - Pin data status register, supplies visibility of the level on the pin at any time
  - Additional interrupt modes on a programmable event: rising edge, falling edge, low level or high level
  - Lock of the configuration by the connected peripheral
- Synchronous output, provides set and clear of several I/O lines in a single write
- Write Protect Registers

- Programmable Schmitt trigger inputs
- Parallel capture mode:
  - Can be used to interface a CMOS digital image sensor (f.ex.)
  - One clock, 8-bit parallel data and two data enable on I/O lines
  - Data can be sampled one time out of two (for chrominance sampling only)
  - Supports connection of one Peripheral DMA Controller channel (PDC) which offers buffer reception without processor intervention

# 11. Peripherals

## 11.1 Peripheral Identifiers

Table 11-1 defines the Peripheral Identifiers of the SAM4E device. A peripheral identifier is required for the control of the peripheral interrupt with the Nested Vectored Interrupt Controller and control of the peripheral clock with the Power Management Controller.

Table 11-1. Peripheral Identifiers

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
0	SUPC	X		Supply Controller
1	RSTC	X		Reset Controller
2	RTC	X		Real-time Clock
3	RTT	X		Real-time Timer
4	WDT	X		Watchdog/Dual Watchdog Timer
5	PMC	X		Power Management Controller
6	EFC	X		Enhanced Embedded Flash Controller
7	UART0	X	X	UART 0
8	SMC		X	Static Memory Controller
9	PIOA	X	X	Parallel I/O Controller A
10	PIOB	X	X	Parallel I/O Controller B
11	PIOC	X	X	Parallel I/O Controller C
12	PIOD	X	X	Parallel I/O Controller D
13	PIOE	X	X	Parallel I/O Controller E
14	USART0	X	X	USART 0
15	USART1	X	X	USART 1
16	HSMCI	X	X	Multimedia Card Interface
17	TWI0	X	X	Two-wire Interface 0
18	TWI1	X	X	Two-wire Interface 1
19	SPI	X	X	Serial Peripheral Interface
20	DMAC	X	X	DMAC
21	TC0	X	X	Timer/Counter 0
22	TC1	X	X	Timer/Counter 1
23	TC2	X	X	Timer/Counter 2
24	TC3	X	X	Timer/Counter 3
25	TC4	X	X	Timer/Counter 4
26	TC5	X	X	Timer/Counter 5
27	TC6	X	X	Timer/Counter 6
28	TC7	X	X	Timer/Counter 7
29	TC8	X	X	Timer/Counter 8

Table 11-1. Peripheral Identifiers (Continued)

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
30	<b>AFEC0</b>	X	X	Analog Front End 0
31	<b>AFEC1</b>	X	X	Analog Front End 1
32	<b>DACC</b>	X	X	Digital to Analog Converter
33	<b>ACC</b>	X	X	Analog Comparator
34	<b>ARM</b>	X		FPU signals: FPIXC, FPOFC, FPUFC, FPIOC, FPDZC, FPIDC, FPIXC
35	<b>UDP</b>	X	X	USB DEVICE
36	<b>PWM</b>	X	X	PWM
37	<b>CAN0</b>	X	X	CAN0
38	<b>CAN1</b>	X	X	CAN1
39	<b>AES</b>	X	X	AES
40				Reserved
41				Reserved
42				Reserved
43				Reserved
44	<b>EMAC</b>	X	X	EMAC
45	<b>UART1</b>	X	X	UART
46				Reserved

## 11.2 Peripheral Signal Multiplexing on I/O Lines

The SAM4E device features five PIO Controllers on 144-pin versions (PIOA, PIOB, PIOC, PIOD and PIOE) that multiplex the I/O lines of the peripheral set.

The SAM4E PIO Controllers control up to 32 lines. Each line can be assigned to one of three peripheral functions: A, B or C. The multiplexing tables in the following paragraphs define how the I/O lines of the peripherals A, B and C are multiplexed on the PIO Controllers. The column “Comments” has been inserted in this table for the user’s own comments; it may be used to track how pins are defined in an application.

Note that some peripheral functions which are output only, might be duplicated within the tables.

### 11.2.1 PIO Controller A Multiplexing

Table 11-2. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Comments
PA0	PWMH0	TIOA0	A17	WKUP0		(1)
PA1	PWMH1	TIOB0	A18	WKUP1		(1)
PA2	PWMH2		DATRG	WKUP2		(1)
PA3	TWD0	NPCS3				
PA4	TWCK0	TCLK0		WKUP3		
PA5		NPCS3	URXD1	WKUP4		
PA6		PCK0	UTXD1			
PA7		PWMH3			XIN32	(1)
PA8		AFE0_ADTRG		WKUP5	XOUT32	
PA9	URXD0	NPCS1	PWMFI0	WKUP6		(1)
PA10	UTXD0	NPCS2				
PA11	NPCS0	PWMH0		WKUP7		(1)
PA12	MISO	PWMH1				(1)
PA13	MOSI	PWMH2				(1)
PA14	SPCK	PWMH3		WKUP8		(1)
PA15		TIOA1	PWML3	WKUP14/PIODCEN1		(1)
PA16		TIOB1	PWML2	WKUP15/PIODCEN2		(1)
PA17		PCK1	PWMH3	AFE0_AD0		(1)
PA18		PCK2	A14	AFE0_AD1		(1)
PA19		PWML0	A15	AFE0_AD2/WKUP9		(1)
PA20		PWML1	A16	AFE0_AD3/WKUP10		(1)
PA21	RXD1	PCK1		AFE1_AD2		
PA22	TXD1	NPCS3	NCS2	AFE1_AD3		(1)
PA23	SCK1	PWMH0	A19	PIODCCLK		(1)
PA24	RTS1	PWMH1	A20	PIODC0		(1)
PA25	CTS1	PWMH2	A23	PIODC1		(1)
PA26	DCD1	TIOA2	MCDA2	PIODC2		
PA27	DTR1	TIOB2	MCDA3	PIODC3		
PA28	DSR1	TCLK1	MCCDA	PIODC4		
PA29	RI1	TCLK2	MCCK	PIODC5		
PA30	PWML2	NPCS2	MCDA0	WKUP11/ PIODC6		(1)
PA31	NPCS1	PCK2	MCDA1	PIODC7		

Note: 1. Used by peripheral: defined as Bidirectional IO.

## 11.2.2 PIO Controller B Multiplexing

Table 11-3. Multiplexing on PIO Controller B (PIOB)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Comments
PB0	PWMH0		RXD0	AFE0_AD4/ RTCOUT0		(1)
PB1	PWMH1		TXD0	AFE0_AD5/ RTCOUT1		(1)
PB2	CANTX0	NPCS2	CTS0	AFE1_AD0/ WKUP12		(1)
PB3	CANRX0	PCK2	RTS0	AFE1_AD1		(1)
PB4	TWD1	PWMH2			TDI	(1)
PB5	TWCK1	PWML0		WKUP13	TDO/ TRACESWO	(1)
PB6					TMS/SWDIO	
PB7					TCK/SWCLK	
PB8					XOUT	
PB9					XIN	
PB10					DDM	
PB11					DDP	
PB12	PWML1				ERASE	(1)
PB13	PWML2	PCK0	SCK0	DAC0		(1)
PB14	NPCS1	PWMH3		DAC1		(1)

Note: 1. Used by peripheral: defined as Bidirectional IO.

### 11.2.3 PIO Controller C Multiplexing

Table 11-4. Multiplexing on PIO Controller C (PIOC)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Comments
PC0	D0	PWML0		AFE0_AD14		144-pin version <sup>(1)</sup>
PC1	D1	PWML1		AFE1_AD4		144-pin version <sup>(1)</sup>
PC2	D2	PWML2		AFE1_AD5		144-pin version <sup>(1)</sup>
PC3	D3	PWML3		AFE1_AD6		144-pin version <sup>(1)</sup>
PC4	D4	NPCS1		AFE1_AD7		144-pin version <sup>(1)</sup>
PC5	D5	TIOA6				144-pin version <sup>(1)</sup>
PC6	D6	TIOB6				144-pin version <sup>(1)</sup>
PC7	D7	TCLK6				144-pin version <sup>(1)</sup>
PC8	NWE	TIOA7				144-pin version <sup>(1)</sup>
PC9	NANDOE	TIOB7				144-pin version <sup>(1)</sup>
PC10	NANDWE	TCLK7				144-pin version <sup>(1)</sup>
PC11	NRD	TIOA8				144-pin version <sup>(1)</sup>
PC12	NCS3	TIOB8	CANRX1	AFE0_AD8		144-pin version <sup>(1)</sup>
PC13	NWAIT	PWML0		AFE0_AD6		144-pin version <sup>(1)</sup>
PC14	NCS0	TCLK8				144-pin version <sup>(1)</sup>
PC15	NCS1	PWML1	CANTX1	AFE0_AD7		144-pin version <sup>(1)</sup>
PC16	A21/ NANDALE					144-pin version <sup>(1)</sup>
PC17	A22/ NANDCLE					144-pin version <sup>(1)</sup>
PC18	A0	PWMH0				144-pin version <sup>(1)</sup>
PC19	A1	PWMH1				144-pin version <sup>(1)</sup>
PC20	A2	PWMH2				144-pin version <sup>(1)</sup>
PC21	A3	PWMH3				144-pin version <sup>(1)</sup>
PC22	A4	PWML3				144-pin version <sup>(1)</sup>
PC23	A5	TIOA3				144-pin version <sup>(1)</sup>
PC24	A6	TIOB3				144-pin version <sup>(1)</sup>
PC25	A7	TCLK3				144-pin version <sup>(1)</sup>
PC26	A8	TIOA4		AFE0_AD12		144-pin version <sup>(1)</sup>
PC27	A9	TIOB4		AFE0_AD13		144-pin version <sup>(1)</sup>
PC28	A10	TCLK4				144-pin version <sup>(1)</sup>
PC29	A11	TIOA5		AFE0_AD9		144-pin version <sup>(1)</sup>
PC30	A12	TIOB5		AFE0_AD10		144-pin version <sup>(1)</sup>
PC31	A13	TCLK5		AFE0_AD11		144-pin version <sup>(1)</sup>

Note: 1. Used by peripheral: defined as Bidirectional IO.

#### 11.2.4 PIO Controller D Multiplexing

Table 11-5. Multiplexing on PIO Controller D (PIOD)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Comments
PD0	GTXCK/GREF CK					
PD1	GTXEN					
PD2	GTX0					
PD3	GTX1					
PD4	GCRSDV/GR XDV					
PD5	GRX0					
PD6	GRX1					
PD7	GRXER					
PD8	GMDC					
PD9	GMDIO					
PD10	GCRS					
PD11	GRX2					
PD12	GRX3					
PD13	GCOL					
PD14	GRXCK					
PD15	GTX2					
PD16	GTX3					
PD17	GTXER					
PD18	NCS1					
PD19	NCS3					
PD20	PWMH0					
PD21	PWMH1					
PD22	PWMH2					
PD23	PWMH3					
PD24	PWML0					
PD25	PWML1					
PD26	PWML2					
PD27	PWML3					
PD28						
PD29						
PD30						
PD31						

### 11.2.5 PIO Controller E Multiplexing

Table 11-6. Multiplexing on PIO Controller E (PIOE)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Comments
PE0						144-pin version
PE1						144-pin version
PE2						144-pin version
PE3						144-pin version
PE4						144-pin version
PE5						144-pin version



## 12. ARM Cortex-M4

### 12.1 Description

The Cortex-M4 processor is a high performance 32-bit processor designed for the microcontroller market. It offers significant benefits to developers, including outstanding processing performance combined with fast interrupt handling, enhanced system debug with extensive breakpoint and trace capabilities, efficient processor core, system and memories, ultra-low power consumption with integrated sleep modes, and platform security robustness, with integrated memory protection unit (MPU).

The Cortex-M4 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including IEEE754-compliant single-precision floating-point computation, a range of single-cycle and SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M4 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M4 processor implements a version of the Thumb® instruction set based on Thumb-2 technology, ensuring high code density and reduced program memory requirements. The Cortex-M4 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M4 processor closely integrates a configurable NVIC, to deliver industry-leading interrupt performance. The NVIC includes a non-maskable interrupt (NMI), and provides up to 256 interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers, and the ability to suspend load-multiple and store-multiple operations. Interrupt handlers do not require wrapping in assembler code, removing any code overhead from the ISRs. A tail-chain optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with the sleep modes, that include a deep sleep function that enables the entire device to be rapidly powered down while still retaining program state.

#### 12.1.1 System Level Interface

The Cortex-M4 processor provides multiple interfaces using AMBA® technology to provide high speed, low latency memory accesses. It supports unaligned data accesses and implements atomic bit manipulation that enables faster peripheral controls, system spinlocks and thread-safe Boolean data handling.

The Cortex-M4 processor has a Memory Protection Unit (MPU) that provides fine grain memory control, enabling applications to utilize multiple privilege levels, separating and protecting code, data and stack on a task-by-task basis. Such requirements are becoming critical in many embedded applications such as automotive.

#### 12.1.2 Integrated Configurable Debug

The Cortex-M4 processor implements a complete hardware debug solution. This provides high system visibility of the processor and memory through either a traditional JTAG port or a 2-pin Serial Wire Debug (SWD) port that is ideal for microcontrollers and other small package devices.

For system trace the processor integrates an Instrumentation Trace Macrocell (ITM) alongside data watchpoints and a profiling unit. To enable simple and cost-effective profiling of the system events these generate, a Serial Wire Viewer (SWV) can export a stream of software-generated messages, data trace, and profiling information through a single pin.

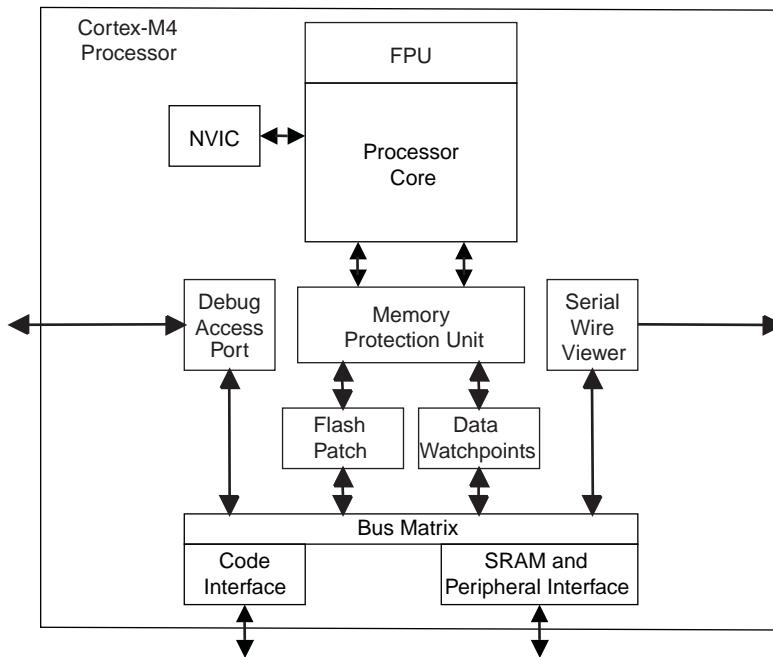
The Flash Patch and Breakpoint Unit (FPB) provides up to 8 hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to 8 words in the program code in the CODE memory region. This enables applications stored on a non-erasable, ROM-based microcontroller to be patched if a small programmable memory, for example flash, is available in the device. During initialization, the application in ROM detects, from the programmable memory, whether a patch is required. If a patch is required, the application programs the FPB to remap a number of addresses. When those addresses are accessed, the accesses are redirected to a remap table specified in the FPB configuration, which means the program in the non-modifiable ROM can be patched.

## 12.2 Embedded Characteristics

- Tight integration of system peripherals reduces area and development costs
- Thumb instruction set combines high code density with 32-bit performance
- IEEE754-compliant single-precision FPU
- Code-patch ability for ROM system updates
- Power control optimization of system components
- Integrated sleep modes for low power consumption
- Fast code execution permits slower processor clock or increases sleep mode time
- Hardware division and fast digital-signal-processing oriented multiply accumulate
- Saturating arithmetic for signal processing
- Deterministic, high-performance interrupt handling for time-critical applications
- Memory Protection Unit (MPU) for safety-critical applications
- Extensive debug and trace capabilities:
  - Serial Wire Debug and Serial Wire Trace reduce the number of pins required for debugging, tracing, and code profiling.

## 12.3 Block Diagram

Figure 12-1. Typical Cortex-M4 Implementation



## 12.4 Cortex-M4 Models

### 12.4.1 Programmers Model

This section describes the Cortex-M4 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and privilege levels for software execution and stacks.

#### 12.4.1.1 Processor Modes and Privilege Levels for Software Execution

The processor *modes* are:

- Thread mode  
Used to execute application software. The processor enters the Thread mode when it comes out of reset.
- Handler mode  
Used to handle exceptions. The processor returns to the Thread mode when it has finished exception processing.

The *privilege levels* for software execution are:

- Unprivileged

The software:

- Has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
- Cannot access the System Timer, NVIC, or System Control Block
- Might have a restricted access to memory or peripherals.

*Unprivileged software* executes at the unprivileged level.

- Privileged

The software can use all the instructions and has access to all resources. *Privileged software* executes at the privileged level.

In Thread mode, the CONTROL register controls whether the software execution is privileged or unprivileged, see “[CONTROL Register](#)”. In Handler mode, software execution is always privileged.

Only privileged software can write to the CONTROL register to change the privilege level for software execution in Thread mode. Unprivileged software can use the SVC instruction to make a *supervisor call* to transfer control to privileged software.

#### 12.4.1.2 Stacks

The processor uses a full descending stack. This means the stack pointer holds the address of the last stacked item in memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements two stacks, the *main stack* and the *process stack*, with a pointer for each held in independent registers, see “[Stack Pointer](#)”.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see “[CONTROL Register](#)”.

In Handler mode, the processor always uses the main stack.

The options for processor operations are:

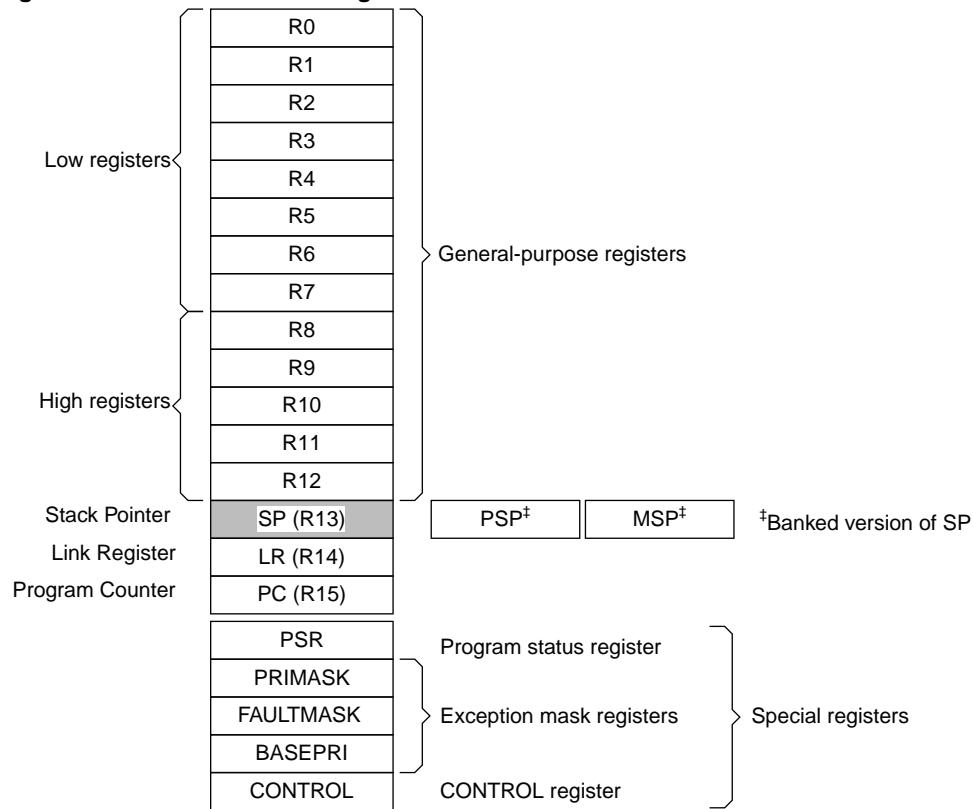
**Table 12-1. Summary of processor mode, execution privilege level, and stack use options**

Processor Mode	Used to Execute	Privilege Level for Software Execution	Stack Used
Thread	Applications	Privileged or unprivileged <sup>(1)</sup>	Main stack or process stack <sup>(1)</sup>
Handler	Exception handlers	Always privileged	Main stack

Note: 1. See “[CONTROL Register](#)”.

#### 12.4.1.3 Core Registers

Figure 12-2. Processor Core Registers



**Table 12-2. Core Processor Registers**

Register	Name	Access <sup>(1)</sup>	Required Privilege <sup>(2)</sup>	Reset
General-purpose registers	R0-R12	Read-write	Either	Unknown
Stack Pointer	MSP	Read-write	Privileged	See description
Stack Pointer	PSP	Read-write	Either	Unknown
Link Register	LR	Read-write	Either	0xFFFFFFFF
Program Counter	PC	Read-write	Either	See description
Program Status Register	PSR	Read-write	Privileged	0x01000000
Application Program Status Register	APSR	Read-write	Either	0x00000000
Interrupt Program Status Register	IPSR	Read-only	Privileged	0x00000000
Execution Program Status Register	EPSR	Read-only	Privileged	0x01000000
Priority Mask Register	PRIMASK	Read-write	Privileged	0x00000000
Fault Mask Register	FAULTMASK	Read-write	Privileged	0x00000000
Base Priority Mask Register	BASEPRI	Read-write	Privileged	0x00000000
CONTROL register	CONTROL	Read-write	Privileged	0x00000000

Notes:

1. Describes access type during program execution in thread mode and Handler mode. Debug access can differ.
2. An entry of Either means privileged and unprivileged software can access the register.

#### 12.4.1.4 General-purpose Registers

R0-R12 are 32-bit general-purpose registers for data operations.

#### 12.4.1.5 Stack Pointer

The *Stack Pointer* (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = *Main Stack Pointer* (MSP). This is the reset value.
- 1 = *Process Stack Pointer* (PSP).

On reset, the processor loads the MSP with the value from address 0x00000000.

#### 12.4.1.6 Link Register

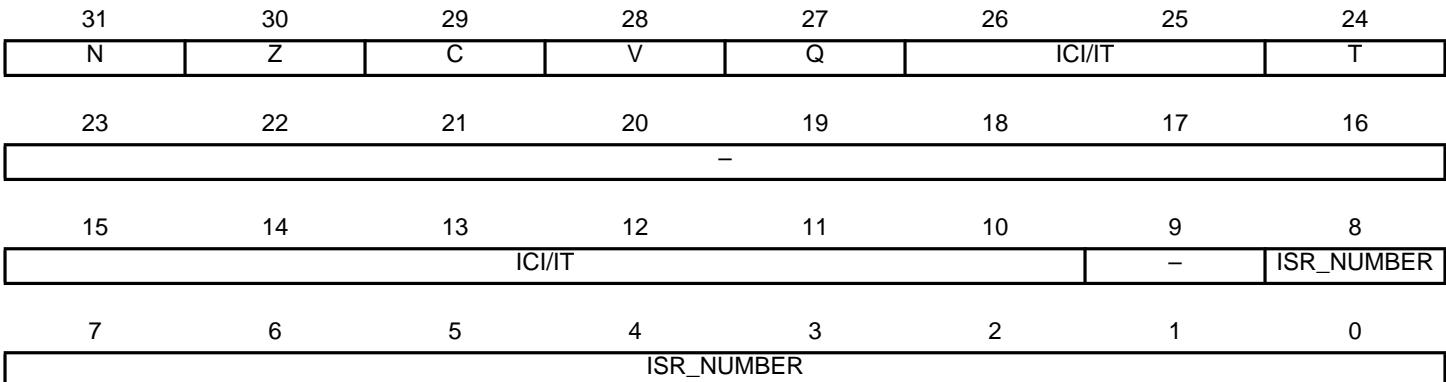
The *Link Register* (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the processor loads the LR value 0xFFFFFFFF.

#### 12.4.1.7 Program Counter

The *Program Counter* (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.

#### 12.4.1.8 Program Status Register

**Name:** PSR  
**Access:** Read-write  
**Reset:** 0x0000000000



The *Program Status Register* (PSR) combines:

- *Application Program Status Register* (APSR)
- *Interrupt Program Status Register* (IPSR)
- *Execution Program Status Register* (EPSR).

These registers are mutually exclusive bitfields in the 32-bit PSR.

The PSR register accesses these registers individually or as a combination of any two or all three registers, using the register name as an argument to the MRS or MRS instructions. For example:

- Read of all the registers using PSR with the MRS instruction
- Write to the APSR N, Z, C, V and Q bits using APSR\_nzcvq with the MSR instruction.

The PSR combinations and attributes are:

Name	Access	Combination
PSR	Read-write <sup>(1)(2)</sup>	APSR, EPSR, and IPSR
IEPSR	Read-only	EPSR and IPSR
IAPSR	Read-write <sup>(1)</sup>	APSR and IPSR
EAPSR	Read-write <sup>(2)</sup>	APSR and EPSR

Notes: 1. The processor ignores writes to the IPSR bits.  
       2. Reads of the EPSR bits return zero, and the processor ignores writes to these bits

See the instruction descriptions “[MRS](#)” and “[MSR](#)” for more information about how to access the program status registers.

#### 12.4.1.9 Application Program Status Register

**Name:** APSR

**Access:** Read-write

**Reset:** 0x0000000000

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

The APSR contains the current state of the condition flags from previous instruction executions.

- **N: Negative Flag**

0: Operation result was positive, zero, greater than, or equal

1: Operation result was negative or less than.

- **Z: Zero Flag**

0: Operation result was not zero

1: Operation result was zero.

- **C: Carry or Borrow Flag**

Carry or borrow flag:

0: Add operation did not result in a carry bit or subtract operation resulted in a borrow bit

1: Add operation resulted in a carry bit or subtract operation did not result in a borrow bit.

- **V: Overflow Flag**

0: Operation did not result in an overflow

1: Operation resulted in an overflow.

- **Q: DSP Overflow and Saturation Flag**

Sticky saturation flag:

0: Indicates that saturation has not occurred since reset or since the bit was last cleared to zero

1: Indicates when an SSAT or USAT instruction results in saturation.

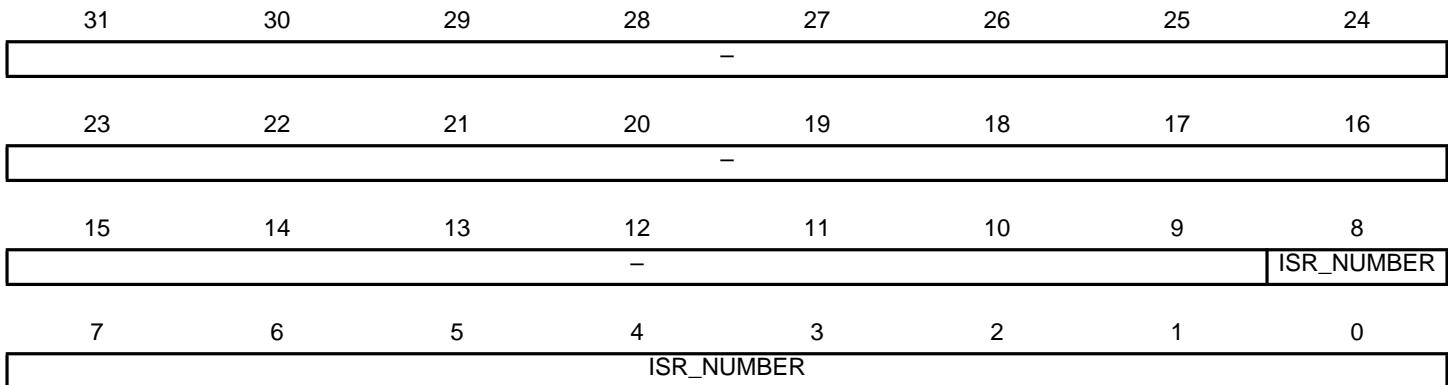
This bit is cleared to zero by software using an MRS instruction.

- **GE[19:16]: Greater Than or Equal Flags**

See “[SEL](#)” for more information.

#### 12.4.1.10 Interrupt Program Status Register

**Name:** IPSR  
**Access:** Read-write  
**Reset:** 0x0000000000



The IPSR contains the exception type number of the current *Interrupt Service Routine* (ISR).

- **ISR\_NUMBER: Number of the Current Exception**

0 = Thread mode

1 = Reserved

2 = NMI

3 = Hard fault

4 = Memory management fault

5 = Bus fault

6 = Usage fault

7-10 = Reserved

11 = SVCall

12 = Reserved for Debug

13 = Reserved

14 = PendSV

15 = SysTick

16 = IRQ0

50 = IRQ47

See "[Exception Types](#)" for more information.

#### 12.4.1.11 Execution Program Status Register

**Name:** EPSR  
**Access:** Read-write  
**Reset:** 0x0000000000

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

The EPSR contains the Thumb state bit, and the execution state bits for either the *If-Then* (IT) instruction, or the *Interruptible-Continuable Instruction* (ICI) field for an interrupted load multiple or store multiple instruction.

Attempts to read the EPSR directly through application software using the MSR instruction always return zero. Attempts to write the EPSR using the MSR instruction in the application software are ignored. Fault handlers can examine the EPSR value in the stacked PSR to indicate the operation that is at fault. See “[Exception Entry and Return](#)”

- **ICI: Interruptible-continuable Instruction**

When an interrupt occurs during the execution of an LDM, STM, PUSH, POP, VLDM, VSTM, VPUSH, or VPOP instruction, the processor:

- Stops the load multiple or store multiple instruction operation temporarily
- Stores the next register operand in the multiple operation to EPSR bits[15:12].

After servicing the interrupt, the processor:

- Returns to the register pointed to by bits[15:12]
- Resumes the execution of the multiple load or store instruction.

When the EPSR holds the ICI execution state, bits[26:25,11:10] are zero.

- **IT: If-Then Instruction**

Indicates the execution state bits of the IT instruction.

The If-Then block contains up to four instructions following an IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See “[IT](#)” for more information.

- **T: Thumb State**

The Cortex-M4 processor only supports the execution of instructions in Thumb state. The following can clear the T bit to 0:

- Instructions BLX, BX and POP{PC}
- Restoration from the stacked xPSR value on an exception return
- Bit[0] of the vector value on an exception entry or reset.

Attempting to execute instructions when the T bit is 0 results in a fault or lockup. See “[Lockup](#)” for more information.

#### 12.4.1.12 Exception Mask Registers

The exception mask registers disable the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks.

To access the exception mask registers use the MSR and MRS instructions, or the CPS instruction to change the value of PRIMASK or FAULTMASK. See “[MRS](#)”, “[MSR](#)”, and “[CPS](#)” for more information.

#### 12.4.1.13 Priority Mask Register

**Name:** PRIMASK

**Access:** Read-write

**Reset:** 0x000000000

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
			—				PRIMASK

The PRIMASK register prevents the activation of all exceptions with a configurable priority.

- **PRIMASK**

0: No effect

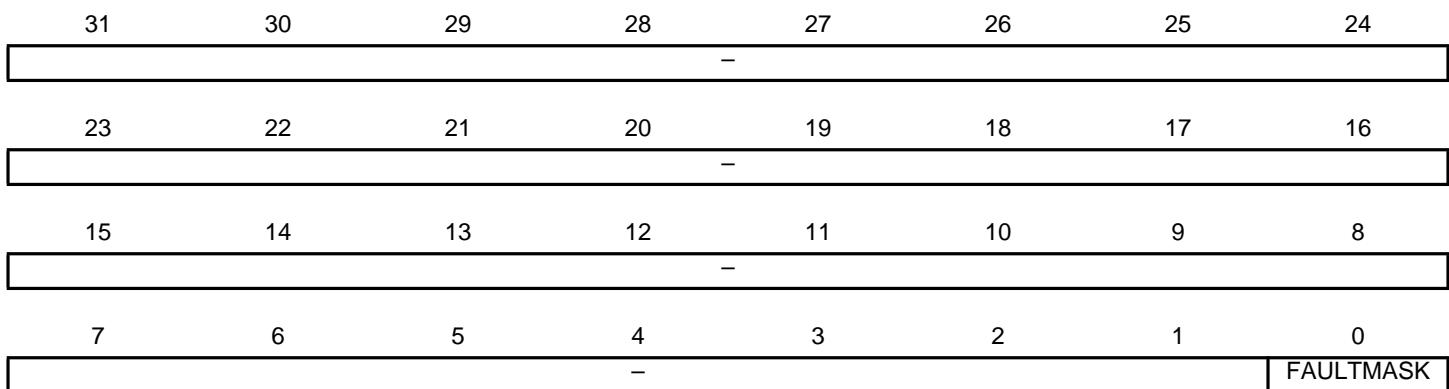
1: Prevents the activation of all exceptions with a configurable priority.

#### 12.4.1.14 Fault Mask Register

**Name:** FAULTMASK

**Access:** Read-write

**Reset:** 0x000000000



The FAULTMASK register prevents the activation of all exceptions except for Non-Maskable Interrupt (NMI).

- **FAULTMASK**

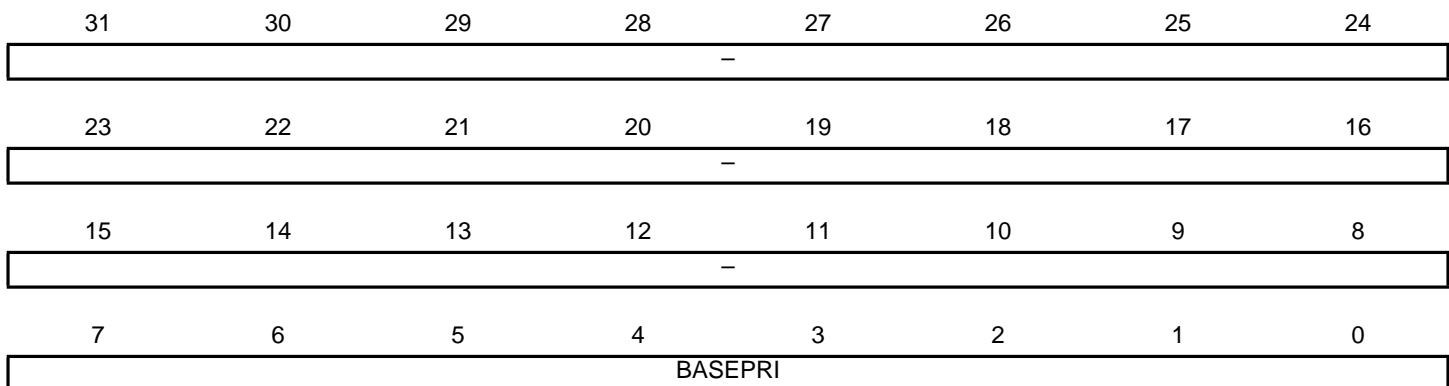
0: No effect.

1: Prevents the activation of all exceptions except for NMI.

The processor clears the FAULTMASK bit to 0 on exit from any exception handler except the NMI handler.

#### 12.4.1.15 Base Priority Mask Register

**Name:** BASEPRI  
**Access:** Read-write  
**Reset:** 0x00000000



The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with same or lower priority level as the BASEPRI value.

- **BASEPRI**

Priority mask bits:

0x0000 = No effect.

Nonzero = Defines the base priority for exception processing.

The processor does not process any exception with a priority value greater than or equal to BASEPRI.

This field is similar to the priority fields in the interrupt priority registers. The processor implements only bits[7:4] of this field, bits[3:0] read as zero and ignore writes. See "[Interrupt Priority Registers](#)" for more information. Remember that higher priority field values correspond to lower exception priorities.

#### 12.4.1.16 CONTROL Register

**Name:** CONTROL

**Access:** Read-write

**Reset:** 0x000000000

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
		—			FPCA	SPSEL	nPRIV

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode and indicates whether the FPU state is active.

- **FPCA: Floating-point Context Active**

Indicates whether the floating-point context is currently active:

0: No floating-point context active.

1: Floating-point context active.

The Cortex-M4 uses this bit to determine whether to preserve the floating-point state when processing an exception.

- **SPSEL: Active Stack Pointer**

Defines the current stack:

0: MSP is the current stack pointer.

1: PSP is the current stack pointer.

In Handler mode, this bit reads as zero and ignores writes. The Cortex-M4 updates this bit automatically on exception return.

- **nPRIV: Thread Mode Privilege Level**

Defines the Thread mode privilege level:

0: Privileged.

1: Unprivileged.

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the CONTROL register when in Handler mode. The exception entry and return mechanisms update the CONTROL register based on the EXC\_RETURN value.

In an OS environment, ARM recommends that threads running in Thread mode use the process stack, and the kernel and exception handlers use the main stack.

By default, the Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, either:

- Use the MSR instruction to set the Active stack pointer bit to 1, see “[MSR](#)”, or
- Perform an exception return to Thread mode with the appropriate EXC\_RETURN value, see [Table 12-10](#).

Note: When changing the stack pointer, the software must use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB execute using the new stack pointer. See “[ISB](#)”.

#### 12.4.1.17 Exceptions and Interrupts

The Cortex-M4 processor supports interrupts and system exceptions. The processor and the *Nested Vectored Interrupt Controller* (NVIC) prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses the Handler mode to handle all exceptions except for reset. See “[Exception Entry](#)” and “[Exception Return](#)” for more information.

The NVIC registers control interrupt handling. See “[Nested Vectored Interrupt Controller \(NVIC\)](#)” for more information.

#### 12.4.1.18 Data Types

The processor supports the following data types:

- 32-bit words
- 16-bit halfwords
- 8-bit bytes
- The processor manages all data memory accesses as little-endian. Instruction memory and *Private Peripheral Bus* (PPB) accesses are always little-endian. See “[Memory Regions, Types and Attributes](#)” for more information.

#### 12.4.1.19 Cortex Microcontroller Software Interface Standard (CMSIS)

For a Cortex-M4 microcontroller system, the *Cortex Microcontroller Software Interface Standard* (CMSIS) defines:

- A common way to:
  - Access peripheral registers
  - Define exception vectors
- The names of:
  - The registers of the core peripherals
  - The core exception vectors
- A device-independent interface for RTOS kernels, including a debug channel.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M4 processor.

The CMSIS simplifies the software development by enabling the reuse of template code and the combination of CMSIS-compliant software components from various middleware vendors. Software vendors can expand the CMSIS to include their peripheral definitions and access functions for those peripherals.

This document includes the register names defined by the CMSIS, and gives short descriptions of the CMSIS functions that address the processor core and the core peripherals.

Note: This document uses the register short names defined by the CMSIS. In a few cases, these differ from the architectural short names that might be used in other documents.

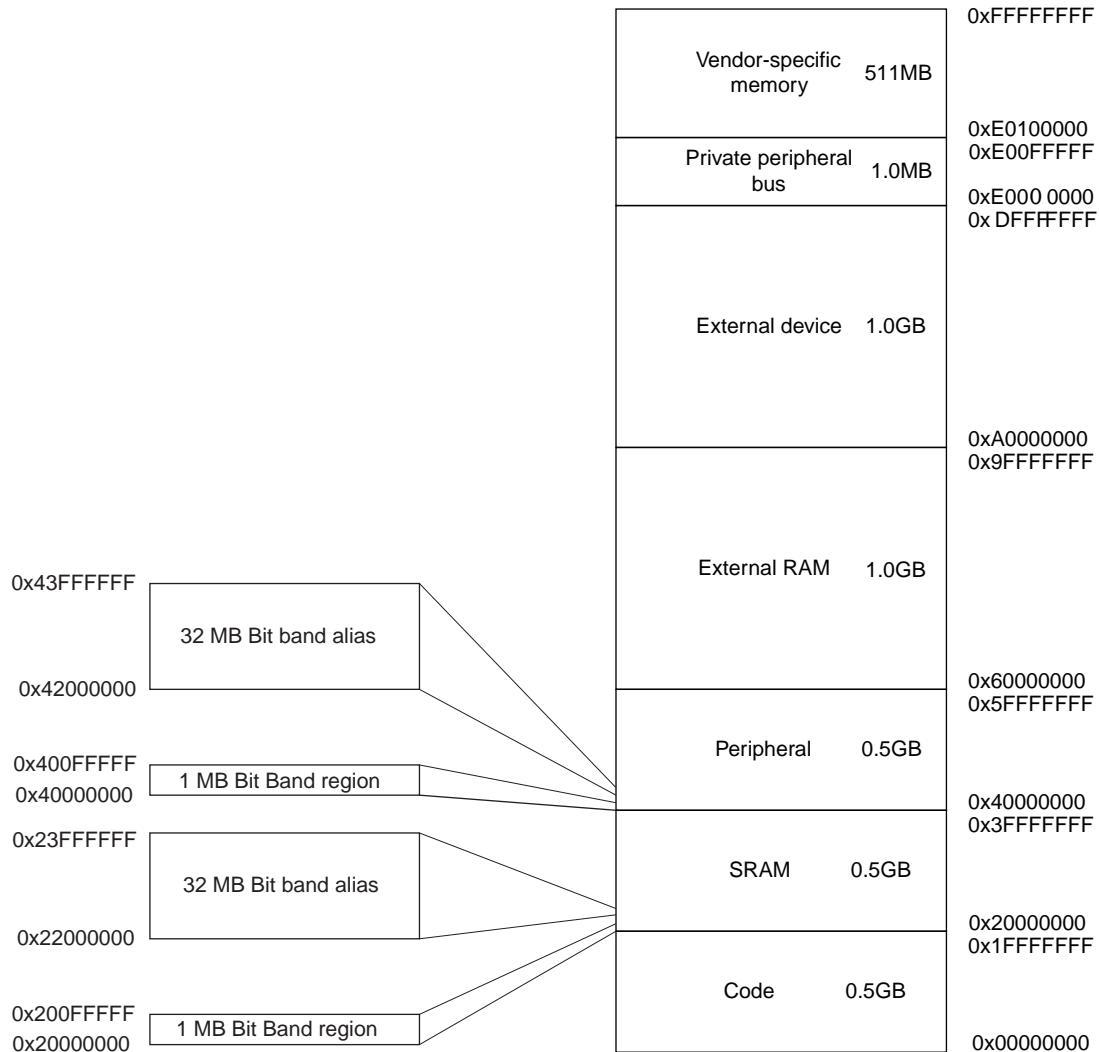
The following sections give more information about the CMSIS:

- [Section 12.5.3 "Power Management Programming Hints"](#)
- [Section 12.6.2 "CMSIS Functions"](#)
- [Section 12.8.2.1 "NVIC Programming Hints"](#) .

## 12.4.2 Memory Model

This section describes the processor memory map, the behavior of memory accesses, and the bit-banding features. The processor has a fixed memory map that provides up to 4GB of addressable memory.

**Figure 12-3. Memory Map**



The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data, see “[Bit-banding](#)”.

The processor reserves regions of the *Private peripheral bus* (PPB) address range for core peripheral registers.

This memory mapping is generic to ARM Cortex-M4 products. To get the specific memory mapping of this product, refer to the Memories section of the datasheet.

#### 12.4.2.1 Memory Regions, Types and Attributes

The memory map and the programming of the MPU split the memory map into regions. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

##### Memory Types

- **Normal**  
The processor can re-order transactions for efficiency, or perform speculative reads.
- **Device**  
The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.
- **Strongly-ordered**  
The processor preserves transaction order relative to all other transactions.

The different ordering requirements for Device and Strongly-ordered memory mean that the memory system can buffer a write to Device memory, but must not buffer a write to Strongly-ordered memory.

##### Additional Memory Attributes

- **Shareable**  
For a shareable memory region, the memory system provides data synchronization between bus masters in a system with multiple bus masters, for example, a processor with a DMA controller.  
Strongly-ordered memory is always shareable.  
If multiple bus masters can access a non-shareable memory region, the software must ensure data coherency between the bus masters.
- **Execute Never (XN)**  
Means the processor prevents instruction accesses. A fault exception is generated only on execution of an instruction executed from an XN region.

#### 12.4.2.2 Memory System Ordering of Memory Accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing this does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, the software must insert a memory barrier instruction between the memory access instructions, see "[Software Ordering of Memory Accesses](#)".

However, the memory system does guarantee some ordering of accesses to Device and Strongly-ordered memory. For two memory access instructions A1 and A2, if A1 occurs before A2 in program order, the ordering of the memory accesses is described below.

Table 12-3. Ordering of the Memory Accesses Caused by Two Instructions

A1	A2	Device Access		Strongly-ordered Access
		Normal Access	Non-shareable	
Normal Access	–	–	–	–
Device access, non-shareable	–	<	–	<
Device access, shareable	–	–	<	<
Strongly-ordered access	–	<	<	<

Where:

- Means that the memory system does not guarantee the ordering of the accesses.
- < Means that accesses are observed in program order, that is, A1 is always observed before A2.

#### 12.4.2.3 Behavior of Memory Accesses

The behavior of accesses to each region in the memory map is:

**Table 12-4. Memory Access Behavior**

Address Range	Memory Region	Memory Type	XN	Description
0x00000000 - 0x1FFFFFFF	Code	Normal <sup>(1)</sup>	-	Executable region for program code. Data can also be put here.
0x20000000 - 0x3FFFFFFF	SRAM	Normal <sup>(1)</sup>	-	Executable region for data. Code can also be put here. This region includes bit band and bit band alias areas, see <a href="#">Table 12-6</a> .
0x40000000 - 0x5FFFFFFF	Peripheral	Device <sup>(1)</sup>	XN	This region includes bit band and bit band alias areas, see <a href="#">Table 12-6</a> .
0x60000000 - 0x9FFFFFFF	External RAM	Normal <sup>(1)</sup>	-	Executable region for data.
0xA0000000 - 0xDFFFFFFF	External device	Device <sup>(1)</sup>	XN	External Device memory
0xE0000000 - 0xE00FFFFF	Private Peripheral Bus	Strongly-ordered <sup>(1)</sup>	XN	This region includes the NVIC, System timer, and system control block.
0xE0100000 - 0xFFFFFFF	Reserved	Device <sup>(1)</sup>	XN	Reserved

Note: 1. See “[Memory Regions, Types and Attributes](#)” for more information.

The Code, SRAM, and external RAM regions can hold programs. However, ARM recommends that programs always use the Code region. This is because the processor has separate buses that enable instruction fetches and data accesses to occur simultaneously.

The MPU can override the default memory access behavior described in this section. For more information, see “[Memory Protection Unit \(MPU\)](#)”.

#### *Additional Memory Access Constraints For Caches and Shared Memory*

When a system includes caches or shared memory, some memory regions have additional access constraints, and some regions are subdivided, as [Table 12-5](#) shows:

**Table 12-5. Memory Region Shareability and Cache Policies**

Address Range	Memory Region	Memory Type	Shareability	Cache Policy
0x00000000-0x1FFFFFFF	Code	Normal <sup>(1)</sup>	-	WT <sup>(2)</sup>
0x20000000-0x3FFFFFFF	SRAM	Normal <sup>(1)</sup>	-	WBWA <sup>(2)</sup>
0x40000000-0x5FFFFFFF	Peripheral	Device <sup>(1)</sup>	-	-
0x60000000-0x7FFFFFFF	External RAM	Normal <sup>(1)</sup>	-	WBWA <sup>(2)</sup>
0x80000000-0x9FFFFFFF				WT <sup>(2)</sup>

**Table 12-5. Memory Region Shareability and Cache Policies (Continued)**

Address Range	Memory Region	Memory Type	Shareability	Cache Policy
0xA0000000-0xBFFFFFFF	External device	Device <sup>(1)</sup>	Shareable <sup>(1)</sup>	-
0xC0000000-0xDFFFFFFF			Non-shareable <sup>(1)</sup>	-
0xE0000000-0xE00FFFFF	Private Peripheral Bus	Strongly- ordered <sup>(1)</sup>	Shareable <sup>(1)</sup>	-
0xE0100000-0xFFFFFFF	Vendor-specific device	Device <sup>(1)</sup>	-	-

Notes: 1. See “[Memory Regions, Types and Attributes](#)” for more information.

2. WT = Write through, no write allocate. WBA = Write back, write allocate. See the “[Glossary](#)” for more information.

#### *Instruction Prefetch and Branch Prediction*

The Cortex-M4 processor:

- Prefetches instructions ahead of execution
- Speculatively prefetches from branch target addresses.

#### **12.4.2.4 Software Ordering of Memory Accesses**

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions. This is because:

- The processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
- The processor has multiple bus interfaces
- Memory or devices in the memory map have different wait states
- Some memory accesses are buffered or speculative.

“[Memory System Ordering of Memory Accesses](#)” describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, the software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

##### *DMB*

The *Data Memory Barrier* (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions. See “[DMB](#)” .

##### *DSB*

The *Data Synchronization Barrier* (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See “[DSB](#)” .

##### *ISB*

The *Instruction Synchronization Barrier* (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions. See “[ISB](#)” .

##### *MPU Programming*

Use a DSB followed by an ISB instruction or exception return to ensure that the new MPU configuration is used by subsequent instructions.

#### 12.4.2.5 Bit-banding

A bit-band region maps each word in a *bit-band alias* region to a single bit in the *bit-band region*. The bit-band regions occupy the lowest 1 MB of the SRAM and peripheral memory regions.

The memory map has two 32 MB alias regions that map to two 1 MB bit-band regions:

- Accesses to the 32 MB SRAM alias region map to the 1 MB SRAM bit-band region, as shown in [Table 12-6](#).
- Accesses to the 32 MB peripheral alias region map to the 1 MB peripheral bit-band region, as shown in [Table 12-7](#).

**Table 12-6. SRAM Memory Bit-banding Regions**

Address Range	Memory Region	Instruction and Data Accesses
0x20000000-0x200FFFFF	SRAM bit-band region	Direct accesses to this memory range behave as SRAM memory accesses, but this region is also bit-addressable through bit-band alias.
0x22000000-0x23FFFFFF	SRAM bit-band alias	Data accesses to this region are remapped to bit-band region. A write operation is performed as read-modify-write. Instruction accesses are not remapped.

**Table 12-7. Peripheral Memory Bit-banding Regions**

Address Range	Memory Region	Instruction and Data Accesses
0x40000000-0x400FFFFF	Peripheral bit-band alias	Direct accesses to this memory range behave as peripheral memory accesses, but this region is also bit-addressable through bit-band alias.
0x42000000-0x43FFFFFF	Peripheral bit-band region	Data accesses to this region are remapped to bit-band region. A write operation is performed as read-modify-write. Instruction accesses are not permitted.

Notes:

1. A word access to the SRAM or peripheral bit-band alias regions map to a single bit in the SRAM or peripheral bit-band region.
2. Bit-band accesses can use byte, halfword, or word transfers. The bit-band transfer size matches the transfer size of the instruction making the bit-band access.

The following formula shows how the alias region maps onto the bit-band region:

```
bit_word_offset = (byte_offset x 32) + (bit_number x 4)
bit_word_addr = bit_band_base + bit_word_offset
```

where:

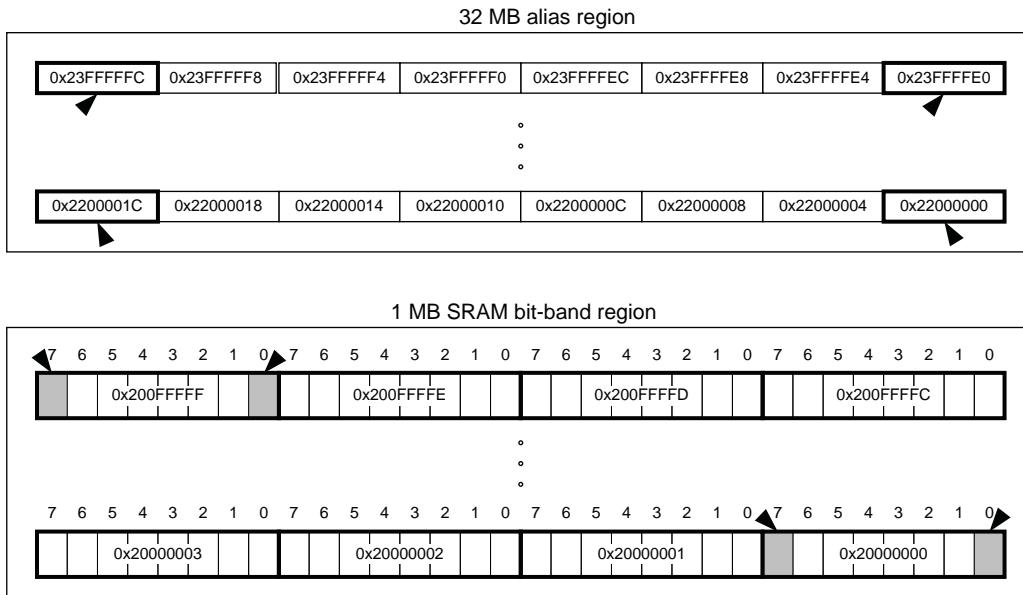
- `Bit_word_offset` is the position of the target bit in the bit-band memory region.
- `Bit_word_addr` is the address of the word in the alias memory region that maps to the targeted bit.
- `Bit_band_base` is the starting address of the alias region.
- `Byte_offset` is the number of the byte in the bit-band region that contains the targeted bit.
- `Bit_number` is the bit position, 0-7, of the targeted bit.

[Figure 12-4](#) shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 0x23FFFFFFE0 maps to bit[0] of the bit-band byte at 0x200FFFFF:  $0x23FFFFFFE0 = 0x22000000 + (0xFFFF * 32) + (0 * 4)$ .
- The alias word at 0x23FFFFFFFC maps to bit[7] of the bit-band byte at 0x200FFFFF:  $0x23FFFFFFFC = 0x22000000 + (0xFFFF * 32) + (7 * 4)$ .
- The alias word at 0x22000000 maps to bit[0] of the bit-band byte at 0x20000000:  $0x22000000 = 0x22000000 + (0 * 32) + (0 * 4)$ .

- The alias word at 0x2200001C maps to bit[7] of the bit-band byte at 0x20000000:  $0x2200001C = 0x20000000 + (0*32) + (7*4)$ .

**Figure 12-4. Bit-band Mapping**



#### Directly Accessing an Alias Region

Writing to a word in the alias region updates a single bit in the bit-band region.

Bit[0] of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit[0] set to 1 writes a 1 to the bit-band bit, and writing a value with bit[0] set to 0 writes a 0 to the bit-band bit.

Bits[31:1] of the alias word have no effect on the bit-band bit. Writing 0x01 has the same effect as writing 0xFF. Writing 0x00 has the same effect as writing 0x0E.

Reading a word in the alias region:

- 0x00000000 indicates that the targeted bit in the bit-band region is set to 0
- 0x00000001 indicates that the targeted bit in the bit-band region is set to 1

#### Directly Accessing a Bit-band Region

"[Behavior of Memory Accesses](#)" describes the behavior of direct byte, halfword, or word accesses to the bit-band regions.

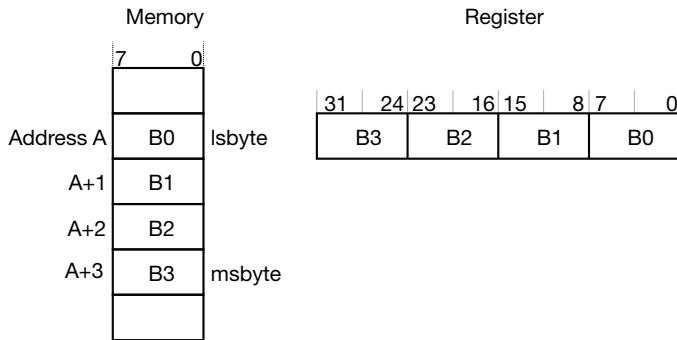
#### 12.4.2.6 Memory Endianness

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. “[Little-endian Format](#)” describes how words of data are stored in memory.

##### Little-endian Format

In little-endian format, the processor stores the least significant byte of a word at the lowest-numbered byte, and the most significant byte at the highest-numbered byte. For example:

**Figure 12-5. Little-endian Format**



#### 12.4.2.7 Synchronization Primitives

The Cortex-M4 instruction set includes pairs of *synchronization primitives*. These provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. The software can use them to perform a guaranteed read-modify-write memory update sequence, or for a semaphore mechanism.

A pair of synchronization primitives comprises:

**A Load-exclusive Instruction**, used to read the value of a memory location, requesting exclusive access to that location.

**A Store-Exclusive instruction**, used to attempt to write to the same memory location, returning a status bit to a register. If this bit is:

- 0: It indicates that the thread or process gained exclusive access to the memory, and the write succeeds,
- 1: It indicates that the thread or process did not gain exclusive access to the memory, and no write is performed.

The pairs of Load-Exclusive and Store-Exclusive instructions are:

- The word instructions LDREX and STREX
- The halfword instructions LDREXH and STREXH
- The byte instructions LDREXB and STREXB.

The software must use a Load-Exclusive instruction with the corresponding Store-Exclusive instruction.

To perform an exclusive read-modify-write of a memory location, the software must:

1. Use a Load-Exclusive instruction to read the value of the location.
2. Update the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location
4. Test the returned status bit. If this bit is:
  - 0: The read-modify-write completed successfully.
  - 1: No write was performed. This indicates that the value returned at step 1 might be out of date. The software must retry the read-modify-write sequence.

The software can use the synchronization primitives to implement a semaphore as follows:

1. Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.
2. If the semaphore is free, use a Store-Exclusive instruction to write the claim value to the semaphore address.
3. If the returned status bit from step 2 indicates that the Store-Exclusive instruction succeeded then the software has claimed the semaphore. However, if the Store-Exclusive instruction failed, another process might have claimed the semaphore after the software performed the first step.

The Cortex-M4 includes an exclusive access monitor, that tags the fact that the processor has executed a Load-Exclusive instruction. If the processor is part of a multiprocessor system, the system also globally tags the memory locations addressed by exclusive accesses by each processor.

The processor removes its exclusive access tag if:

- It executes a CLREX instruction
- It executes a Store-Exclusive instruction, regardless of whether the write succeeds.
- An exception occurs. This means that the processor can resolve semaphore conflicts between different threads.

In a multiprocessor implementation:

- Executing a CLREX instruction removes only the local exclusive access tag for the processor
- Executing a Store-Exclusive instruction, or an exception, removes the local exclusive access tags, and all global exclusive access tags for the processor.

For more information about the synchronization primitive instructions, see “[LDREX and STREX](#)” and “[CLREX](#)” .

#### 12.4.2.8 Programming Hints for the Synchronization Primitives

ISO/IEC C cannot directly generate the exclusive access instructions. CMSIS provides intrinsic functions for generation of these instructions:

**Table 12-8. CMSIS Functions for Exclusive Access Instructions**

Instruction	CMSIS Function
LDREX	uint32_t __LDREXW (uint32_t *addr)
LDREXH	uint16_t __LDREXH (uint16_t *addr)
LDREXB	uint8_t __LDREXB (uint8_t *addr)
STREX	uint32_t __STREXW (uint32_t value, uint32_t *addr)
STREXH	uint32_t __STREXH (uint16_t value, uint16_t *addr)
STREXB	uint32_t __STREXB (uint8_t value, uint8_t *addr)
CLREX	void __CLREX (void)

The actual exclusive access instruction generated depends on the data type of the pointer passed to the intrinsic function. For example, the following C code generates the required LDREXB operation:

```
_ldrex((volatile char *) 0xFF);
```

#### 12.4.3 Exception Model

This section describes the exception model.

##### 12.4.3.1 Exception States

Each exception is in one of the following states:

###### *Inactive*

The exception is not active and not pending.

###### *Pending*

The exception is waiting to be serviced by the processor.

An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.

#### *Active*

An exception is being serviced by the processor but has not completed.

An exception handler can interrupt the execution of another exception handler. In this case, both exceptions are in the active state.

#### *Active and Pending*

The exception is being serviced by the processor and there is a pending exception from the same source.

### 12.4.3.2 Exception Types

The exception types are:

#### *Reset*

Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.

#### *Non Maskable Interrupt (NMI)*

A non maskable interrupt (NMI) can be signalled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2.

NMIs cannot be:

- Masked or prevented from activation by any other exception.
- Preempted by any exception other than Reset.

#### *Hard Fault*

A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. Hard Faults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.

#### *Memory Management Fault (MemManage)*

A Memory Management Fault is an exception that occurs because of a memory protection related fault. The MPU or the fixed memory protection constraints determines this fault, for both instruction and data memory transactions. This fault is used to abort instruction accesses to *Execute Never (XN)* memory regions, even if the MPU is disabled.

#### *Bus Fault*

A Bus Fault is an exception that occurs because of a memory related fault for an instruction or data memory transaction. This might be from an error detected on a bus in the memory system.

#### *Usage Fault*

A Usage Fault is an exception that occurs because of a fault related to an instruction execution. This includes:

- An undefined instruction
- An illegal unaligned access
- An invalid state on instruction execution
- An error on exception return.

The following can cause a Usage Fault when the core is configured to report them:

- An unaligned address on word and halfword memory access
- A division by zero.

#### *SVCall*

A *supervisor call* (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.

## PendSV

PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.

## SysTick

A SysTick exception is an exception the system timer generates when it reaches zero. Software can also generate a SysTick exception. In an OS environment, the processor can use this exception as system tick.

## Interrupt (IRQ)

A interrupt, or IRQ, is an exception signalled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor.

**Table 12-9. Properties of the Different Exception Types**

Exception Number <sup>(1)</sup>	Irq Number <sup>(1)</sup>	Exception Type	Priority	Vector Address or Offset <sup>(2)</sup>	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	Hard fault	-1	0x0000000C	-
4	-12	Memory management fault	Configurable <sup>(3)</sup>	0x00000010	Synchronous
5	-11	Bus fault	Configurable <sup>(3)</sup>	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	Usage fault	Configurable <sup>(3)</sup>	0x00000018	Synchronous
7-10	-	-	-	Reserved	-
11	-5	SVCALL	Configurable <sup>(3)</sup>	0x0000002C	Synchronous
12-13	-	-	-	Reserved	-
14	-2	PendSV	Configurable <sup>(3)</sup>	0x00000038	Asynchronous
15	-1	SysTick	Configurable <sup>(3)</sup>	0x0000003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable <sup>(4)</sup>	0x00000040 and above <sup>(5)</sup>	Asynchronous

Notes: 1. To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see "[Interrupt Program Status Register](#)".

2. See "[Vector Table](#)" for more information

3. See "[System Handler Priority Registers](#)"

4. See "[Interrupt Priority Registers](#)"

5. Increasing in steps of 4.

For an asynchronous exception, other than reset, the processor can execute another instruction between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that [Table 12-9](#) shows as having configurable priority, see:

- "[System Handler Control and State Register](#)"
- "[Interrupt Clear-enable Registers](#)" .

For more information about hard faults, memory management faults, bus faults, and usage faults, see "[Fault Handling](#)" .

### 12.4.3.3 Exception Handlers

The processor handles exceptions using:

- Interrupt Service Routines (ISRs)  
Interrupts IRQ0 to IRQ47 are the exceptions handled by ISRs.
- Fault Handlers  
Hard fault, memory management fault, usage fault, bus fault are fault exceptions handled by the fault handlers.
- System Handlers  
NMI, PendSV, SVCALL SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

#### 12.4.3.4 Vector Table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. [Figure 12-6](#) shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code.

**Figure 12-6. Vector Table**

Exception number	IRQ number	Offset	Vector
255	239	0x03FC	IRQ239
.	.	.	.
.	.	.	.
18	2	0x004C	IRQ2
17	1	0x0048	IRQ1
16	0	0x0044	IRQ0
15	-1	0x0040	SysTick
14	-2	0x003C	PendSV
13		0x0038	Reserved
12			Reserved for Debug
11	-5	0x002C	SVCALL
10			Reserved
9			Reserved
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

On system reset, the vector table is fixed at address 0x00000000. Privileged software can write to the SCB\_VTOR register to relocate the vector table start address to a different memory location, in the range 0x00000080 to 0x3FFFFF80, see [“Vector Table Offset Register”](#).

#### 12.4.3.5 Exception Priorities

As [Table 12-9](#) shows, all exceptions have an associated priority, with:

- A lower priority value indicating a higher priority
- Configurable priorities for all exceptions except Reset, Hard fault and NMI.

If the software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see “[System Handler Priority Registers](#)” , and “[Interrupt Priority Registers](#)” .

Note: Configurable priority values are in the range 0-15. This means that the Reset, Hard fault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

#### 12.4.3.6 Interrupt Priority Grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This divides each interrupt priority register entry into two fields:

- An upper field that defines the *group priority*
- A lower field that defines a *subpriority* within the group.

Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler.

If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see “[Application Interrupt and Reset Control Register](#)” .

#### 12.4.3.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

##### *Preemption*

When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See “[Interrupt Priority Grouping](#)” for more information about preemption by an interrupt.

When one exception preempts another, the exceptions are called nested exceptions. See “[Exception Entry](#)” more information.

##### *Return*

This occurs when the exception handler is completed, and:

- There is no pending exception with sufficient priority to be serviced
- The completed exception handler was not handling a late-arriving exception.

The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See “[Exception Return](#)” for more information.

##### *Tail-chaining*

This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.

##### *Late-arriving*

This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved is the same for both exceptions. Therefore the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

#### *Exception Entry*

An Exception entry occurs when there is a pending exception with sufficient priority and either the processor is in Thread mode, or the new exception is of a higher priority than the exception being handled, in which case the new exception preempts the original exception.

When one exception preempts another, the exceptions are nested.

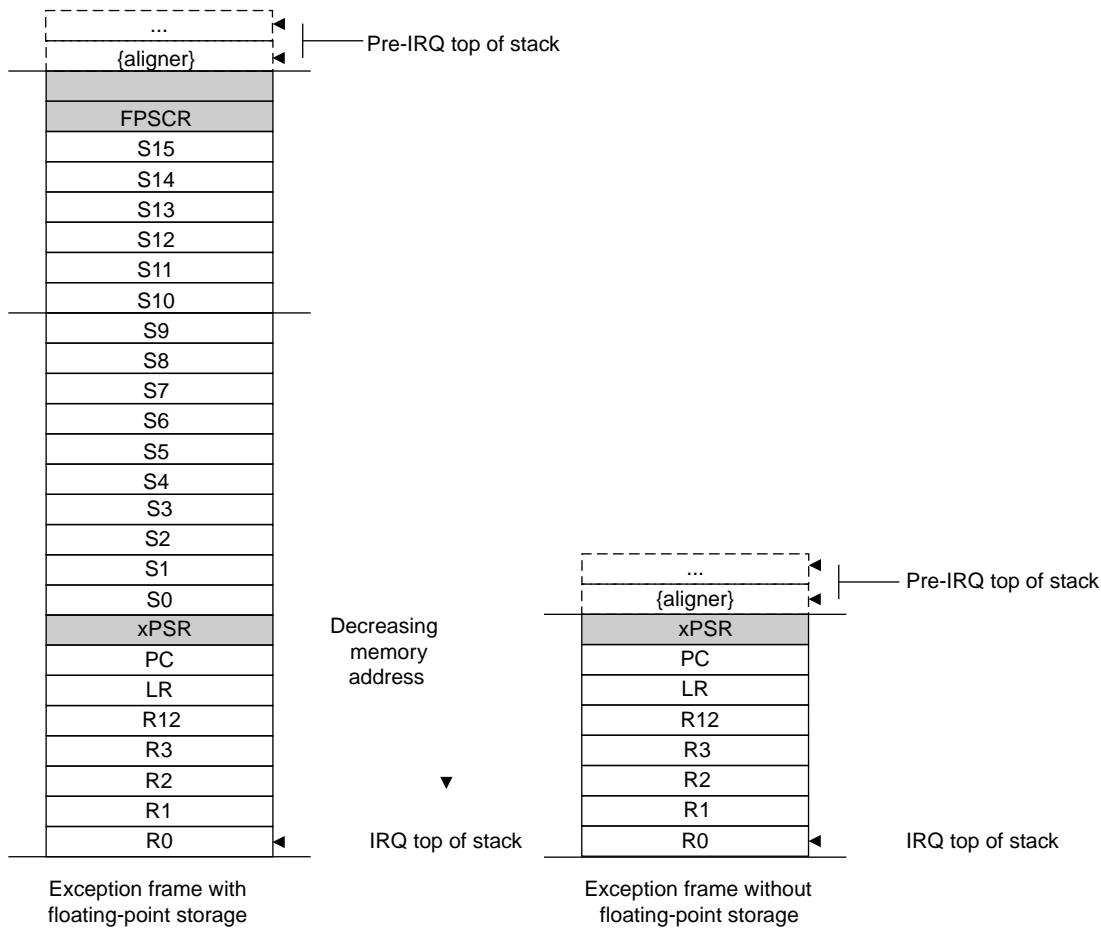
Sufficient priority means that the exception has more priority than any limits set by the mask registers, see “[Exception Mask Registers](#)” . An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred as *stacking* and the structure of eight data words is referred to as *stack frame*.

When using floating-point routines, the Cortex-M4 processor automatically stacks the architected floating-point state on exception entry. Figure 2-3 on page 2-27 shows the Cortex-M4 stack frame layout when floating-point state is preserved on the stack as the result of an interrupt or an exception.

Note: Where stack space for floating-point state is not allocated, the stack frame is the same as that of ARMv7-M implementations without an FPU. Figure 2-3 on page 2-27 shows this stack frame also.

**Figure 12-7. Exception Stack Frame**



Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The alignment of the stack frame is controlled via the STKALIGN bit of the Configuration Control Register (CCR).

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC\_RETURN value to the LR. This indicates which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher priority exception occurs during the exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during the exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

## *Exception Return*

An Exception return occurs when the processor is in Handler mode and executes one of the following instructions to load the EXC\_RETURN value into the PC:

- An LDM or POP instruction that loads the PC
- An LDR instruction with the PC as the destination.
- A BX instruction using any register.

EXC\_RETURN is the value loaded into the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest five bits of this value provide information on the return stack and processor mode. [Table 12-10](#) shows the EXC\_RETURN values with a description of the exception return behavior.

All EXC\_RETURN values have bits[31:5] set to one. When this value is loaded into the PC, it indicates to the processor that the exception is complete, and the processor initiates the appropriate exception return sequence.

**Table 12-10. Exception Return Behavior**

EXC_RETURN[31:0]	Description
0xFFFFFFFF1	Return to Handler mode, exception return uses non-floating-point state from the MSP and execution uses MSP after return.
0xFFFFFFFF9	Return to Thread mode, exception return uses state from MSP and execution uses MSP after return.
0xFFFFFFFFD	Return to Thread mode, exception return uses state from the PSP and execution uses PSP after return.
0xFFFFFFF1	Return to Handler mode, exception return uses floating-point-state from MSP and execution uses MSP after return.
0xFFFFFFF9	Return to Thread mode, exception return uses floating-point state from MSP and execution uses MSP after return.
0xFFFFFED	Return to Thread mode, exception return uses floating-point state from PSP and execution uses PSP after return.

#### 12.4.3.8 Fault Handling

Faults are a subset of the exceptions, see “[Exception Model](#)” . The following generate a fault:

- A bus error on:
  - An instruction fetch or vector table load
  - A data access
- An internally-detected error such as an undefined instruction
- An attempt to execute an instruction from a memory region marked as *Non-Executable* (XN).
- A privilege violation or an attempt to access an unmanaged region causing an MPU fault.

#### Fault Types

[Table 12-11](#) shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates that the fault has occurred. See “[Configurable Fault Status Register](#)” for more information about the fault status registers.

**Table 12-11. Faults**

Fault	Handler	Bit Name	Fault Status Register
Bus error on a vector read	Hard fault	VECTTBL	“ <a href="#">Hard Fault Status Register</a> ”
Fault escalated to a hard fault		FORCED	
MPU or default memory map mismatch:		-	
on instruction access	Memory management fault	IACCVIOL	“ <a href="#">MMFSR: Memory Management Fault Status Subregister</a> ”
on data access		DACCVIOL <sup>(2)</sup>	
during exception stacking		MSTKERR	
during exception unstacking		MUNSKERR	
during lazy floating-point state preservation		MLSPERR	
Bus error:		-	
during exception stacking	Bus fault	STKERR	“ <a href="#">BFSR: Bus Fault Status Subregister</a> ”
during exception unstacking		UNSTKERR	
during instruction prefetch		IBUSERR	
during lazy floating-point state preservation		LSPERR	
Precise data bus error		PRECISERR	
Imprecise data bus error		IMPRECISERR	
Attempt to access a coprocessor	Usage fault	NOCP	“ <a href="#">UFSR: Usage Fault Status Subregister</a> ”
Undefined instruction		UNDEFINSTR	
Attempt to enter an invalid instruction set state <sup>(1)</sup>		INVSTATE	
Invalid EXC_RETURN value		INVPC	
Illegal unaligned load or store		UNALIGNED	
Divide By 0		DIVBYZERO	

- Notes: 1. Occurs on an access to an XN region even if the processor does not include an MPU or the MPU is disabled.
2. Attempt to use an instruction set other than the Thumb instruction set, or return to a non load/store-multiple instruction with ICI continuation.

#### Fault Escalation and Hard Faults

All faults exceptions except for hard fault have configurable exception priority, see “[System Handler Priority Registers](#)” . The software can disable the execution of the handlers for these faults, see “[System Handler Control and State Register](#)” .

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler, as described in “[Exception Model](#)” .

In some situations, a fault with configurable priority is treated as a hard fault. This is called *priority escalation*, and the fault is described as *escalated to hard fault*. Escalation to hard fault occurs when:

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to hard fault occurs because a fault handler cannot preempt itself; it must have the same priority as the current priority level.
- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This is because the handler for the new fault cannot preempt the currently executing fault handler.
- An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.
- A fault occurs and the handler for that fault is not enabled.

If a bus fault occurs during a stack push when entering a bus fault handler, the bus fault does not escalate to a hard fault. This means that if a corrupted stack causes a fault, the fault handler executes even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

Note: Only Reset and NMI can preempt the fixed priority hard fault. A hard fault can preempt any exception other than Reset, NMI, or another hard fault.

#### *Fault Status Registers and Fault Address Registers*

The fault status registers indicate the cause of a fault. For bus faults and memory management faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in [Table 12-12](#).

**Table 12-12. Fault Status and Fault Address Registers**

Handler	Status Register Name	Address Register Name	Register Description
Hard fault	SCB_HFSR	-	“ <a href="#">Hard Fault Status Register</a> ”
Memory management fault	MMFSR	SCB_MMFAR	“ <a href="#">MMFSR: Memory Management Fault Status Subregister</a> ” “ <a href="#">MemManage Fault Address Register</a> ”
Bus fault	BFSR	SCB_BFAR	“ <a href="#">BFSR: Bus Fault Status Subregister</a> ” “ <a href="#">Bus Fault Address Register</a> ”
Usage fault	UFSR	-	“ <a href="#">UFSR: Usage Fault Status Subregister</a> ”

#### *Lockup*

The processor enters a lockup state if a hard fault occurs when executing the NMI or hard fault handlers. When the processor is in lockup state, it does not execute any instructions. The processor remains in lockup state until either:

- It is reset
- An NMI occurs
- It is halted by a debugger.

Note: If the lockup state occurs from the NMI handler, a subsequent NMI does not cause the processor to leave the lockup state.

## 12.5 Power Management

The Cortex-M4 processor sleep modes reduce the power consumption:

- Sleep mode stops the processor clock
- Deep sleep mode stops the system clock and switches off the PLL and flash memory.

The SLEEPDEEP bit of the SCR selects which sleep mode is used; see “[System Control Register](#)” .

This section describes the mechanisms for entering sleep mode, and the conditions for waking up from sleep mode.

### 12.5.1 Entering Sleep Mode

This section describes the mechanisms software can use to put the processor into sleep mode.

The system can generate spurious wakeup events, for example a debug operation wakes up the processor. Therefore, the software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back to sleep mode.

#### 12.5.1.1 Wait for Interrupt

The *wait for interrupt* instruction, WFI, causes immediate entry to sleep mode. When the processor executes a WFI instruction it stops executing instructions and enters sleep mode. See “[WFI](#)” for more information.

#### 12.5.1.2 Wait for Event

The *wait for event* instruction (WFE), causes entry to sleep mode conditional on the value of an one-bit event register. When the processor executes a WFE instruction, it checks this register:

- If the register is 0, the processor stops executing instructions and enters sleep mode
- If the register is 1, the processor clears the register to 0 and continues executing instructions without entering sleep mode.

See “[WFE](#)” for more information.

#### 12.5.1.3 Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1 when the processor completes the execution of an exception handler, it returns to Thread mode and immediately enters sleep mode. Use this mechanism in applications that only require the processor to run when an exception occurs.

### 12.5.2 Wakeup from Sleep Mode

The conditions for the processor to wake up depend on the mechanism that cause it to enter sleep mode.

#### 12.5.2.1 Wakeup from WFI or Sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry.

Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this, set the PRIMASK bit to 1 and the FAULTMASK bit to 0. If an interrupt arrives that is enabled and has a higher priority than the current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero. For more information about PRIMASK and FAULTMASK, see “[Exception Mask Registers](#)” .

#### 12.5.2.2 Wakeup from WFE

The processor wakes up if:

- It detects an exception with sufficient priority to cause an exception entry
- It detects an external event signal. See “[External Event Input](#)”
- In a multiprocessor system, another processor in the system executes an SEV instruction.

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause an exception entry. For more information about the SCR, see “[System Control Register](#)” .

### 12.5.2.3 External Event Input

The processor provides an external event input signal. Peripherals can drive this signal, either to wake the processor from WFE, or to set the internal WFE event register to 1 to indicate that the processor must not enter sleep mode on a later WFE instruction. See “[Wait for Event](#)” for more information.

### 12.5.3 Power Management Programming Hints

ISO/IEC C cannot directly generate the WFI and WFE instructions. The CMSIS provides the following functions for these instructions:

```
void __WFE(void) // Wait for Event  
void __WFE(void) // Wait for Interrupt
```

## 12.6 Cortex-M4 Instruction Set

### 12.6.1 Instruction Set Summary

The processor implements a version of the Thumb instruction set. [Table 12-13](#) lists the supported instructions.

- Angle brackets, <>, enclose alternative forms of the operand
- Braces, {}, enclose optional operands
- The Operands column is not exhaustive
- Op2 is a flexible second operand that can be either a register or a constant
- Most instructions can use an optional condition code suffix.

For more information on the instructions and operands, see the instruction descriptions.

**Table 12-13. Cortex-M4 Instructions**

Mnemonic	Operands	Description	Flags
ADC, ADCS	{Rd,} Rn, Op2	Add with Carry	N,Z,C,V
ADD, ADDS	{Rd,} Rn, Op2	Add	N,Z,C,V
ADD, ADDW	{Rd,} Rn, #imm12	Add	N,Z,C,V
ADR	Rd, label	Load PC-relative address	-
AND, ANDS	{Rd,} Rn, Op2	Logical AND	N,Z,C
ASR, ASRS	Rd, Rm, <Rs #n>	Arithmetic Shift Right	N,Z,C
B	label	Branch	-
BFC	Rd, #lsb, #width	Bit Field Clear	-
BFI	Rd, Rn, #lsb, #width	Bit Field Insert	-
BIC, BICS	{Rd,} Rn, Op2	Bit Clear	N,Z,C
BKPT	#imm	Breakpoint	-
BL	label	Branch with Link	-
BLX	Rm	Branch indirect with Link	-
BX	Rm	Branch indirect	-
CBNZ	Rn, label	Compare and Branch if Non Zero	-
CBZ	Rn, label	Compare and Branch if Zero	-
CLREX	-	Clear Exclusive	-
CLZ	Rd, Rm	Count leading zeros	-
CMN	Rn, Op2	Compare Negative	N,Z,C,V
CMP	Rn, Op2	Compare	N,Z,C,V
CPSID	i	Change Processor State, Disable Interrupts	-
CPSIE	i	Change Processor State, Enable Interrupts	-
DMB	-	Data Memory Barrier	-
DSB	-	Data Synchronization Barrier	-
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR	N,Z,C
ISB	-	Instruction Synchronization Barrier	-
IT	-	If-Then condition block	-
LDM	Rn{!}, reglist	Load Multiple registers, increment after	-

**Table 12-13. Cortex-M4 Instructions (Continued)**

Mnemonic	Operands	Description	Flags
LDMDB, LDMEA	Rn{!}, reglist	Load Multiple registers, decrement before	-
LDMFD, LDMIA	Rn{!}, reglist	Load Multiple registers, increment after	-
LDR	Rt, [Rn, #offset]	Load Register with word	-
LDRB, LDRBT	Rt, [Rn, #offset]	Load Register with byte	-
LDRD	Rt, Rt2, [Rn, #offset]	Load Register with two bytes	-
LDREX	Rt, [Rn, #offset]	Load Register Exclusive	-
LDREXB	Rt, [Rn]	Load Register Exclusive with byte	-
LDREXH	Rt, [Rn]	Load Register Exclusive with halfword	-
LDRH, LDRHT	Rt, [Rn, #offset]	Load Register with halfword	-
LDRSB, DRSBT	Rt, [Rn, #offset]	Load Register with signed byte	-
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load Register with signed halfword	-
LDRT	Rt, [Rn, #offset]	Load Register with word	-
LSL, LSLS	Rd, Rm, <Rs>#n>	Logical Shift Left	N,Z,C
LSR, LSRS	Rd, Rm, <Rs>#n>	Logical Shift Right	N,Z,C
MLA	Rd, Rn, Rm, Ra	Multiply with Accumulate, 32-bit result	-
MLS	Rd, Rn, Rm, Ra	Multiply and Subtract, 32-bit result	-
MOV, MOVS	Rd, Op2	Move	N,Z,C
MOVT	Rd, #imm16	Move Top	-
MOVW, MOV	Rd, #imm16	Move 16-bit constant	N,Z,C
MRS	Rd, spec_reg	Move from special register to general register	-
MSR	spec_reg, Rm	Move from general register to special register	N,Z,C,V
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result	N,Z
MVN, MVNS	Rd, Op2	Move NOT	N,Z,C
NOP	-	No Operation	-
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT	N,Z,C
ORR, ORRS	{Rd,} Rn, Op2	Logical OR	N,Z,C
PKHTB, PKHBT	{Rd,} Rn, Rm, Op2	Pack Halfword	-
POP	reglist	Pop registers from stack	-
PUSH	reglist	Push registers onto stack	-
QADD	{Rd,} Rn, Rm	Saturating double and Add	Q
QADD16	{Rd,} Rn, Rm	Saturating Add 16	-
QADD8	{Rd,} Rn, Rm	Saturating Add 8	-
QASX	{Rd,} Rn, Rm	Saturating Add and Subtract with Exchange	-
QDADD	{Rd,} Rn, Rm	Saturating Add	Q
QDSUB	{Rd,} Rn, Rm	Saturating double and Subtract	Q
QSAX	{Rd,} Rn, Rm	Saturating Subtract and Add with Exchange	-
QSUB	{Rd,} Rn, Rm	Saturating Subtract	Q

**Table 12-13. Cortex-M4 Instructions (Continued)**

Mnemonic	Operands	Description	Flags
QSUB16	{Rd,} Rn, Rm	Saturating Subtract 16	-
QSUB8	{Rd,} Rn, Rm	Saturating Subtract 8	-
RBIT	Rd, Rn	Reverse Bits	-
REV	Rd, Rn	Reverse byte order in a word	-
REV16	Rd, Rn	Reverse byte order in each halfword	-
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend	-
ROR, RORS	Rd, Rm, <Rs #n>	Rotate Right	N,Z,C
RRX, RRXS	Rd, Rm	Rotate Right with Extend	N,Z,C
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract	N,Z,C,V
SADD16	{Rd,} Rn, Rm	Signed Add 16	GE
SADD8	{Rd,} Rn, Rm	Signed Add 8 and Subtract with Exchange	GE
SASX	{Rd,} Rn, Rm	Signed Add	GE
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry	N,Z,C,V
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract	-
SDIV	{Rd,} Rn, Rm	Signed Divide	-
SEL	{Rd,} Rn, Rm	Select bytes	-
SEV	-	Send Event	-
SHADD16	{Rd,} Rn, Rm	Signed Halving Add 16	-
SHADD8	{Rd,} Rn, Rm	Signed Halving Add 8	-
SHASX	{Rd,} Rn, Rm	Signed Halving Add and Subtract with Exchange	-
SHSAX	{Rd,} Rn, Rm	Signed Halving Subtract and Add with Exchange	-
SHSUB16	{Rd,} Rn, Rm	Signed Halving Subtract 16	-
SHSUB8	{Rd,} Rn, Rm	Signed Halving Subtract 8	-
SMLABB, SMLABT, SMLATB, SMLATT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate Long (halfwords)	Q
SMLAD, SMLADX	Rd, Rn, Rm, Ra	Signed Multiply Accumulate Dual	Q
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply with Accumulate (32 x 32 + 64), 64-bit result	-
SMLALBB, SMLALBT, SMLALTB, SMLALTT	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long, halfwords	-
SMLALD, SMLALDX	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long Dual	-
SMLAWB, SMLAWT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate, word by halfword	Q
SMLSD	Rd, Rn, Rm, Ra	Signed Multiply Subtract Dual	Q
SMLS LD	RdLo, RdHi, Rn, Rm	Signed Multiply Subtract Long Dual	
SMMLA	Rd, Rn, Rm, Ra	Signed Most significant word Multiply Accumulate	-
SMMLS, SMMLR	Rd, Rn, Rm, Ra	Signed Most significant word Multiply Subtract	-
SMMUL, SMMULR	{Rd,} Rn, Rm	Signed Most significant word Multiply	-
SMUAD	{Rd,} Rn, Rm	Signed dual Multiply Add	Q

**Table 12-13. Cortex-M4 Instructions (Continued)**

Mnemonic	Operands	Description	Flags
SMULBB, SMULBT SMULTB, SMULTT	{Rd,} Rn, Rm	Signed Multiply (halfwords)	-
SMULL	RdLo, RdHi, Rn, Rm	Signed Multiply (32 x 32), 64-bit result	-
SMULWB, SMULWT	{Rd,} Rn, Rm	Signed Multiply word by halfword	-
SMUSD, SMUSDX	{Rd,} Rn, Rm	Signed dual Multiply Subtract	-
SSAT	Rd, #n, Rm {,shift #s}	Signed Saturate	Q
SSAT16	Rd, #n, Rm	Signed Saturate 16	Q
SSAX	{Rd,} Rn, Rm	Signed Subtract and Add with Exchange	GE
SSUB16	{Rd,} Rn, Rm	Signed Subtract 16	-
SSUB8	{Rd,} Rn, Rm	Signed Subtract 8	-
STM	Rn{!}, reglist	Store Multiple registers, increment after	-
STMDB, STMEA	Rn{!}, reglist	Store Multiple registers, decrement before	-
STMFD, STMIA	Rn{!}, reglist	Store Multiple registers, increment after	-
STR	Rt, [Rn, #offset]	Store Register word	-
STRB, STRBT	Rt, [Rn, #offset]	Store Register byte	-
STRD	Rt, Rt2, [Rn, #offset]	Store Register two words	-
STREX	Rd, Rt, [Rn, #offset]	Store Register Exclusive	-
STREXB	Rd, Rt, [Rn]	Store Register Exclusive byte	-
STREXH	Rd, Rt, [Rn]	Store Register Exclusive halfword	-
STRH, STRHT	Rt, [Rn, #offset]	Store Register halfword	-
STRT	Rt, [Rn, #offset]	Store Register word	-
SUB, SUBS	{Rd,} Rn, Op2	Subtract	N,Z,C,V
SUB, SUBW	{Rd,} Rn, #imm12	Subtract	N,Z,C,V
SVC	#imm	Supervisor Call	-
SXTAB	{Rd,} Rn, Rm,{,ROR #}	Extend 8 bits to 32 and add	-
SXTAB16	{Rd,} Rn, Rm,{,ROR #}	Dual extend 8 bits to 16 and add	-
SXTAH	{Rd,} Rn, Rm,{,ROR #}	Extend 16 bits to 32 and add	-
SXTB16	{Rd,} Rm {,ROR #n}	Signed Extend Byte 16	-
SXTB	{Rd,} Rm {,ROR #n}	Sign extend a byte	-
SXTH	{Rd,} Rm {,ROR #n}	Sign extend a halfword	-
TBB	[Rn, Rm]	Table Branch Byte	-
TBH	[Rn, Rm, LSL #1]	Table Branch Halfword	-
TEQ	Rn, Op2	Test Equivalence	N,Z,C
TST	Rn, Op2	Test	N,Z,C
UADD16	{Rd,} Rn, Rm	Unsigned Add 16	GE
UADD8	{Rd,} Rn, Rm	Unsigned Add 8	GE
USAX	{Rd,} Rn, Rm	Unsigned Subtract and Add with Exchange	GE

**Table 12-13. Cortex-M4 Instructions (Continued)**

Mnemonic	Operands	Description	Flags
UHADD16	{Rd,} Rn, Rm	Unsigned Halving Add 16	-
UHADD8	{Rd,} Rn, Rm	Unsigned Halving Add 8	-
UHASX	{Rd,} Rn, Rm	Unsigned Halving Add and Subtract with Exchange	-
UHSAX	{Rd,} Rn, Rm	Unsigned Halving Subtract and Add with Exchange	-
UHSUB16	{Rd,} Rn, Rm	Unsigned Halving Subtract 16	-
UHSUB8	{Rd,} Rn, Rm	Unsigned Halving Subtract 8	-
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract	-
UDIV	{Rd,} Rn, Rm	Unsigned Divide	-
UMAAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply Accumulate Accumulate Long (32 x 32 + 32 +32), 64-bit result	-
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result	-
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32 x 32), 64-bit result	-
UQADD16	{Rd,} Rn, Rm	Unsigned Saturating Add 16	-
UQADD8	{Rd,} Rn, Rm	Unsigned Saturating Add 8	-
UQASX	{Rd,} Rn, Rm	Unsigned Saturating Add and Subtract with Exchange	-
UQSAX	{Rd,} Rn, Rm	Unsigned Saturating Subtract and Add with Exchange	-
UQSUB16	{Rd,} Rn, Rm	Unsigned Saturating Subtract 16	-
UQSUB8	{Rd,} Rn, Rm	Unsigned Saturating Subtract 8	-
USAD8	{Rd,} Rn, Rm	Unsigned Sum of Absolute Differences	-
USADA8	{Rd,} Rn, Rm, Ra	Unsigned Sum of Absolute Differences and Accumulate	-
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate	Q
USAT16	Rd, #n, Rm	Unsigned Saturate 16	Q
UASX	{Rd,} Rn, Rm	Unsigned Add and Subtract with Exchange	GE
USUB16	{Rd,} Rn, Rm	Unsigned Subtract 16	GE
USUB8	{Rd,} Rn, Rm	Unsigned Subtract 8	GE
UXTAB	{Rd,} Rn, Rm,{,ROR #}	Rotate, extend 8 bits to 32 and Add	-
UXTAB16	{Rd,} Rn, Rm,{,ROR #}	Rotate, dual extend 8 bits to 16 and Add	-
UXTAH	{Rd,} Rn, Rm,{,ROR #}	Rotate, unsigned extend and Add Halfword	-
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a byte	-
UXTB16	{Rd,} Rm {,ROR #n}	Unsigned Extend Byte 16	-
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a halfword	-
VABS.F32	Sd, Sm	Floating-point Absolute	-
VADD.F32	{Sd,} Sn, Sm	Floating-point Add	-
VCMP.F32	Sd, <Sm   #0.0>	Compare two floating-point registers, or one floating-point register and zero	FPSCR
VCMPE.F32	Sd, <Sm   #0.0>	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check	FPSCR

**Table 12-13. Cortex-M4 Instructions (Continued)**

Mnemonic	Operands	Description	Flags
VCVT.S32.F32	Sd, Sm	Convert between floating-point and integer	-
VCVT.S16.F32	Sd, Sd, #fbits	Convert between floating-point and fixed point	-
VCVTR.S32.F32	Sd, Sm	Convert between floating-point and integer with rounding	-
VCVT<B H>.F32.F16	Sd, Sm	Converts half-precision value to single-precision	-
VCVTT<B T>.F32.F16	Sd, Sm	Converts single-precision register to half-precision	-
VDIV.F32	{Sd,} Sn, Sm	Floating-point Divide	-
VFMA.F32	{Sd,} Sn, Sm	Floating-point Fused Multiply Accumulate	-
VFNMA.F32	{Sd,} Sn, Sm	Floating-point Fused Negate Multiply Accumulate	-
VFMS.F32	{Sd,} Sn, Sm	Floating-point Fused Multiply Subtract	-
VFNMS.F32	{Sd,} Sn, Sm	Floating-point Fused Negate Multiply Subtract	-
VLDM.F<32 64>	Rn{!}, list	Load Multiple extension registers	-
VLDR.F<32 64>	<Dd Sd>, [Rn]	Load an extension register from memory	-
VLMA.F32	{Sd,} Sn, Sm	Floating-point Multiply Accumulate	-
VLMS.F32	{Sd,} Sn, Sm	Floating-point Multiply Subtract	-
VMOV.F32	Sd, #imm	Floating-point Move immediate	-
VMOV	Sd, Sm	Floating-point Move register	-
VMOV	Sn, Rt	Copy ARM core register to single precision	-
VMOV	Sm, Sm1, Rt, Rt2	Copy 2 ARM core registers to 2 single precision	-
VMOV	Dd[x], Rt	Copy ARM core register to scalar	-
VMOV	Rt, Dn[x]	Copy scalar to ARM core register	-
VMRS	Rt, FPSCR	Move FPSCR to ARM core register or APSR	N,Z,C,V
VMSR	FPSCR, Rt	Move to FPSCR from ARM Core register	FPSCR
VMUL.F32	{Sd,} Sn, Sm	Floating-point Multiply	-
VNEG.F32	Sd, Sm	Floating-point Negate	-
VNMLA.F32	Sd, Sn, Sm	Floating-point Multiply and Add	-
VNMLS.F32	Sd, Sn, Sm	Floating-point Multiply and Subtract	-
VNMUL	{Sd,} Sn, Sm	Floating-point Multiply	-
VPOP	list	Pop extension registers	-
VPUSH	list	Push extension registers	-
VSQRT.F32	Sd, Sm	Calculates floating-point Square Root	-
VSTM	Rn{!}, list	Floating-point register Store Multiple	-
VSTR.F<32 64>	Sd, [Rn]	Stores an extension register to memory	-
VSUB.F<32 64>	{Sd,} Sn, Sm	Floating-point Subtract	-
WFE	-	Wait For Event	-
WFI	-	Wait For Interrupt	-

## 12.6.2 CMSIS Functions

ISO/IEC cannot directly access some Cortex-M4 instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMSIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, the user might have to use inline assembler to access some instructions.

The CMSIS provides the following intrinsic functions to generate instructions that ISO/IEC C code cannot directly access:

**Table 12-14. CMSIS Functions to Generate some Cortex-M4 Instructions**

Instruction	CMSIS Function
CPSIE I	void __enable_irq(void)
CPSID I	void __disable_irq(void)
CPSIE F	void __enable_fault_irq(void)
CPSID F	void __disable_fault_irq(void)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
REV	uint32_t __REV(uint32_t int value)
REV16	uint32_t __REV16(uint32_t int value)
REVSH	uint32_t __REVSH(uint32_t int value)
RBIT	uint32_t __RBIT(uint32_t int value)
SEV	void __SEV(void)
WFE	void __WFE(void)
WFI	void __WFI(void)

The CMSIS also provides a number of functions for accessing the special registers using MRS and MSR instructions:

**Table 12-15. CMSIS Intrinsic Functions to Access the Special Registers**

Special Register	Access	CMSIS Function
PRIMASK	Read	uint32_t __get_PRIMASK (void)
	Write	void __set_PRIMASK (uint32_t value)
FAULTMASK	Read	uint32_t __get_FAULTMASK (void)
	Write	void __set_FAULTMASK (uint32_t value)
BASEPRI	Read	uint32_t __get_BASEPRI (void)
	Write	void __set_BASEPRI (uint32_t value)
CONTROL	Read	uint32_t __get_CONTROL (void)
	Write	void __set_CONTROL (uint32_t value)
MSP	Read	uint32_t __get_MSP (void)
	Write	void __set_MSP (uint32_t TopOfMainStack)
PSP	Read	uint32_t __get_PSP (void)
	Write	void __set_PSP (uint32_t TopOfProcStack)

## 12.6.3 Instruction Descriptions

### 12.6.3.1 Operands

An instruction operand can be an ARM register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. When there is a destination register in the instruction, it is usually specified before the operands.

Operands in some instructions are flexible, can either be a register or a constant. See “[Flexible Second Operand](#)”.

### 12.6.3.2 Restrictions when Using PC or SP

Many instructions have restrictions on whether the *Program Counter* (PC) or *Stack Pointer* (SP) for the operands or destination register can be used. See instruction descriptions for more information.

Note: Bit[0] of any address written to the PC with a BX, BLX, LDM, LDR, or POP instruction must be 1 for correct execution, because this bit indicates the required instruction set, and the Cortex-M4 processor only supports Thumb instructions.

### 12.6.3.3 Flexible Second Operand

Many general data processing instructions have a flexible second operand. This is shown as *Operand2* in the descriptions of the syntax of each instruction.

*Operand2* can be a:

- “Constant”
- “Register with Optional Shift”

#### *Constant*

Specify an *Operand2* constant in the form:

`#constant`

where *constant* can be:

- Any constant that can be produced by shifting an 8-bit value left by any number of bits within a 32-bit word
- Any constant of the form 0x00XY00XY
- Any constant of the form 0xXY00XY00
- Any constant of the form 0xXYXYXYXY.

Note: In the constants shown above, X and Y are hexadecimal digits.

In addition, in a small number of instructions, *constant* can take a wider range of values. These are described in the individual instruction descriptions.

When an *Operand2* constant is used with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to bit[31] of the constant, if the constant is greater than 255 and can be produced by shifting an 8-bit value. These instructions do not affect the carry flag if *Operand2* is any other constant.

#### *Instruction Substitution*

The assembler might be able to produce an equivalent instruction in cases where the user specifies a constant that is not permitted. For example, an assembler might assemble the instruction `CMP Rd, #0xFFFFFFF` as the equivalent instruction `CMN Rd, #0x2`.

#### *Register with Optional Shift*

Specify an *Operand2* register in the form:

`Rm { , shift }`

where:

*Rm* is the register holding the data for the second operand.

*shift* is an optional shift to be applied to *Rm*. It can be one of:

`ASR #n` arithmetic shift right *n* bits,  $1 \leq n \leq 32$ .

LSL #n	logical shift left $n$ bits, $1 \leq n \leq 31$ .
LSR #n	logical shift right $n$ bits, $1 \leq n \leq 32$ .
ROR #n	rotate right $n$ bits, $1 \leq n \leq 31$ .
RRX	rotate right one bit, with extend.
-	if omitted, no shift occurs, equivalent to LSL #0.

If the user omits the shift, or specifies LSL #0, the instruction uses the value in  $Rm$ .

If the user specifies a shift, the shift is applied to the value in  $Rm$ , and the resulting 32-bit value is used by the instruction. However, the contents in the register  $Rm$  remains unchanged. Specifying a register with shift also updates the carry flag when used with certain instructions. For information on the shift operations and how they affect the carry flag, see “Flexible Second Operand”

#### 12.6.3.4 Shift Operations

Register shift operations move the bits in a register left or right by a specified number of bits, the *shift length*. Register shift can be performed:

- Directly by the instructions ASR, LSR, LSL, ROR, and RRX, and the result is written to a destination register
- During the calculation of *Operand2* by the instructions that specify the second operand as a register with shift. See “Flexible Second Operand”. The result is used by the instruction.

The permitted shift lengths depend on the shift type and the instruction. If the shift length is 0, no shift occurs. Register shift operations update the carry flag except when the specified shift length is 0. The following sub-sections describe the various shift operations and how they affect the carry flag. In these descriptions,  $Rm$  is the register containing the value to be shifted, and  $n$  is the shift length.

#### ASR

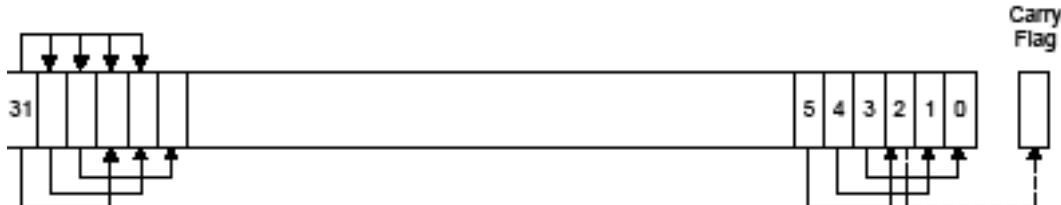
Arithmetic shift right by  $n$  bits moves the left-hand 32- $n$  bits of the register,  $Rm$ , to the right by  $n$  places, into the right-hand 32- $n$  bits of the result. And it copies the original bit[31] of the register into the left-hand  $n$  bits of the result. See Figure 12-8.

The ASR # $n$  operation can be used to divide the value in the register  $Rm$  by  $2^n$ , with the result being rounded towards negative-infinity.

When the instruction is ASRS or when ASR # $n$  is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[ $n$ -1], of the register  $Rm$ .

- If  $n$  is 32 or more, then all the bits in the result are set to the value of bit[31] of  $Rm$ .
- If  $n$  is 32 or more and the carry flag is updated, it is updated to the value of bit[31] of  $Rm$ .

Figure 12-8. ASR #3



#### LSR

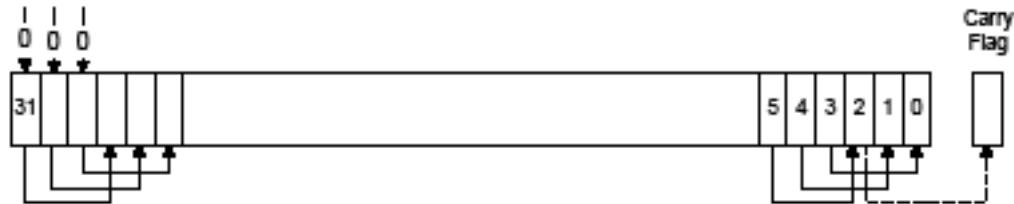
Logical shift right by  $n$  bits moves the left-hand 32- $n$  bits of the register  $Rm$ , to the right by  $n$  places, into the right-hand 32- $n$  bits of the result. And it sets the left-hand  $n$  bits of the result to 0. See Figure 12-9.

The LSR # $n$  operation can be used to divide the value in the register  $Rm$  by  $2^n$ , if the value is regarded as an unsigned integer.

When the instruction is LSRS or when LSR # $n$  is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[ $n$ -1], of the register  $Rm$ .

- If  $n$  is 32 or more, then all the bits in the result are cleared to 0.
- If  $n$  is 33 or more and the carry flag is updated, it is updated to 0.

**Figure 12-9. LSR #3**



### LSL

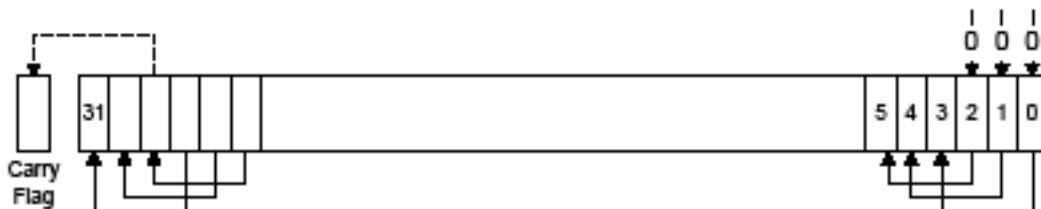
Logical shift left by  $n$  bits moves the right-hand 32- $n$  bits of the register  $Rm$ , to the left by  $n$  places, into the left-hand 32- $n$  bits of the result; and it sets the right-hand  $n$  bits of the result to 0. See [Figure 12-10](#).

The LSL # $n$  operation can be used to multiply the value in the register  $Rm$  by  $2^n$ , if the value is regarded as an unsigned integer or a two's complement signed integer. Overflow can occur without warning.

When the instruction is LSLS or when LSL # $n$ , with non-zero  $n$ , is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[32- $n$ ], of the register  $Rm$ . These instructions do not affect the carry flag when used with LSL #0.

- If  $n$  is 32 or more, then all the bits in the result are cleared to 0.
- If  $n$  is 33 or more and the carry flag is updated, it is updated to 0.

**Figure 12-10. LSL #3**



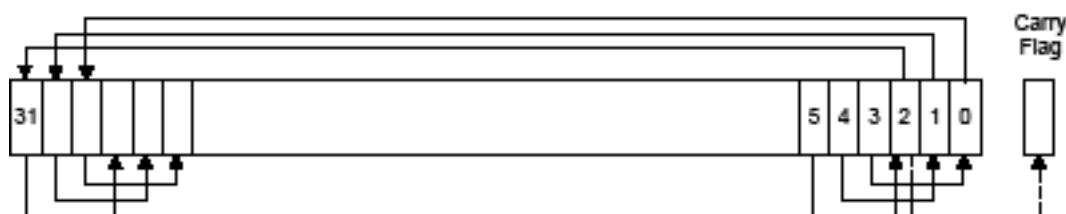
### ROT

Rotate right by  $n$  bits moves the left-hand 32- $n$  bits of the register  $Rm$ , to the right by  $n$  places, into the right-hand 32- $n$  bits of the result; and it moves the right-hand  $n$  bits of the register into the left-hand  $n$  bits of the result. See [Figure 12-11](#).

When the instruction is RORS or when ROR # $n$  is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit rotation, bit[ $n$ -1], of the register  $Rm$ .

- If  $n$  is 32, then the value of the result is same as the value in  $Rm$ , and if the carry flag is updated, it is updated to bit[31] of  $Rm$ .
- ROR with shift length,  $n$ , more than 32 is the same as ROR with shift length  $n$ -32.

**Figure 12-11. ROR #3**

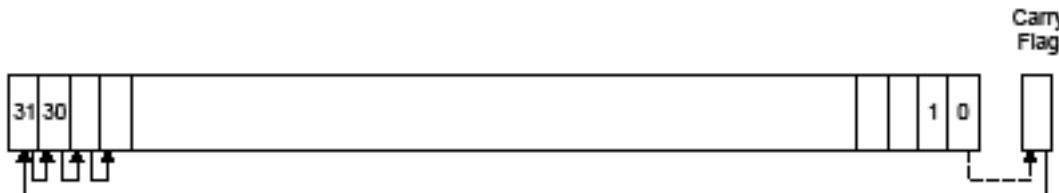


## RRX

Rotate right with extend moves the bits of the register  $Rm$  to the right by one bit; and it copies the carry flag into bit[31] of the result. See [Figure 12-12](#).

When the instruction is RRXS or when RRX is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to bit[0] of the register  $Rm$ .

**Figure 12-12. RRX**



### 12.6.3.5 Address Alignment

An aligned access is an operation where a word-aligned address is used for a word, dual word, or multiple word access, or where a halfword-aligned address is used for a halfword access. Byte accesses are always aligned.

The Cortex-M4 processor supports unaligned access only for the following instructions:

- LDR, LDRT
- LDRH, LDRHT
- LDRSH, LDRSHT
- STR, STRT
- STRH, STRHT

All other load and store instructions generate a usage fault exception if they perform an unaligned access, and therefore their accesses must be address-aligned. For more information about usage faults, see ["Fault Handling"](#).

Unaligned accesses are usually slower than aligned accesses. In addition, some memory regions might not support unaligned accesses. Therefore, ARM recommends that programmers ensure that accesses are aligned. To avoid accidental generation of unaligned accesses, use the UNALIGN\_TRP bit in the Configuration and Control Register to trap all unaligned accesses, see ["Configuration and Control Register"](#).

### 12.6.3.6 PC-relative Expressions

A PC-relative expression or *label* is a symbol that represents the address of an instruction or literal data. It is represented in the instruction as the PC value plus or minus a numeric offset. The assembler calculates the required offset from the label and the address of the current instruction. If the offset is too big, the assembler produces an error.

- For B, BL, CBNZ, and CBZ instructions, the value of the PC is the address of the current instruction plus 4 bytes.
- For all other instructions that use labels, the value of the PC is the address of the current instruction plus 4 bytes, with bit[1] of the result cleared to 0 to make it word-aligned.
- Your assembler might permit other syntaxes for PC-relative expressions, such as a label plus or minus a number, or an expression of the form [PC, #number].

### 12.6.3.7 Conditional Execution

Most data processing instructions can optionally update the condition flags in the *Application Program Status Register* (APSR) according to the result of the operation, see ["Application Program Status Register"](#). Some instructions update all flags, and some only update a subset. If a flag is not updated, the original value is preserved. See the instruction descriptions for the flags they affect.

An instruction can be executed conditionally, based on the condition flags set in another instruction, either:

- Immediately after the instruction that updated the flags
- After any number of intervening instructions that have not updated the flags.

Conditional execution is available by using conditional branches or by adding condition code suffixes to instructions. See [Table 12-16](#) for a list of the suffixes to add to instructions to make them conditional instructions. The condition code suffix enables the processor to test a condition based on the flags. If the condition test of a conditional instruction fails, the instruction:

- Does not execute
- Does not write any value to its destination register
- Does not affect any of the flags
- Does not generate any exception.

Conditional instructions, except for conditional branches, must be inside an If-Then instruction block. See [“IT”](#) for more information and restrictions when using the IT instruction. Depending on the vendor, the assembler might automatically insert an IT instruction if there are conditional instructions outside the IT block.

The CBZ and CBNZ instructions are used to compare the value of a register against zero and branch on the result.

This section describes:

- [“Condition Flags”](#)
- [“Condition Code Suffixes”](#).

#### *Condition Flags*

The APSR contains the following condition flags:

N	Set to 1 when the result of the operation was negative, cleared to 0 otherwise.
Z	Set to 1 when the result of the operation was zero, cleared to 0 otherwise.
C	Set to 1 when the operation resulted in a carry, cleared to 0 otherwise.
V	Set to 1 when the operation caused overflow, cleared to 0 otherwise.

For more information about the APSR, see [“Program Status Register”](#).

A carry occurs:

- If the result of an addition is greater than or equal to  $2^{32}$
- If the result of a subtraction is positive or zero
- As the result of an inline barrel shifter operation in a move or logical instruction.

An overflow occurs when the sign of the result, in bit[31], does not match the sign of the result, had the operation been performed at infinite precision, for example:

- If adding two negative values results in a positive value
- If adding two positive values results in a negative value
- If subtracting a positive value from a negative value generates a positive value
- If subtracting a negative value from a positive value generates a negative value.

The Compare operations are identical to subtracting, for CMP, or adding, for CMN, except that the result is discarded. See the instruction descriptions for more information.

Note: Most instructions update the status flags only if the S suffix is specified. See the instruction descriptions for more information.

#### *Condition Code Suffixes*

The instructions that can be conditional have an optional condition code, shown in syntax descriptions as {cond}. Conditional execution requires a preceding IT instruction. An instruction with a condition code is only executed if the condition code flags in the APSR meet the specified condition. [Table 12-16](#) shows the condition codes to use.

A conditional execution can be used with the IT instruction to reduce the number of branch instructions in code.

Table 12-16 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

**Table 12-16. Condition Code Suffixes**

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned $\geq$
CC or LO	C = 0	Lower, unsigned <
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned >
LS	C = 0 or Z = 1	Lower or same, unsigned $\leq$
GE	N = V	Greater than or equal, signed $\geq$
LT	N != V	Less than, signed <
GT	Z = 0 and N = V	Greater than, signed >
LE	Z = 1 and N != V	Less than or equal, signed $\leq$
AL	Can have any value	Always. This is the default when no suffix is specified.

### Absolute Value

The example below shows the use of a conditional instruction to find the absolute value of a number. R0 = ABS(R1).

```
MOVS    R0, R1      ; R0 = R1, setting flags
IT      MI          ; IT instruction for the negative condition
RSBMI   R0, R1, #0    ; If negative, R0 = -R1
```

### Compare and Update Value

The example below shows the use of conditional instructions to update the value of R4 if the signed values R0 is greater than R1 and R2 is greater than R3.

```
CMP    R0, R1      ; Compare R0 and R1, setting flags
ITT    GT          ; ITT instruction for the two GT conditions
CMPGT   R2, R3      ; If 'greater than', compare R2 and R3, setting flags
MOVGT   R4, R5      ; If still 'greater than', do R4 = R5
```

### 12.6.3.8 Instruction Width Selection

There are many instructions that can generate either a 16-bit encoding or a 32-bit encoding depending on the operands and destination register specified. For some of these instructions, the user can force a specific instruction size by using an instruction width suffix. The .W suffix forces a 32-bit instruction encoding. The .N suffix forces a 16-bit instruction encoding.

If the user specifies an instruction width suffix and the assembler cannot generate an instruction encoding of the requested width, it generates an error.

Note: In some cases, it might be necessary to specify the .W suffix, for example if the operand is the label of an instruction or literal data, as in the case of branch instructions. This is because the assembler might not automatically generate the right size encoding.

To use an instruction width suffix, place it immediately after the instruction mnemonic and condition code, if any. The example below shows instructions with the instruction width suffix.

```
BCS.W  label      ; creates a 32-bit instruction even for a short
                  ; branch
ADDS.W R0, R0, R1 ; creates a 32-bit instruction even though the same
                  ; operation can be done by a 16-bit instruction
```

## 12.6.4 Memory Access Instructions

The table below shows the memory access instructions:

**Table 12-17. Memory Access Instructions**

Mnemonic	Description
ADR	Load PC-relative address
CLREX	Clear Exclusive
LDM{mode}	Load Multiple registers
LDR{type}	Load Register using immediate offset
LDR{type}	Load Register using register offset
LDR{type}T	Load Register with unprivileged access
LDR	Load Register using PC-relative address
LDRD	Load Register Dual
LDREX{type}	Load Register Exclusive
POP	Pop registers from stack
PUSH	Push registers onto stack
STM{mode}	Store Multiple registers
STR{type}	Store Register using immediate offset
STR{type}	Store Register using register offset
STR{type}T	Store Register with unprivileged access
STREX{type}	Store Register Exclusive

### 12.6.4.1 ADR

Load PC-relative address.

Syntax

```
ADR{cond} Rd, label
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

label is a PC-relative expression. See “[PC-relative Expressions](#)” .

Operation

ADR determines the address by adding an immediate value to the PC, and writes the result to the destination register.

ADR produces position-independent code, because the address is PC-relative.

If ADR is used to generate a target address for a BX or BLX instruction, ensure that bit[0] of the address generated is set to 1 for correct execution.

Values of *label* must be within the range of –4095 to +4095 from the address in the PC.

Note: The user might have to use the .W suffix to get the maximum offset range or to generate addresses that are not word-aligned. See “[Instruction Width Selection](#)”.

#### Restrictions

*Rd* must not be SP and must not be PC.

#### Condition Flags

This instruction does not change the flags.

#### Examples

```
ADR      R1, TextMessage ; Write address value of a location labelled as
                           ; TextMessage to R1
```

### 12.6.4.2 LDR and STR, Immediate Offset

Load and Store with immediate offset, pre-indexed immediate offset, or post-indexed immediate offset.

#### Syntax

<i>op{type}{cond} Rt, [Rn {, #offset}]</i>	; immediate offset
<i>op{type}{cond} Rt, [Rn, #offset]!</i>	; pre-indexed
<i>op{type}{cond} Rt, [Rn], #offset</i>	; post-indexed
<i>opD{cond} Rt, Rt2, [Rn {, #offset}]</i>	; immediate offset, two words
<i>opD{cond} Rt, Rt2, [Rn, #offset]!</i>	; pre-indexed, two words
<i>opD{cond} Rt, Rt2, [Rn], #offset</i>	; post-indexed, two words

where:

*op* is one of:

LDR Load Register.

STR Store Register.

*type* is one of:

B unsigned byte, zero extend to 32 bits on loads.

SB signed byte, sign extend to 32 bits (LDR only).

H unsigned halfword, zero extend to 32 bits on loads.

SH signed halfword, sign extend to 32 bits (LDR only).

- omit, for word.

*cond* is an optional condition code, see “[Conditional Execution](#)”.

*Rt* is the register to load or store.

*Rn* is the register on which the memory address is based.

*offset* is an offset from *Rn*. If *offset* is omitted, the address is the contents of *Rn*.

*Rt2* is the additional register to load or store for two-word operations.

#### Operation

LDR instructions load one or two registers with a value from memory.

STR instructions store one or two register values to memory.

Load and store instructions with immediate offset can use the following addressing modes:

#### Offset Addressing

The offset value is added to or subtracted from the address obtained from the register *Rn*. The result is used as the address for the memory access. The register *Rn* is unaltered. The assembly language syntax for this mode is:

[*Rn*, #*offset*]

#### Pre-indexed Addressing

The offset value is added to or subtracted from the address obtained from the register  $Rn$ . The result is used as the address for the memory access and written back into the register  $Rn$ . The assembly language syntax for this mode is:

`[Rn, #offset]!`

### Post-indexed Addressing

The address obtained from the register  $Rn$  is used as the address for the memory access. The offset value is added to or subtracted from the address, and written back into the register  $Rn$ . The assembly language syntax for this mode is:

`[Rn], #offset`

The value to load or store can be a byte, halfword, word, or two words. Bytes and halfwords can either be signed or unsigned. See “[Address Alignment](#)”.

The table below shows the ranges of offset for immediate, pre-indexed and post-indexed forms.

**Table 12-18. Offset Ranges**

Instruction Type	Immediate Offset	Pre-indexed	Post-indexed
Word, halfword, signed halfword, byte, or signed byte	-255 to 4095	-255 to 255	-255 to 255
Two words	multiple of 4 in the range -1020 to 1020	multiple of 4 in the range -1020 to 1020	multiple of 4 in the range -1020 to 1020

### Restrictions

For load instructions:

- $Rt$  can be SP or PC for word loads only
- $Rt$  must be different from  $Rt2$  for two-word loads
- $Rn$  must be different from  $Rt$  and  $Rt2$  in the pre-indexed or post-indexed forms.

When  $Rt$  is PC in a word load instruction:

- Bit[0] of the loaded value must be 1 for correct execution
- A branch occurs to the address created by changing bit[0] of the loaded value to 0
- If the instruction is conditional, it must be the last instruction in the IT block.

For store instructions:

- $Rt$  can be SP for word stores only
- $Rt$  must not be PC
- $Rn$  must not be PC
- $Rn$  must be different from  $Rt$  and  $Rt2$  in the pre-indexed or post-indexed forms.

### Condition Flags

These instructions do not change the flags.

### Examples

```

LDR    R8, [R10]          ; Loads R8 from the address in R10.
LDRNE  R2, [R5, #960]!    ; Loads (conditionally) R2 from a word
                           ; 960 bytes above the address in R5, and
                           ; increments R5 by 960.
STR    R2, [R9,#const-struc] ; const-struc is an expression evaluating
                           ; to a constant in the range 0-4095.
STRH   R3, [R4], #4        ; Store R3 as halfword data into address in
                           ; R4, then increment R4 by 4
LDRD   R8, R9, [R3, #0x20] ; Load R8 from a word 32 bytes above the
                           ; address in R3, and load R9 from a word 36
                           ; bytes above the address in R3
STRD   R0, R1, [R8], #-16   ; Store R0 to address in R8, and store R1 to
                           ; a word 4 bytes above the address in R8,
                           ; and then decrement R8 by 16.

```

#### 12.6.4.3 LDR and STR, Register Offset

Load and Store with register offset.

##### Syntax

*op{type}{cond} Rt, [Rn, Rm {, LSL #n}]*

where:

*op*      is one of:

  LDR    Load Register.

  STR    Store Register.

*type*    is one of:

  B    unsigned byte, zero extend to 32 bits on loads.

  SB    signed byte, sign extend to 32 bits (LDR only).

  H    unsigned halfword, zero extend to 32 bits on loads.

  SH    signed halfword, sign extend to 32 bits (LDR only).

  -    omit, for word.

*cond*    is an optional condition code, see “[Conditional Execution](#)” .

*Rt*      is the register to load or store.

*Rn*      is the register on which the memory address is based.

*Rm*      is a register containing a value to be used as the offset.

*LSL #n*    is an optional shift, with *n* in the range 0 to 3.

##### Operation

LDR instructions load a register with a value from memory.

STR instructions store a register value into memory.

The memory address to load from or store to is at an offset from the register *Rn*. The offset is specified by the register *Rm* and can be shifted left by up to 3 bits using LSL.

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See “[Address Alignment](#)” .

##### Restrictions

In these instructions:

- *Rn* must not be PC
- *Rm* must not be SP and must not be PC
- *Rt* can be SP only for word loads and word stores
- *Rt* can be PC only for word loads.

When *Rt* is PC in a word load instruction:

- Bit[0] of the loaded value must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- If the instruction is conditional, it must be the last instruction in the IT block.

##### Condition Flags

These instructions do not change the flags.

##### Examples

```
STR    R0, [R5, R1]          ; Store value of R0 into an address equal to  
                           ; sum of R5 and R1  
LDRSB  R0, [R5, R1, LSL #1] ; Read byte value from an address equal to  
                           ; sum of R5 and two times R1, sign extended it  
                           ; to a word value and put it in R0
```

```
STR      R0, [R1, R2, LSL #2] ; Stores R0 to an address equal to sum of R1  
; and four times R2
```

#### 12.6.4.4 LDR and STR, Unprivileged

Load and Store with unprivileged access.

Syntax

```
op{type}T{cond} Rt, [Rn {, #offset}] ; immediate offset
```

where:

op is one of:

LDR Load Register.

STR Store Register.

type is one of:

B unsigned byte, zero extend to 32 bits on loads.

SB signed byte, sign extend to 32 bits (LDR only).

H unsigned halfword, zero extend to 32 bits on loads.

SH signed halfword, sign extend to 32 bits (LDR only).

- omit, for word.

cond is an optional condition code, see “[Conditional Execution](#)” .

Rt is the register to load or store.

Rn is the register on which the memory address is based.

offset is an offset from Rn and can be 0 to 255.

If offset is omitted, the address is the value in Rn.

Operation

These load and store instructions perform the same function as the memory access instructions with immediate offset, see “[LDR and STR, Immediate Offset](#)” . The difference is that these instructions have only unprivileged access even when used in privileged software.

When used in unprivileged software, these instructions behave in exactly the same way as normal memory access instructions with immediate offset.

Restrictions

In these instructions:

- Rn must not be PC
- Rt must not be SP and must not be PC.

Condition Flags

These instructions do not change the flags.

Examples

```
STRBTEQ R4, [R7]      ; Conditionally store least significant byte in  
; R4 to an address in R7, with unprivileged access  
LDRHT    R2, [R2, #8]   ; Load halfword value from an address equal to  
; sum of R2 and 8 into R2, with unprivileged access
```

#### 12.6.4.5 LDR, PC-relative

Load register from memory.

##### Syntax

```
LDR{type}{cond} Rt, label  
LDRD{cond} Rt, Rt2, label ; Load two words
```

where:

type is one of:

B unsigned byte, zero extend to 32 bits.

SB signed byte, sign extend to 32 bits.

H unsigned halfword, zero extend to 32 bits.

SH signed halfword, sign extend to 32 bits.

- omit, for word.

cond is an optional condition code, see “[Conditional Execution](#)” .

Rt is the register to load or store.

Rt2 is the second register to load or store.

label is a PC-relative expression. See “[PC-relative Expressions](#)” .

##### Operation

LDR loads a register with a value from a PC-relative memory address. The memory address is specified by a label or by an offset from the PC.

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See “[Address Alignment](#)” .

*label* must be within a limited range of the current instruction. The table below shows the possible offsets between *label* and the PC.

**Table 12-19. Offset Ranges**

Instruction Type	Offset Range
Word, halfword, signed halfword, byte, signed byte	-4095 to 4095
Two words	-1020 to 1020

The user might have to use the .W suffix to get the maximum offset range. See “[Instruction Width Selection](#)” .

##### Restrictions

In these instructions:

- Rt can be SP or PC only for word loads
- Rt2 must not be SP and must not be PC
- Rt must be different from Rt2.

When Rt is PC in a word load instruction:

- Bit[0] of the loaded value must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- If the instruction is conditional, it must be the last instruction in the IT block.

##### Condition Flags

These instructions do not change the flags.

##### Examples

```
LDR    R0, LookUpTable ; Load R0 with a word of data from an address  
                      ; labelled as LookUpTable  
LDRSB  R7, localdata   ; Load a byte value from an address labelled
```

```
; as localdata, sign extend it to a word  
; value, and put it in R7
```

#### 12.6.4.6 LDM and STM

Load and Store Multiple registers.

Syntax

```
op{addr_mode}{cond} Rn{!}, reglist
```

where:

op is one of:

LDM Load Multiple registers.

STM Store Multiple registers.

addr\_mode is any one of the following:

IA Increment address After each access. This is the default.

DB Decrement address Before each access.

cond is an optional condition code, see “[Conditional Execution](#)” .

Rn is the register on which the memory addresses are based.

! is an optional writeback suffix.

If ! is present, the final address, that is loaded from or stored to, is written back into Rn.

reglist is a list of one or more registers to be loaded or stored, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range, see “[Examples](#)” .

LDM and LDMFD are synonyms for LDMIA. LDMFD refers to its use for popping data from Full Descending stacks.

LDMEA is a synonym for LDMDB, and refers to its use for popping data from Empty Ascending stacks.

STM and STMEA are synonyms for STMIA. STMEA refers to its use for pushing data onto Empty Ascending stacks.

STMD is a synonym for STMDB, and refers to its use for pushing data onto Full Descending stacks

Operation

LDM instructions load the registers in reglist with word values from memory addresses based on Rn.

STM instructions store the word values in the registers in reglist to memory addresses based on Rn.

For LDM, LDMIA, LDMFD, STM, STMIA, and STMEA the memory addresses used for the accesses are at 4-byte intervals ranging from Rn to Rn + 4 \* (n-1), where n is the number of registers in reglist. The accesses happens in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest number register using the highest memory address. If the writeback suffix is specified, the value of Rn + 4 \* (n-1) is written back to Rn.

For LDMDB, LDMEA, STMDB, and STMD the memory addresses used for the accesses are at 4-byte intervals ranging from Rn to Rn - 4 \* (n-1), where n is the number of registers in reglist. The accesses happen in order of decreasing register numbers, with the highest numbered register using the highest memory address and the lowest number register using the lowest memory address. If the writeback suffix is specified, the value of Rn - 4 \* (n-1) is written back to Rn.

The PUSH and POP instructions can be expressed in this form. See “[PUSH and POP](#)” for details.

Restrictions

In these instructions:

- Rn must not be PC
- reglist must not contain SP
- In any STM instruction, reglist must not contain PC
- In any LDM instruction, reglist must not contain PC if it contains LR
- reglist must not contain Rn if the writeback suffix is specified.

When PC is in *reglist* in an LDM instruction:

- Bit[0] of the value loaded to the PC must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- If the instruction is conditional, it must be the last instruction in the IT block.

#### Condition Flags

These instructions do not change the flags.

#### Examples

```
LDM      R8 , {R0,R2,R9} ; LDMIA is a synonym for LDM  
STMDB   R1! , {R3-R6,R11,R12}
```

#### Incorrect Examples

```
STM      R5! , {R5,R4,R9} ; Value stored for R5 is unpredictable  
LDM      R2, {}           ; There must be at least one register in the list
```

### 12.6.4.7 PUSH and POP

Push registers onto, and pop registers off a full-descending stack.

#### Syntax

```
PUSH{cond} reglist  
POP{cond} reglist
```

where:

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*reglist* is a non-empty list of registers, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range.

PUSH and POP are synonyms for STMDB and LDM (or LDMIA) with the memory addresses for the access based on SP, and with the final address for the access written back to the SP. PUSH and POP are the preferred mnemonics in these cases.

#### Operation

PUSH stores registers on the stack in order of decreasing the register numbers, with the highest numbered register using the highest memory address and the lowest numbered register using the lowest memory address.

POP loads registers from the stack in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

See “[LDM and STM](#)” for more information.

#### Restrictions

In these instructions:

- *reglist* must not contain SP
- For the PUSH instruction, *reglist* must not contain PC
- For the POP instruction, *reglist* must not contain PC if it contains LR.

When PC is in *reglist* in a POP instruction:

- Bit[0] of the value loaded to the PC must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- If the instruction is conditional, it must be the last instruction in the IT block.

#### Condition Flags

These instructions do not change the flags.

#### Examples

```

PUSH    {R0,R4-R7}
PUSH    {R2,LR}
POP     {R0,R10,PC}

```

#### 12.6.4.8 LDREX and STREX

Load and Store Register Exclusive.

##### Syntax

```

LDREX{cond} Rt, [Rn {, #offset}]
STREX{cond} Rd, Rt, [Rn {, #offset}]
LDREXB{cond} Rt, [Rn]
STREXB{cond} Rd, Rt, [Rn]
LDREXH{cond} Rt, [Rn]
STREXH{cond} Rd, Rt, [Rn]

```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .
- Rd is the destination register for the returned status.
- Rt is the register to load or store.
- Rn is the register on which the memory address is based.
- offset is an optional offset applied to the value in Rn.  
If offset is omitted, the address is the value in Rn.

##### Operation

LDREX, LDREXB, and LDREXH load a word, byte, and halfword respectively from a memory address.

STREX, STREXB, and STREXH attempt to store a word, byte, and halfword respectively to a memory address. The address used in any Store-Exclusive instruction must be the same as the address in the most recently executed Load-exclusive instruction. The value stored by the Store-Exclusive instruction must also have the same data size as the value loaded by the preceding Load-exclusive instruction. This means software must always use a Load-exclusive instruction and a matching Store-Exclusive instruction to perform a synchronization operation, see “[Synchronization Primitives](#)” .

If an Store-Exclusive instruction performs the store, it writes 0 to its destination register. If it does not perform the store, it writes 1 to its destination register. If the Store-Exclusive instruction writes 0 to the destination register, it is guaranteed that no other process in the system has accessed the memory location between the Load-exclusive and Store-Exclusive instructions.

For reasons of performance, keep the number of instructions between corresponding Load-Exclusive and Store-Exclusive instruction to a minimum.

The result of executing a Store-Exclusive instruction to an address that is different from that used in the preceding Load-Exclusive instruction is unpredictable.

##### Restrictions

In these instructions:

- Do not use PC
- Do not use SP for Rd and Rt
- For STREX, Rd must be different from both Rt and Rn
- The value of offset must be a multiple of four in the range 0-1020.

##### Condition Flags

These instructions do not change the flags.

##### Examples

```

MOV      R1, #0x1          ; Initialize the 'lock taken' value try
LDREX   R0, [LockAddr]    ; Load the lock value

```

```

    CMP      R0, #0          ; Is the lock free?
    ITT      EQ              ; IT instruction for STREXEQ and CMPEQ
    STREXEQ R0, R1, [LockAddr] ; Try and claim the lock
    CMPEQ    R0, #0          ; Did this succeed?
    BNE      try             ; No - try again
    ....                ; Yes - we have the lock

```

#### 12.6.4.9 CLREX

Clear Exclusive.

##### Syntax

`CLREX{cond}`

where:

`cond` is an optional condition code, see “[Conditional Execution](#)” .

##### Operation

Use CLREX to make the next STREX, STREXB, or STREXH instruction write 1 to its destination register and fail to perform the store. It is useful in exception handler code to force the failure of the store exclusive if the exception occurs between a load exclusive instruction and the matching store exclusive instruction in a synchronization operation.

See “[Synchronization Primitives](#)” for more information.

##### Condition Flags

These instructions do not change the flags.

##### Examples

`CLREX`

#### 12.6.5 General Data Processing Instructions

The table below shows the data processing instructions:

**Table 12-20. Data Processing Instructions**

Mnemonic	Description
ADC	Add with Carry
ADD	Add
ADDW	Add
AND	Logical AND
ASR	Arithmetic Shift Right
BIC	Bit Clear
CLZ	Count leading zeros
CMN	Compare Negative
CMP	Compare
EOR	Exclusive OR
LSL	Logical Shift Left
LSR	Logical Shift Right
MOV	Move
MOVT	Move Top
MOVW	Move 16-bit constant

**Table 12-20. Data Processing Instructions (Continued)**

Mnemonic	Description
MVN	Move NOT
ORN	Logical OR NOT
ORR	Logical OR
RBIT	Reverse Bits
REV	Reverse byte order in a word
REV16	Reverse byte order in each halfword
REVSH	Reverse byte order in bottom halfword and sign extend
ROR	Rotate Right
RRX	Rotate Right with Extend
RSB	Reverse Subtract
SADD16	Signed Add 16
SADD8	Signed Add 8
SASX	Signed Add and Subtract with Exchange
SSAX	Signed Subtract and Add with Exchange
SBC	Subtract with Carry
SHADD16	Signed Halving Add 16
SHADD8	Signed Halving Add 8
SHASX	Signed Halving Add and Subtract with Exchange
SHSAX	Signed Halving Subtract and Add with Exchange
SHSUB16	Signed Halving Subtract 16
SHSUB8	Signed Halving Subtract 8
SSUB16	Signed Subtract 16
SSUB8	Signed Subtract 8
SUB	Subtract
SUBW	Subtract
TEQ	Test Equivalence
TST	Test
UADD16	Unsigned Add 16
UADD8	Unsigned Add 8
UASX	Unsigned Add and Subtract with Exchange
USAX	Unsigned Subtract and Add with Exchange
UHADD16	Unsigned Halving Add 16
UHADD8	Unsigned Halving Add 8
UHASX	Unsigned Halving Add and Subtract with Exchange
UHSAX	Unsigned Halving Subtract and Add with Exchange
UHSUB16	Unsigned Halving Subtract 16
UHSUB8	Unsigned Halving Subtract 8

**Table 12-20. Data Processing Instructions (Continued)**

Mnemonic	Description
USAD8	Unsigned Sum of Absolute Differences
USADA8	Unsigned Sum of Absolute Differences and Accumulate
USUB16	Unsigned Subtract 16
USUB8	Unsigned Subtract 8

### 12.6.5.1 ADD, ADC, SUB, SBC, and RSB

Add, Add with carry, Subtract, Subtract with carry, and Reverse Subtract.

Syntax

```
op{S}{cond} {Rd,} Rn, Operand2  
op{cond} {Rd,} Rn, #imm12 ; ADD and SUB only
```

where:

op is one of:

ADD Add.

ADC Add with Carry.

SUB Subtract.

SBC Subtract with Carry.

RSB Reverse Subtract.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “[Conditional Execution](#)” .

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register. If Rd is omitted, the destination register is Rn.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See “[Flexible Second Operand](#)” for details of the options.

imm12 is any value in the range 0-4095.

Operation

The ADD instruction adds the value of Operand2 or imm12 to the value in Rn.

The ADC instruction adds the values in Rn and Operand2, together with the carry flag.

The SUB instruction subtracts the value of Operand2 or imm12 from the value in Rn.

The SBC instruction subtracts the value of Operand2 from the value in Rn. If the carry flag is clear, the result is reduced by one.

The RSB instruction subtracts the value in Rn from the value of Operand2. This is useful because of the wide range of options for Operand2.

Use ADC and SBC to synthesize multiword arithmetic, see [Multiword arithmetic examples](#) on.

See also “[ADR](#)” .

Note: ADDW is equivalent to the ADD syntax that uses the imm12 operand. SUBW is equivalent to the SUB syntax that uses the imm12 operand.

Restrictions

In these instructions:

- Operand2 must not be SP and must not be PC
- Rd can be SP only in ADD and SUB, and only with the additional restrictions:
  - Rn must also be SP
  - Any shift in Operand2 must be limited to a maximum of 3 bits using LSL
- Rn can be SP only in ADD and SUB
- Rd can be PC only in the ADD{cond} PC, PC, Rm instruction where:
  - The user must not specify the S suffix
  - Rm must not be PC and must not be SP
  - If the instruction is conditional, it must be the last instruction in the IT block

- With the exception of the ADD{cond} PC, PC, Rm instruction,  $Rn$  can be PC only in ADD and SUB, and only with the additional restrictions:
  - The user must not specify the S suffix
  - The second operand must be a constant in the range 0 to 4095.
  - Note: When using the PC for an addition or a subtraction, bits[1:0] of the PC are rounded to 0b00 before performing the calculation, making the base address for the calculation word-aligned.
  - Note: To generate the address of an instruction, the constant based on the value of the PC must be adjusted. ARM recommends to use the ADR instruction instead of ADD or SUB with  $Rn$  equal to the PC, because the assembler automatically calculates the correct constant for the ADR instruction.

When  $Rd$  is PC in the ADD{cond} PC, PC, Rm instruction:

- Bit[0] of the value written to the PC is ignored
- A branch occurs to the address created by forcing bit[0] of that value to 0.

#### Condition Flags

If S is specified, these instructions update the N, Z, C and V flags according to the result.

#### Examples

```

ADD      R2, R1, R3          ; Sets the flags on the result
SUBS    R8, R6, #240         ; Subtracts contents of R4 from 1280
RSB     R4, R4, #1280        ; Only executed if C flag set and Z
ADCHI   R11, R0, R3         ; flag clear.

```

#### Multiword Arithmetic Examples

The example below shows two instructions that add a 64-bit integer contained in R2 and R3 to another 64-bit integer contained in R0 and R1, and place the result in R4 and R5.

#### 64-bit Addition Example

```

ADDS    R4, R0, R2          ; add the least significant words
ADC     R5, R1, R3          ; add the most significant words with carry

```

Multiword values do not have to use consecutive registers. The example below shows instructions that subtract a 96-bit integer contained in R9, R1, and R11 from another contained in R6, R2, and R8. The example stores the result in R6, R9, and R2.

#### 96-bit Subtraction Example

```

SUBS   R6, R6, R9          ; subtract the least significant words
SBCS   R9, R2, R1          ; subtract the middle words with carry
SBC    R2, R8, R11         ; subtract the most significant words with carry

```

## 12.6.5.2 AND, ORR, EOR, BIC, and ORN

Logical AND, OR, Exclusive OR, Bit Clear, and OR NOT.

### Syntax

*op{S}{cond} {Rd, } Rn, Operand2*

where:

*op* is one of:

AND logical AND.

ORR logical OR, or bit set.

EOR logical Exclusive OR.

BIC logical AND NOT, or bit clear.

ORN logical OR NOT.

*S* is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see “[Conditional Execution](#)” .

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn* is the register holding the first operand.

*Operand2* is a flexible second operand. See “[Flexible Second Operand](#)” for details of the options

### Operation

The AND, EOR, and ORR instructions perform bitwise AND, Exclusive OR, and OR operations on the values in *Rn* and *Operand2*.

The BIC instruction performs an AND operation on the bits in *Rn* with the complements of the corresponding bits in the value of *Operand2*.

The ORN instruction performs an OR operation on the bits in *Rn* with the complements of the corresponding bits in the value of *Operand2*.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

If *S* is specified, these instructions:

- Update the N and Z flags according to the result
- Can update the C flag during the calculation of *Operand2*, see “[Flexible Second Operand](#)”
- Do not affect the V flag.

### Examples

AND	R9, R2, #0xFF00
ORREQ	R2, R0, R5
ANDS	R9, R8, #0x19
EORS	R7, R11, #0x18181818
BIC	R0, R1, #0xab
ORN	R7, R11, R14, ROR #4
ORNS	R7, R11, R14, ASR #32

### 12.6.5.3 ASR, LSL, LSR, ROR, and RRX

Arithmetic Shift Right, Logical Shift Left, Logical Shift Right, Rotate Right, and Rotate Right with Extend.

#### Syntax

```
op{S}{cond} Rd, Rm, Rs  
op{S}{cond} Rd, Rm, #n  
RRX{S}{cond} Rd, Rm
```

where:

op is one of:

ASR Arithmetic Shift Right.

LSL Logical Shift Left.

LSR Logical Shift Right.

ROR Rotate Right.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “[Conditional Execution](#)” .

Rd is the destination register.

Rm is the register holding the value to be shifted.

Rs is the register holding the shift length to apply to the value in Rm. Only the least significant byte is used and can be in the range 0 to 255.

n is the shift length. The range of shift length depends on the instruction:

ASR shift length from 1 to 32

LSL shift length from 0 to 31

LSR shift length from 1 to 32

ROR shift length from 0 to 31

MOVS Rd, Rm is the preferred syntax for LSLS Rd, Rm, #0.

#### Operation

ASR, LSL, LSR, and ROR move the bits in the register Rm to the left or right by the number of places specified by constant n or register Rs.

RRX moves the bits in register Rm to the right by 1.

In all these instructions, the result is written to Rd, but the value in register Rm remains unchanged. For details on what result is generated by the different instructions, see “[Shift Operations](#)” .

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

If S is specified:

- These instructions update the N and Z flags according to the result
- The C flag is updated to the last bit shifted out, except when the shift length is 0, see “[Shift Operations](#)” .

#### Examples

```
ASR    R7, R8, #9 ; Arithmetic shift right by 9 bits  
SLS    R1, R2, #3 ; Logical shift left by 3 bits with flag update  
LSR    R4, R5, #6 ; Logical shift right by 6 bits  
ROR    R4, R5, R6 ; Rotate right by the value in the bottom byte of R6  
RRX    R4, R5      ; Rotate right with extend.
```

#### 12.6.5.4 CLZ

Count Leading Zeros.

Syntax

CLZ{cond} Rd, Rm

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

Rm is the operand register.

Operation

The CLZ instruction counts the number of leading zeros in the value in Rm and returns the result in Rd. The result value is 32 if no bits are set and zero if bit[31] is set.

Restrictions

Do not use SP and do not use PC.

Condition Flags

This instruction does not change the flags.

Examples

CLZ	R4, R9
CLZNE	R2, R3

### 12.6.5.5 CMP and CMN

Compare and Compare Negative.

Syntax

```
CMP{cond} Rn, Operand2  
CMN{cond} Rn, Operand2
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See “[Flexible Second Operand](#)” for details of the options

Operation

These instructions compare the value in a register with *Operand2*. They update the condition flags on the result, but do not write the result to a register.

The CMP instruction subtracts the value of *Operand2* from the value in *Rn*. This is the same as a SUBS instruction, except that the result is discarded.

The CMN instruction adds the value of *Operand2* to the value in *Rn*. This is the same as an ADDS instruction, except that the result is discarded.

Restrictions

In these instructions:

- Do not use PC
- *Operand2* must not be SP.

Condition Flags

These instructions update the N, Z, C and V flags according to the result.

Examples

```
CMP      R2, R9  
CMN      R0, #6400  
CMPGT    SP, R7, LSL #2
```

### 12.6.5.6 MOV and MVN

Move and Move NOT.

Syntax

```
MOV{S}{cond} Rd, Operand2  
MOV{cond} Rd, #imm16  
MVN{S}{cond} Rd, Operand2
```

where:

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “[Conditional Execution](#)” .

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

Operand2 is a flexible second operand. See “[Flexible Second Operand](#)” for details of the options

imm16 is any value in the range 0-65535.

Operation

The MOV instruction copies the value of *Operand2* into *Rd*.

When *Operand2* in a MOV instruction is a register with a shift other than LSL #0, the preferred syntax is the corresponding shift instruction:

- ASR{S}{cond} Rd, Rm, #n is the preferred syntax for MOV{S}{cond} Rd, Rm, ASR #n
- LSL{S}{cond} Rd, Rm, #n is the preferred syntax for MOV{S}{cond} Rd, Rm, LSL #n if n != 0
- LSR{S}{cond} Rd, Rm, #n is the preferred syntax for MOV{S}{cond} Rd, Rm, LSR #n
- ROR{S}{cond} Rd, Rm, #n is the preferred syntax for MOV{S}{cond} Rd, Rm, ROR #n
- RRX{S}{cond} Rd, Rm is the preferred syntax for MOV{S}{cond} Rd, Rm, RRX.

Also, the MOV instruction permits additional forms of *Operand2* as synonyms for shift instructions:

- MOV{S}{cond} Rd, Rm, ASR Rs is a synonym for ASR{S}{cond} Rd, Rm, Rs
- MOV{S}{cond} Rd, Rm, LSL Rs is a synonym for LSL{S}{cond} Rd, Rm, Rs
- MOV{S}{cond} Rd, Rm, LSR Rs is a synonym for LSR{S}{cond} Rd, Rm, Rs
- MOV{S}{cond} Rd, Rm, ROR Rs is a synonym for ROR{S}{cond} Rd, Rm, Rs

See "[ASR, LSL, LSR, ROR, and RRX](#)".

The MVN instruction takes the value of *Operand2*, performs a bitwise logical NOT operation on the value, and places the result into *Rd*.

The MOVW instruction provides the same function as MOV, but is restricted to using the *imm16* operand.

#### Restrictions

SP and PC only can be used in the MOV instruction, with the following restrictions:

- The second operand must be a register without shift
- The S suffix must not be specified.

When *Rd* is PC in a MOV instruction:

- Bit[0] of the value written to the PC is ignored
- A branch occurs to the address created by forcing bit[0] of that value to 0.

Though it is possible to use MOV as a branch instruction, ARM strongly recommends the use of a BX or BLX instruction to branch for software portability to the ARM instruction set.

#### Condition Flags

If S is specified, these instructions:

- Update the N and Z flags according to the result
- Can update the C flag during the calculation of *Operand2*, see "[Flexible Second Operand](#)"
- Do not affect the V flag.

#### Examples

```
    MOVS  R11, #0x000B          ; Write value of 0x000B to
    R11, flags get updated
        MOV   R1, #0xFA05          ; Write value of 0xFA05 to
    R1, flags are not updated
        MOVS  R10, R12           ; Write value in R12 to R10,
    flags get updated
        MOV   R3, #23             ; Write value of 23 to R3
        MOV   R8, SP              ; Write value of stack pointer to R8
        MVNS  R2, #0xF             ; Write value of 0xFFFFFFFF (bitwise inverse of 0xF)
                                ; to the R2 and update flags.
```

### 12.6.5.7 MOVT

Move Top.

Syntax

```
MOVT{cond} Rd, #imm16
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

imm16 is a 16-bit immediate constant.

Operation

MOVT writes a 16-bit immediate value, *imm16*, to the top halfword, *Rd*[31:16], of its destination register. The write does not affect *Rd*[15:0].

The MOV, MOVT instruction pair enables to generate any 32-bit constant.

Restrictions

*Rd* must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

```
MOVT    R3, #0xF123 ; Write 0xF123 to upper halfword of R3, lower halfword  
           ; and APSR are unchanged.
```

### 12.6.5.8 REV, REV16, REVSH, and RBIT

Reverse bytes and Reverse bits.

Syntax

```
op{cond} Rd, Rn
```

where:

op is any of:

REV Reverse byte order in a word.

REV16 Reverse byte order in each halfword independently.

REVSH Reverse byte order in the bottom halfword, and sign extend to 32 bits.

RBIT Reverse the bit order in a 32-bit word.

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

Rn is the register holding the operand.

Operation

Use these instructions to change endianness of data:

REV converts either:

- 32-bit big-endian data into little-endian data
- 32-bit little-endian data into big-endian data.

REV16 converts either:

- 16-bit big-endian data into little-endian data
- 16-bit little-endian data into big-endian data.

REVSH converts either:

- 16-bit signed big-endian data into 32-bit signed little-endian data

- 16-bit signed little-endian data into 32-bit signed big-endian data.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
REV    R3, R7; Reverse byte order of value in R7 and write it to R3
REV16   R0, R0; Reverse byte order of each 16-bit halfword in R0
REVSH   R0, R5; Reverse Signed Halfword
REVHS   R3, R7; Reverse with Higher or Same condition
RBIT    R7, R8; Reverse bit order of value in R8 and write the result to R7.
```

### 12.6.5.9 SADD16 and SADD8

Signed Add 16 and Signed Add 8

Syntax

```
op{cond}{Rd, } Rn, Rm
```

where:

op        is any of:

SADD16 Performs two 16-bit signed integer additions.

SADD8 Performs four 8-bit signed integer additions.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Rd        is the destination register.

Rn        is the first register holding the operand.

Rm        is the second register holding the operand.

Operation

Use these instructions to perform a halfword or byte add in parallel:

The SADD16 instruction:

1. Adds each halfword from the first operand to the corresponding halfword of the second operand.
2. Writes the result in the corresponding halfwords of the destination register.

The SADD8 instruction:

1. Adds each byte of the first operand to the corresponding byte of the second operand.

Writes the result in the corresponding bytes of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SADD16 R1, R0      ; Adds the halfwords in R0 to the corresponding
                     ; halfwords of R1 and writes to corresponding halfword
                     ; of R1.
SADD8  R4, R0, R5 ; Adds bytes of R0 to the corresponding byte in R5 and
                     ; writes to the corresponding byte in R4.
```

## 12.6.5.10 SHADD16 and SHADD8

Signed Halving Add 16 and Signed Halving Add 8

Syntax

*op{cond}{Rd, } Rn, Rm*

where:

*op* is any of:

SHADD16 Signed Halving Add 16.

SHADD8 Signed Halving Add 8.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn* is the first operand register.

*Rm* is the second operand register.

Operation

Use these instructions to add 16-bit and 8-bit data and then to halve the result before writing the result to the destination register:

The SHADD16 instruction:

1. Adds each halfword from the first operand to the corresponding halfword of the second operand.
2. Shuffles the result by one bit to the right, halving the data.
3. Writes the halfword results in the destination register.

The SHADDB8 instruction:

1. Adds each byte of the first operand to the corresponding byte of the second operand.
2. Shuffles the result by one bit to the right, halving the data.
3. Writes the byte results in the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SHADD16 R1, R0      ; Adds halfwords in R0 to corresponding halfword of R1
                      ; and writes halved result to corresponding halfword in
                      ; R1
SHADD8  R4, R0, R5 ; Adds bytes of R0 to corresponding byte in R5 and
                      ; writes halved result to corresponding byte in R4.
```

### 12.6.5.11 SHASX and SHSAX

Signed Halving Add and Subtract with Exchange and Signed Halving Subtract and Add with Exchange.

#### Syntax

*op{cond} {Rd}, Rn, Rm*

where:

*op* is any of:

SHASX Add and Subtract with Exchange and Halving.

SHSAX Subtract and Add with Exchange and Halving.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn, Rm* are registers holding the first and second operands.

#### Operation

The SHASX instruction:

1. Adds the top halfword of the first operand with the bottom halfword of the second operand.
2. Writes the halfword result of the addition to the top halfword of the destination register, shifted by one bit to the right causing a divide by two, or halving.
3. Subtracts the top halfword of the second operand from the bottom highword of the first operand.
4. Writes the halfword result of the division in the bottom halfword of the destination register, shifted by one bit to the right causing a divide by two, or halving.

The SHSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Writes the halfword result of the subtraction to the bottom halfword of the destination register, shifted by one bit to the right causing a divide by two, or halving.
3. Adds the bottom halfword of the first operand with the top halfword of the second operand.
4. Writes the halfword result of the addition in the top halfword of the destination register, shifted by one bit to the right causing a divide by two, or halving.

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

These instructions do not affect the condition code flags.

#### Examples

```
SHASX  R7, R4, R2 ; Adds top halfword of R4 to bottom halfword of R2  
          ; and writes halved result to top halfword of R7  
          ; Subtracts top halfword of R2 from bottom halfword of  
          ; R4 and writes halved result to bottom halfword of R7  
SHSAX  R0, R3, R5 ; Subtracts bottom halfword of R5 from top halfword  
          ; of R3 and writes halved result to top halfword of R0  
          ; Adds top halfword of R5 to bottom halfword of R3 and  
          ; writes halved result to bottom halfword of R0.
```

## 12.6.5.12 SHSUB16 and SHSUB8

Signed Halving Subtract 16 and Signed Halving Subtract 8

Syntax

*op{cond}{Rd, } Rn, Rm*

where:

*op* is any of:

SHSUB16 Signed Halving Subtract 16.

SHSUB8 Signed Halving Subtract 8.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn* is the first operand register.

*Rm* is the second operand register.

Operation

Use these instructions to add 16-bit and 8-bit data and then to halve the result before writing the result to the destination register:

The SHSUB16 instruction:

1. Subtracts each halfword of the second operand from the corresponding halfwords of the first operand.
2. Shuffles the result by one bit to the right, halving the data.
3. Writes the halved halfword results in the destination register.

The SHSUBB8 instruction:

1. Subtracts each byte of the second operand from the corresponding byte of the first operand,
2. Shuffles the result by one bit to the right, halving the data,
3. Writes the corresponding signed byte results in the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SHSUB16 R1, R0      ; Subtracts halfwords in R0 from corresponding halfword  
                      ; of R1 and writes to corresponding halfword of R1  
SHSUB8  R4, R0, R5 ; Subtracts bytes of R0 from corresponding byte in R5,  
                      ; and writes to corresponding byte in R4.
```

### 12.6.5.13 SSUB16 and SSUB8

Signed Subtract 16 and Signed Subtract 8

Syntax

*op{cond}{Rd, } Rn, Rm*

where:

*op* is any of:

SSUB16 Performs two 16-bit signed integer subtractions.

SSUB8 Performs four 8-bit signed integer subtractions.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn* is the first operand register.

*Rm* is the second operand register.

Operation

Use these instructions to change endianness of data:

The SSUB16 instruction:

1. Subtracts each halfword from the second operand from the corresponding halfword of the first operand
2. Writes the difference result of two signed halfwords in the corresponding halfword of the destination register.

The SSUB8 instruction:

1. Subtracts each byte of the second operand from the corresponding byte of the first operand
2. Writes the difference result of four signed bytes in the corresponding byte of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SSUB16 R1, R0      ; Subtracts halfwords in R0 from corresponding halfword
                     ; of R1 and writes to corresponding halfword of R1
SSUB8  R4, R0, R5 ; Subtracts bytes of R5 from corresponding byte in
                     ; R0, and writes to corresponding byte of R4.
```

## 12.6.5.14 SASX and SSAX

Signed Add and Subtract with Exchange and Signed Subtract and Add with Exchange.

### Syntax

*op{cond} {Rd}, Rm, Rn*

where:

*op* is any of:

SASX Signed Add and Subtract with Exchange.

SSAX Signed Subtract and Add with Exchange.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn, Rm* are registers holding the first and second operands.

### Operation

The SASX instruction:

1. Adds the signed top halfword of the first operand with the signed bottom halfword of the second operand.
2. Writes the signed result of the addition to the top halfword of the destination register.
3. Subtracts the signed bottom halfword of the second operand from the top signed highword of the first operand.
4. Writes the signed result of the subtraction to the bottom halfword of the destination register.

The SSAX instruction:

1. Subtracts the signed bottom halfword of the second operand from the top signed highword of the first operand.
2. Writes the signed result of the subtraction to the bottom halfword of the destination register.
3. Adds the signed top halfword of the first operand with the signed bottom halfword of the second operand.
4. Writes the signed result of the addition to the top halfword of the destination register.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

These instructions do not affect the condition code flags.

### Examples

```
SASX R0, R4, R5 ; Adds top halfword of R4 to bottom halfword of R5 and  
; writes to top halfword of R0  
; Subtracts bottom halfword of R5 from top halfword of R4  
; and writes to bottom halfword of R0  
SSAX R7, R3, R2 ; Subtracts top halfword of R2 from bottom halfword of R3  
; and writes to bottom halfword of R7  
; Adds top halfword of R3 with bottom halfword of R2 and  
; writes to top halfword of R7.
```

## 12.6.5.15 TST and TEQ

Test bits and Test Equivalence.

### Syntax

```
TST{cond} Rn, Operand2  
TEQ{cond} Rn, Operand2
```

where

cond is an optional condition code, see “[Conditional Execution](#)” .

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See “[Flexible Second Operand](#)” for details of the options

### Operation

These instructions test the value in a register against *Operand2*. They update the condition flags based on the result, but do not write the result to a register.

The TST instruction performs a bitwise AND operation on the value in *Rn* and the value of *Operand2*. This is the same as the ANDS instruction, except that it discards the result.

To test whether a bit of *Rn* is 0 or 1, use the TST instruction with an *Operand2* constant that has that bit set to 1 and all other bits cleared to 0.

The TEQ instruction performs a bitwise Exclusive OR operation on the value in *Rn* and the value of *Operand2*. This is the same as the EORS instruction, except that it discards the result.

Use the TEQ instruction to test if two values are equal without affecting the V or C flags.

TEQ is also useful for testing the sign of a value. After the comparison, the N flag is the logical Exclusive OR of the sign bits of the two operands.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

These instructions:

- Update the N and Z flags according to the result
- Can update the C flag during the calculation of *Operand2*, see “[Flexible Second Operand](#)”
- Do not affect the V flag.

### Examples

```
TST      R0, #0x3F8 ; Perform bitwise AND of R0 value to 0x3F8,  
                      ; APSR is updated but result is discarded  
TEQEQQ R10, R9      ; Conditionally test if value in R10 is equal to  
                      ; value in R9, APSR is updated but result is discarded.
```

## 12.6.5.16 UADD16 and UADD8

Unsigned Add 16 and Unsigned Add 8

Syntax

*op{cond}{Rd, } Rn, Rm*

where:

op is any of:

UADD16 Performs two 16-bit unsigned integer additions.

UADD8 Performs four 8-bit unsigned integer additions.

cond is an optional condition code, see "[Conditional Execution](#)" .

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

Use these instructions to add 16- and 8-bit unsigned data:

The UADD16 instruction:

1. Adds each halfword from the first operand to the corresponding halfword of the second operand.
2. Writes the unsigned result in the corresponding halfwords of the destination register.

The UADD16 instruction:

1. Adds each byte of the first operand to the corresponding byte of the second operand.
2. Writes the unsigned result in the corresponding byte of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
UADD16 R1, R0      ; Adds halfwords in R0 to corresponding halfword of R1,  
                     ; writes to corresponding halfword of R1  
UADD8  R4, R0, R5   ; Adds bytes of R0 to corresponding byte in R5 and  
                     ; writes to corresponding byte in R4.
```

## 12.6.5.17 UASX and USAX

Add and Subtract with Exchange and Subtract and Add with Exchange.

### Syntax

*op{cond} {Rd}, Rn, Rm*

where:

*op* is one of:

UASX Add and Subtract with Exchange.

USAX Subtract and Add with Exchange.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn, Rm* are registers holding the first and second operands.

### Operation

The UASX instruction:

1. Subtracts the top halfword of the second operand from the bottom halfword of the first operand.
2. Writes the unsigned result from the subtraction to the bottom halfword of the destination register.
3. Adds the top halfword of the first operand with the bottom halfword of the second operand.
4. Writes the unsigned result of the addition to the top halfword of the destination register.

The USAX instruction:

1. Adds the bottom halfword of the first operand with the top halfword of the second operand.
2. Writes the unsigned result of the addition to the bottom halfword of the destination register.
3. Subtracts the bottom halfword of the second operand from the top halfword of the first operand.
4. Writes the unsigned result from the subtraction to the top halfword of the destination register.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

These instructions do not affect the condition code flags.

### Examples

```
UASX R0, R4, R5 ; Adds top halfword of R4 to bottom halfword of R5 and  
; writes to top halfword of R0  
; Subtracts bottom halfword of R5 from top halfword of R0  
; and writes to bottom halfword of R0  
USAX R7, R3, R2 ; Subtracts top halfword of R2 from bottom halfword of R3  
; and writes to bottom halfword of R7  
; Adds top halfword of R3 to bottom halfword of R2 and  
; writes to top halfword of R7.
```

## 12.6.5.18 UHADD16 and UHADD8

Unsigned Halving Add 16 and Unsigned Halving Add 8

Syntax

*op{cond}{Rd, } Rn, Rm*

where:

*op* is any of:

UHADD16 Unsigned Halving Add 16.

UHADD8 Unsigned Halving Add 8.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn* is the register holding the first operand.

*Rm* is the register holding the second operand.

Operation

Use these instructions to add 16- and 8-bit data and then to halve the result before writing the result to the destination register:

The UHADD16 instruction:

1. Adds each halfword from the first operand to the corresponding halfword of the second operand.
2. Shuffles the halfword result by one bit to the right, halving the data.
3. Writes the unsigned results to the corresponding halfword in the destination register.

The UHADD8 instruction:

1. Adds each byte of the first operand to the corresponding byte of the second operand.
2. Shuffles the byte result by one bit to the right, halving the data.
3. Writes the unsigned results in the corresponding byte in the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
UHADD16 R7, R3      ; Adds halfwords in R7 to corresponding halfword of R3  
                      ; and writes halved result to corresponding halfword  
                      ; in R7  
UHADD8  R4, R0, R5   ; Adds bytes of R0 to corresponding byte in R5 and  
                      ; writes halved result to corresponding byte in R4.
```

## 12.6.5.19 UHASX and UHSAX

Unsigned Halving Add and Subtract with Exchange and Unsigned Halving Subtract and Add with Exchange.

### Syntax

*op{cond} {Rd}, Rn, Rm*

where:

*op* is one of:

UHASX Add and Subtract with Exchange and Halving.

UHSAX Subtract and Add with Exchange and Halving.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn, Rm* are registers holding the first and second operands.

### Operation

The UHASX instruction:

1. Adds the top halfword of the first operand with the bottom halfword of the second operand.
2. Shifts the result by one bit to the right causing a divide by two, or halving.
3. Writes the halfword result of the addition to the top halfword of the destination register.
4. Subtracts the top halfword of the second operand from the bottom highword of the first operand.
5. Shifts the result by one bit to the right causing a divide by two, or halving.
6. Writes the halfword result of the division in the bottom halfword of the destination register.

The UHSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Shifts the result by one bit to the right causing a divide by two, or halving.
3. Writes the halfword result of the subtraction in the top halfword of the destination register.
4. Adds the bottom halfword of the first operand with the top halfword of the second operand.
5. Shifts the result by one bit to the right causing a divide by two, or halving.
6. Writes the halfword result of the addition to the bottom halfword of the destination register.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

These instructions do not affect the condition code flags.

### Examples

```
UHASX R7, R4, R2 ; Adds top halfword of R4 with bottom halfword of R2  
; and writes halved result to top halfword of R7  
; Subtracts top halfword of R2 from bottom halfword of  
; R7 and writes halved result to bottom halfword of R7  
UHSAX R0, R3, R5 ; Subtracts bottom halfword of R5 from top halfword of  
; R3 and writes halved result to top halfword of R0  
; Adds top halfword of R5 to bottom halfword of R3 and  
; writes halved result to bottom halfword of R0.
```

## 12.6.5.20 UHSUB16 and UHSUB8

Unsigned Halving Subtract 16 and Unsigned Halving Subtract 8

Syntax

*op{cond}{Rd, } Rn, Rm*

where:

*op* is any of:

UHSUB16 Performs two unsigned 16-bit integer additions, halves the results, and writes the results to the destination register.

UHSUB8 Performs four unsigned 8-bit integer additions, halves the results, and writes the results to the destination register.

*cond* is an optional condition code, see “[Conditional Execution](#)”.

*Rd* is the destination register.

*Rn* is the first register holding the operand.

*Rm* is the second register holding the operand.

Operation

Use these instructions to add 16-bit and 8-bit data and then to halve the result before writing the result to the destination register:

The UHSUB16 instruction:

1. Subtracts each halfword of the second operand from the corresponding halfword of the first operand.
2. Shuffles each halfword result to the right by one bit, halving the data.
3. Writes each unsigned halfword result to the corresponding halfwords in the destination register.

The UHSUB8 instruction:

1. Subtracts each byte of second operand from the corresponding byte of the first operand.
2. Shuffles each byte result by one bit to the right, halving the data.
3. Writes the unsigned byte results to the corresponding byte of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
UHSUB16 R1, R0      ; Subtracts halfwords in R0 from corresponding halfword of  
                      ; R1 and writes halved result to corresponding halfword in R1  
UHSUB8  R4, R0, R5   ; Subtracts bytes of R5 from corresponding byte in R0 and  
                      ; writes halved result to corresponding byte in R4.
```

### 12.6.5.21 SEL

Select Bytes. Selects each byte of its result from either its first operand or its second operand, according to the values of the GE flags.

#### Syntax

```
SEL{<c>}{<q>} {<Rd>, } <Rn>, <Rm>
```

where:

c, q are standard assembler syntax fields.

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

#### Operation

The SEL instruction:

1. Reads the value of each bit of APSR.GE.
2. Depending on the value of APSR.GE, assigns the destination register the value of either the first or second operand register.

#### Restrictions

None.

#### Condition Flags

These instructions do not change the flags.

#### Examples

```
SADD16 R0, R1, R2      ; Set GE bits based on result  
SEL     R0, R0, R3      ; Select bytes from R0 or R3, based on GE.
```

### 12.6.5.22 USAD8

Unsigned Sum of Absolute Differences

#### Syntax

```
USAD8{cond}{Rd, } Rn, Rm
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

#### Operation

The USAD8 instruction:

1. Subtracts each byte of the second operand register from the corresponding byte of the first operand register.
2. Adds the absolute values of the differences together.
3. Writes the result to the destination register.

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

These instructions do not change the flags.

#### Examples

```
USAD8 R1, R4, R0 ; Subtracts each byte in R0 from corresponding byte of R4  
                   ; adds the differences and writes to R1
```

```
USAD8 R0, R5      ; Subtracts bytes of R5 from corresponding byte in R0,  
; adds the differences and writes to R0.
```

### 12.6.5.23 USADA8

Unsigned Sum of Absolute Differences and Accumulate

Syntax

```
USADA8{cond}{Rd,} Rn, Rm, Ra
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Rd is the destination register.  
Rn is the first operand register.  
Rm is the second operand register.  
Ra is the register that contains the accumulation value.

Operation

The USADA8 instruction:

1. Subtracts each byte of the second operand register from the corresponding byte of the first operand register.
2. Adds the unsigned absolute differences together.
3. Adds the accumulation value to the sum of the absolute differences.
4. Writes the result to the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
USADA8 R1, R0, R6      ; Subtracts bytes in R0 from corresponding halfword of R1  
; adds differences, adds value of R6, writes to R1  
USADA8 R4, R0, R5, R2  ; Subtracts bytes of R5 from corresponding byte in R0  
; adds differences, adds value of R2 writes to R4.
```

### 12.6.5.24 USUB16 and USUB8

Unsigned Subtract 16 and Unsigned Subtract 8

Syntax

```
op{cond}{Rd,} Rn, Rm
```

where

- op is any of:  
USUB16 Unsigned Subtract 16.  
USUB8 Unsigned Subtract 8.  
cond is an optional condition code, see “[Conditional Execution](#)” .  
Rd is the destination register.  
Rn is the first operand register.  
Rm is the second operand register.

Operation

Use these instructions to subtract 16-bit and 8-bit data before writing the result to the destination register:

The USUB16 instruction:

1. Subtracts each halfword from the second operand register from the corresponding halfword of the first operand register.
2. Writes the unsigned result in the corresponding halfwords of the destination register.

The USUB8 instruction:

1. Subtracts each byte of the second operand register from the corresponding byte of the first operand register.
2. Writes the unsigned byte result in the corresponding byte of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
USUB16 R1, R0 ; Subtracts halfwords in R0 from corresponding halfword of R1  
; and writes to corresponding halfword in R1USUB8 R4, R0, R5  
; Subtracts bytes of R5 from corresponding byte in R0 and  
; writes to the corresponding byte in R4.
```

## 12.6.6 Multiply and Divide Instructions

The table below shows the multiply and divide instructions:

**Table 12-21. Multiply and Divide Instructions**

Mnemonic	Description
MLA	Multiply with Accumulate, 32-bit result
MLS	Multiply and Subtract, 32-bit result
MUL	Multiply, 32-bit result
SDIV	Signed Divide
SMLA[B,T]	Signed Multiply Accumulate (halfwords)
SMLAD, SMLADX	Signed Multiply Accumulate Dual
SMLAL	Signed Multiply with Accumulate (32x32+64), 64-bit result
SMLAL[B,T]	Signed Multiply Accumulate Long (halfwords)
SMLALD, SMLALDX	Signed Multiply Accumulate Long Dual
SMLAW[B T]	Signed Multiply Accumulate (word by halfword)
SMLSD	Signed Multiply Subtract Dual
SMLS LD	Signed Multiply Subtract Long Dual
SMMLA	Signed Most Significant Word Multiply Accumulate
SMMLS, SMMLSR	Signed Most Significant Word Multiply Subtract
SMUAD, SMUADX	Signed Dual Multiply Add
SMUL[B,T]	Signed Multiply (word by halfword)
SMMUL, SMMULR	Signed Most Significant Word Multiply
SMULL	Signed Multiply (32x32), 64-bit result
SMULWB, SMULWT	Signed Multiply (word by halfword)
SMUSD, SMUSDX	Signed Dual Multiply Subtract
UDIV	Unsigned Divide
UMAAL	Unsigned Multiply Accumulate Accumulate Long (32x32+32+32), 64-bit result
UMLAL	Unsigned Multiply with Accumulate (32x32+64), 64-bit result
UMULL	Unsigned Multiply (32x32), 64-bit result

## 12.6.6.1 MUL, MLA, and MLS

Multiply, Multiply with Accumulate, and Multiply with Subtract, using 32-bit operands, and producing a 32-bit result.

### Syntax

```
MUL{S}{cond} {Rd,} Rn, Rm ; Multiply  
MLA{cond} Rd, Rn, Rm, Ra ; Multiply with accumulate  
MLS{cond} Rd, Rn, Rm, Ra ; Multiply with subtract
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “[Conditional Execution](#)” .

Rd is the destination register. If Rd is omitted, the destination register is Rn.

Rn, Rm are registers holding the values to be multiplied.

Ra is a register holding the value to be added or subtracted from.

### Operation

The MUL instruction multiplies the values from Rn and Rm, and places the least significant 32 bits of the result in Rd.

The MLA instruction multiplies the values from Rn and Rm, adds the value from Ra, and places the least significant 32 bits of the result in Rd.

The MLS instruction multiplies the values from Rn and Rm, subtracts the product from the value from Ra, and places the least significant 32 bits of the result in Rd.

The results of these instructions do not depend on whether the operands are signed or unsigned.

### Restrictions

In these instructions, do not use SP and do not use PC.

If the S suffix is used with the MUL instruction:

- Rd, Rn, and Rm must all be in the range R0 to R7
- Rd must be the same as Rm
- The cond suffix must not be used.

### Condition Flags

If S is specified, the MUL instruction:

- Updates the N and Z flags according to the result
- Does not affect the C and V flags.

### Examples

```
MUL    R10, R2, R5      ; Multiply, R10 = R2 x R5  
MLA    R10, R2, R1, R5 ; Multiply with accumulate, R10 = (R2 x R1) + R5  
MULS   R0, R2, R2      ; Multiply with flag update, R0 = R2 x R2  
MULLT  R2, R3, R2      ; Conditionally multiply, R2 = R3 x R2  
MLS    R4, R5, R6, R7  ; Multiply with subtract, R4 = R7 - (R5 x R6)
```

## 12.6.6.2 UMULL, UMAAL, UMLAL

Unsigned Long Multiply, with optional Accumulate, using 32-bit operands and producing a 64-bit result.

### Syntax

*op{cond} RdLo, RdHi, Rn, Rm*

where:

*op* is one of:

UMULL Unsigned Long Multiply.

UMAAL Unsigned Long Multiply with Accumulate Accumulate.

UMLAL Unsigned Long Multiply, with Accumulate.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*RdHi, RdLo* are the destination registers. For UMAAL, UMLAL and UMLAL they also hold the accumulating value.

*Rn, Rm* are registers holding the first and second operands.

### Operation

These instructions interpret the values from *Rn* and *Rm* as unsigned 32-bit integers.

The UMULL instruction:

- Multiplies the two unsigned integers in the first and second operands.
- Writes the least significant 32 bits of the result in *RdLo*.
- Writes the most significant 32 bits of the result in *RdHi*.

The UMAAL instruction:

- Multiplies the two unsigned 32-bit integers in the first and second operands.
- Adds the unsigned 32-bit integer in *RdHi* to the 64-bit result of the multiplication.
- Adds the unsigned 32-bit integer in *RdLo* to the 64-bit result of the addition.
- Writes the top 32-bits of the result to *RdHi*.
- Writes the lower 32-bits of the result to *RdLo*.

The UMLAL instruction:

- Multiplies the two unsigned integers in the first and second operands.
- Adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*.
- Writes the result back to *RdHi* and *RdLo*.

### Restrictions

In these instructions:

- Do not use SP and do not use PC.
- *RdHi* and *RdLo* must be different registers.

### Condition Flags

These instructions do not affect the condition code flags.

### Examples

```
UMULL    R0, R4, R5, R6      ; Multiplies R5 and R6, writes the top 32 bits to R4  
          ; and the bottom 32 bits to R0  
UMAAL    R3, R6, R2, R7      ; Multiplies R2 and R7, adds R6, adds R3, writes the  
          ; top 32 bits to R6, and the bottom 32 bits to R3  
UMLAL    R2, R1, R3, R5      ; Multiplies R5 and R3, adds R1:R2, writes to R1:R2.
```

### 12.6.6.3 SMLA and SMLAW

Signed Multiply Accumulate (halfwords).

Syntax

```
op{XY}{cond} Rd, Rn, Rm  
op{Y}{cond} Rd, Rn, Rm, Ra
```

where:

op       is one of:

SMLA Signed Multiply Accumulate Long (halfwords).

X and Y specifies which half of the source registers *Rn* and *Rm* are used as the first and second multiply operand.

If X is *B*, then the bottom halfword, bits [15:0], of *Rn* is used.

If X is *T*, then the top halfword, bits [31:16], of *Rn* is used.

If Y is *B*, then the bottom halfword, bits [15:0], of *Rm* is used.

If Y is *T*, then the top halfword, bits [31:16], of *Rm* is used

SMLAW Signed Multiply Accumulate (word by halfword).

Y specifies which half of the source register *Rm* is used as the second multiply operand.

If Y is *T*, then the top halfword, bits [31:16] of *Rm* is used.

If Y is *B*, then the bottom halfword, bits [15:0] of *Rm* is used.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Rd        is the destination register. If *Rd* is omitted, the destination register is *Rn*.

Rn, Rm    are registers holding the values to be multiplied.

Ra        is a register holding the value to be added or subtracted from.

Operation

The SMALBB, SMLABT, SMLATB, SMLATT instructions:

- Multiplies the specified signed halfword, top or bottom, values from *Rn* and *Rm*.
- Adds the value in *Ra* to the resulting 32-bit product.
- Writes the result of the multiplication and addition in *Rd*.

The non-specified halfwords of the source registers are ignored.

The SMLAWB and SMLAWT instructions:

- Multiply the 32-bit signed values in *Rn* with:
  - The top signed halfword of *Rm*, *T* instruction suffix.
  - The bottom signed halfword of *Rm*, *B* instruction suffix.
- Add the 32-bit signed value in *Ra* to the top 32 bits of the 48-bit product
- Writes the result of the multiplication and addition in *Rd*.

The bottom 16 bits of the 48-bit product are ignored.

If overflow occurs during the addition of the accumulate value, the instruction sets the Q flag in the APSR. No overflow can occur during the multiplication.

Restrictions

In these instructions, do not use SP and do not use PC.

Condition Flags

If an overflow is detected, the Q flag is set.

Examples

```

SMLABB R5, R6, R4, R1 ; Multiplies bottom halfwords of R6 and R4, adds
; R1 and writes to R5
SMLATB R5, R6, R4, R1 ; Multiplies top halfword of R6 with bottom halfword
; of R4, adds R1 and writes to R5
SMLATT R5, R6, R4, R1 ; Multiplies top halfwords of R6 and R4, adds
; R1 and writes the sum to R5
SMLABT R5, R6, R4, R1 ; Multiplies bottom halfword of R6 with top halfword
; of R4, adds R1 and writes to R5
SMLABT R4, R3, R2 ; Multiplies bottom halfword of R4 with top halfword of
; R3, adds R2 and writes to R4
SMLAWB R10, R2, R5, R3 ; Multiplies R2 with bottom halfword of R5, adds
; R3 to the result and writes top 32-bits to R10
SMLAWT R10, R2, R1, R5 ; Multiplies R2 with top halfword of R1, adds R5
; and writes top 32-bits to R10.

```

#### 12.6.6.4 SMLAD

Signed Multiply Accumulate Long Dual

Syntax

```
op{X}{cond} Rd, Rn, Rm, Ra ;
```

where:

op        is one of:

SMLAD Signed Multiply Accumulate Dual.

SMLADX Signed Multiply Accumulate Dual Reverse.

X specifies which halfword of the source register *Rn* is used as the multiply operand.

If X is omitted, the multiplications are bottom × bottom and top × top.

If X is present, the multiplications are bottom × top and top × bottom.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Rd        is the destination register.

Rn        is the first operand register holding the values to be multiplied.

Rm        the second operand register.

Ra        is the accumulate value.

Operation

The SMLAD and SMLADX instructions regard the two operands as four halfword 16-bit values. The SMLAD and SMLADX instructions:

- If X is not present, multiply the top signed halfword value in *Rn* with the top signed halfword of *Rm* and the bottom signed halfword values in *Rn* with the bottom signed halfword of *Rm*.
- Or if X is present, multiply the top signed halfword value in *Rn* with the bottom signed halfword of *Rm* and the bottom signed halfword values in *Rn* with the top signed halfword of *Rm*.
- Add both multiplication results to the signed 32-bit value in *Ra*.
- Writes the 32-bit signed result of the multiplication and addition to *Rd*.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```

SMLAD   R10, R2, R1, R5 ; Multiplies two halfword values in R2 with
; corresponding halfwords in R1, adds R5 and

```

```

; writes to R10
SMLALDX R0, R2, R4, R6 ; Multiplies top halfword of R2 with bottom
; halfword of R4, multiplies bottom halfword of R2
; with top halfword of R4, adds R6 and writes to
; R0.

```

#### 12.6.6.5 SMLAL and SMLALD

Signed Multiply Accumulate Long, Signed Multiply Accumulate Long (halfwords) and Signed Multiply Accumulate Long Dual.

##### Syntax

```

op{cond} RdLo, RdHi, Rn, Rm
op{XY}{cond} RdLo, RdHi, Rn, Rm
op{X}{cond} RdLo, RdHi, Rn, Rm

```

where:

op        is one of:

MLAL Signed Multiply Accumulate Long.

SMLAL Signed Multiply Accumulate Long (halfwords, X and Y).

X and Y specify which halfword of the source registers *Rn* and *Rm* are used as the first and second multiply operand:

If X is B, then the bottom halfword, bits [15:0], of *Rn* is used.

If X is T, then the top halfword, bits [31:16], of *Rn* is used.

If Y is B, then the bottom halfword, bits [15:0], of *Rm* is used.

If Y is T, then the top halfword, bits [31:16], of *Rm* is used.

SMLALD Signed Multiply Accumulate Long Dual.

SMLALDX Signed Multiply Accumulate Long Dual Reversed.

If the X is omitted, the multiplications are bottom × bottom and top × top.

If X is present, the multiplications are bottom × top and top × bottom.

cond        is an optional condition code, see “[Conditional Execution](#)” .

RdHi, RdLo    are the destination registers.

*RdLo* is the lower 32 bits and *RdHi* is the upper 32 bits of the 64-bit integer.

For SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD and SMLALDX, they also hold the accumulating value.

Rn, Rm        are registers holding the first and second operands.

##### Operation

The SMLAL instruction:

- Multiplies the two's complement signed word values from *Rn* and *Rm*.
- Adds the 64-bit value in *RdLo* and *RdHi* to the resulting 64-bit product.
- Writes the 64-bit result of the multiplication and addition in *RdLo* and *RdHi*.

The SMLALBB, SMLALBT, SMLALTB and SMLALTT instructions:

- Multiplies the specified signed halfword, Top or Bottom, values from *Rn* and *Rm*.
- Adds the resulting sign-extended 32-bit product to the 64-bit value in *RdLo* and *RdHi*.
- Writes the 64-bit result of the multiplication and addition in *RdLo* and *RdHi*.

The non-specified halfwords of the source registers are ignored.

The SMLALD and SMLALDX instructions interpret the values from *Rn* and *Rm* as four halfword two's complement signed 16-bit integers. These instructions:

- If  $X$  is not present, multiply the top signed halfword value of  $Rn$  with the top signed halfword of  $Rm$  and the bottom signed halfword values of  $Rn$  with the bottom signed halfword of  $Rm$ .
- Or if  $X$  is present, multiply the top signed halfword value of  $Rn$  with the bottom signed halfword of  $Rm$  and the bottom signed halfword values of  $Rn$  with the top signed halfword of  $Rm$ .
- Add the two multiplication results to the signed 64-bit value in  $RdLo$  and  $RdHi$  to create the resulting 64-bit product.
- Write the 64-bit product in  $RdLo$  and  $RdHi$ .

#### Restrictions

In these instructions:

- Do not use SP and do not use PC.
- $RdHi$  and  $RdLo$  must be different registers.

#### Condition Flags

These instructions do not affect the condition code flags.

#### Examples

```

SMLAL    R4, R5, R3, R8 ; Multiplies R3 and R8, adds R5:R4 and writes to
; R5:R4
SMLALBT  R2, R1, R6, R7 ; Multiplies bottom halfword of R6 with top
; halfword of R7, sign extends to 32-bit, adds
; R1:R2 and writes to R1:R2
SMLALTB  R2, R1, R6, R7 ; Multiplies top halfword of R6 with bottom
; halfword of R7, sign extends to 32-bit, adds R1:R2
; and writes to R1:R2
SMLALD   R6, R8, R5, R1 ; Multiplies top halfwords in R5 and R1 and bottom
; halfwords of R5 and R1, adds R8:R6 and writes to
; R8:R6
SMLALDX  R6, R8, R5, R1 ; Multiplies top halfword in R5 with bottom
; halfword of R1, and bottom halfword of R5 with
; top halfword of R1, adds R8:R6 and writes to
; R8:R6.

```

#### 12.6.6.6 SMLSD and SMLSLD

Signed Multiply Subtract Dual and Signed Multiply Subtract Long Dual

##### Syntax

$op\{X\}\{cond\} Rd, Rn, Rm, Ra$

where:

op      is one of:

SMLSD Signed Multiply Subtract Dual.

SMLSDX Signed Multiply Subtract Dual Reversed.

SMLSLD Signed Multiply Subtract Long Dual.

SMLSLDX Signed Multiply Subtract Long Dual Reversed.

SMLAW Signed Multiply Accumulate (word by halfword).

If  $X$  is present, the multiplications are bottom  $\times$  top and top  $\times$  bottom.

If the  $X$  is omitted, the multiplications are bottom  $\times$  bottom and top  $\times$  top.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Rd      is the destination register.

Rn, Rm      are registers holding the first and second operands.

Ra      is the register holding the accumulate value.

Operation

The SMLSD instruction interprets the values from the first and second operands as four signed halfwords. This instruction:

- Optionally rotates the halfwords of the second operand.
- Performs two signed  $16 \times 16$ -bit halfword multiplications.
- Subtracts the result of the upper halfword multiplication from the result of the lower halfword multiplication.
- Adds the signed accumulate value to the result of the subtraction.
- Writes the result of the addition to the destination register.

The SMLS LD instruction interprets the values from  $Rn$  and  $Rm$  as four signed halfwords.

This instruction:

- Optionally rotates the halfwords of the second operand.
- Performs two signed  $16 \times 16$ -bit halfword multiplications.
- Subtracts the result of the upper halfword multiplication from the result of the lower halfword multiplication.
- Adds the 64-bit value in  $RdHi$  and  $RdLo$  to the result of the subtraction.
- Writes the 64-bit result of the addition to the  $RdHi$  and  $RdLo$ .

Restrictions

In these instructions:

- Do not use SP and do not use PC.

Condition Flags

This instruction sets the Q flag if the accumulate operation overflows. Overflow cannot occur during the multiplications or subtraction.

For the Thumb instruction set, these instructions do not affect the condition code flags.

Examples

```
SMLSD    R0, R4, R5, R6 ; Multiplies bottom halfword of R4 with bottom  
; halfword of R5, multiplies top halfword of R4  
; with top halfword of R5, subtracts second from  
; first, adds R6, writes to R0  
SMLSDX   R1, R3, R2, R0 ; Multiplies bottom halfword of R3 with top  
; halfword of R2, multiplies top halfword of R3  
; with bottom halfword of R2, subtracts second from  
; first, adds R0, writes to R1  
SMLS LD  R3, R6, R2, R7 ; Multiplies bottom halfword of R6 with bottom  
; halfword of R2, multiplies top halfword of R6  
; with top halfword of R2, subtracts second from  
; first, adds R6:R3, writes to R6:R3  
SMLS LDX R3, R6, R2, R7 ; Multiplies bottom halfword of R6 with top  
; halfword of R2, multiplies top halfword of R6  
; with bottom halfword of R2, subtracts second from  
; first, adds R6:R3, writes to R6:R3.
```

## 12.6.6.7 SMMLA and SMMLS

Signed Most Significant Word Multiply Accumulate and Signed Most Significant Word Multiply Subtract

Syntax

op{R}{cond} Rd, Rn, Rm, Ra

where:

op      is one of:

SMMLA Signed Most Significant Word Multiply Accumulate.

SMMLS Signed Most Significant Word Multiply Subtract.

If the X is omitted, the multiplications are bottom × bottom and top × top.

R      is a rounding error flag. If R is specified, the result is rounded instead of being truncated. In this case the constant 0x80000000 is added to the product before the high word is extracted.

cond    is an optional condition code, see “[Conditional Execution](#)” .

Rd      is the destination register.

Rn, Rm    are registers holding the first and second multiply operands.

Ra      is the register holding the accumulate value.

Operation

The SMMLA instruction interprets the values from Rn and Rm as signed 32-bit words.

The SMMLA instruction:

- Multiplies the values in Rn and Rm.
- Optionally rounds the result by adding 0x80000000.
- Extracts the most significant 32 bits of the result.
- Adds the value of Ra to the signed extracted value.
- Writes the result of the addition in Rd.

The SMMLS instruction interprets the values from Rn and Rm as signed 32-bit words.

The SMMLS instruction:

- Multiplies the values in Rn and Rm.
- Optionally rounds the result by adding 0x80000000.
- Extracts the most significant 32 bits of the result.
- Subtracts the extracted value of the result from the value in Ra.
- Writes the result of the subtraction in Rd.

Restrictions

In these instructions:

- Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
SMMLA R0, R4, R5, R6 ; Multiplies R4 and R5, extracts top 32 bits, adds  
; R6, truncates and writes to R0  
SMMLAR R6, R2, R1, R4 ; Multiplies R2 and R1, extracts top 32 bits, adds  
; R4, rounds and writes to R6  
SMMLSR R3, R6, R2, R7 ; Multiplies R6 and R2, extracts top 32 bits,  
; subtracts R7, rounds and writes to R3
```

```
SMMLS R4, R5, R3, R8 ; Multiplies R5 and R3, extracts top 32 bits,  
; subtracts R8, truncates and writes to R4.
```

### 12.6.6.8 SMMUL

Signed Most Significant Word Multiply

Syntax

```
op{R}{cond} Rd, Rn, Rm
```

where:

op       is one of:

SMMUL Signed Most Significant Word Multiply.

R       is a rounding error flag. If R is specified, the result is rounded instead of being truncated. In this case the constant 0x80000000 is added to the product before the high word is extracted.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Rd       is the destination register.

Rn, Rm     are registers holding the first and second operands.

Operation

The SMMUL instruction interprets the values from Rn and Rm as two's complement 32-bit signed integers. The SMMUL instruction:

- Multiplies the values from Rn and Rm.
- Optionally rounds the result, otherwise truncates the result.
- Writes the most significant signed 32 bits of the result in Rd.

Restrictions

In this instruction:

- do not use SP and do not use PC.

Condition Flags

This instruction does not affect the condition code flags.

Examples

```
SMULL R0, R4, R5 ; Multiplies R4 and R5, truncates top 32 bits  
; and writes to R0  
SMULLR R6, R2 ; Multiplies R6 and R2, rounds the top 32 bits  
; and writes to R6.
```

## 12.6.6.9 SMUAD and SMUSD

Signed Dual Multiply Add and Signed Dual Multiply Subtract

Syntax

*op{X}{cond} Rd, Rn, Rm*

where:

*op* is one of:

SMUAD Signed Dual Multiply Add.

SMUADX Signed Dual Multiply Add Reversed.

SMUSD Signed Dual Multiply Subtract.

SMUSDX Signed Dual Multiply Subtract Reversed.

If *X* is present, the multiplications are bottom  $\times$  top and top  $\times$  bottom.

If the *X* is omitted, the multiplications are bottom  $\times$  bottom and top  $\times$  top.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn, Rm* are registers holding the first and second operands.

Operation

The SMUAD instruction interprets the values from the first and second operands as two signed halfwords in each operand. This instruction:

- Optionally rotates the halfwords of the second operand.
- Performs two signed  $16 \times 16$ -bit multiplications.
- Adds the two multiplication results together.
- Writes the result of the addition to the destination register.

The SMUSD instruction interprets the values from the first and second operands as two's complement signed integers.

This instruction:

- Optionally rotates the halfwords of the second operand.
- Performs two signed  $16 \times 16$ -bit multiplications.
- Subtracts the result of the top halfword multiplication from the result of the bottom halfword multiplication.
- Writes the result of the subtraction to the destination register.

Restrictions

In these instructions:

- Do not use SP and do not use PC.

Condition Flags

Sets the Q flag if the addition overflows. The multiplications cannot overflow.

Examples

```
SMUAD    R0, R4, R5 ; Multiplies bottom halfword of R4 with the bottom
                      ; halfword of R5, adds multiplication of top halfword
                      ; of R4 with top halfword of R5, writes to R0
SMUADX   R3, R7, R4 ; Multiplies bottom halfword of R7 with top halfword
                      ; of R4, adds multiplication of top halfword of R7
                      ; with bottom halfword of R4, writes to R3
SMUSD    R3, R6, R2 ; Multiplies bottom halfword of R4 with bottom halfword
                      ; of R6, subtracts multiplication of top halfword of R6
                      ; with top halfword of R3, writes to R3
SMUSDX   R4, R5, R3 ; Multiplies bottom halfword of R5 with top halfword of
                      ; R3, subtracts multiplication of top halfword of R5
                      ; with bottom halfword of R3, writes to R4.
```

## 12.6.6.10 SMUL and SMULW

Signed Multiply (halfwords) and Signed Multiply (word by halfword)

Syntax

*op{XY}{cond} Rd, Rn, Rm*  
*op{Y}{cond} Rd, Rn, Rm*

For SMULXY only:

*op*      is one of:

**SMUL{XY}**      Signed Multiply (halfwords).

*X* and *Y* specify which halfword of the source registers *Rn* and *Rm* is used as the first and second multiply operand.

If *X* is B, then the bottom halfword, bits [15:0] of *Rn* is used.

If *X* is T, then the top halfword, bits [31:16] of *Rn* is used. If *Y* is B, then the bottom halfword, bits [15:0], of *Rm* is used.

If *Y* is T, then the top halfword, bits [31:16], of *Rm* is used.

**SMULW{Y}**      Signed Multiply (word by halfword).

*Y* specifies which halfword of the source register *Rm* is used as the second multiply operand.

If *Y* is B, then the bottom halfword (bits [15:0]) of *Rm* is used.

If *Y* is T, then the top halfword (bits [31:16]) of *Rm* is used.

*cond*      is an optional condition code, see “[Conditional Execution](#)” .

*Rd*      is the destination register.

*Rn, Rm*      are registers holding the first and second operands.

Operation

The SMULBB, SMULTB, SMULBT and SMULTT instructions interprets the values from *Rn* and *Rm* as four signed 16-bit integers. These instructions:

- Multiplies the specified signed halfword, Top or Bottom, values from *Rn* and *Rm*.
- Writes the 32-bit result of the multiplication in *Rd*.

The SMULWT and SMULWB instructions interprets the values from *Rn* as a 32-bit signed integer and *Rm* as two halfword 16-bit signed integers. These instructions:

- Multiplies the first operand and the top, T suffix, or the bottom, B suffix, halfword of the second operand.
- Writes the signed most significant 32 bits of the 48-bit result in the destination register.

Restrictions

In these instructions:

- Do not use SP and do not use PC.
- *RdHi* and *RdLo* must be different registers.

Examples

```
SMULBT    R0, R4, R5 ; Multiplies the bottom halfword of R4 with the
                      ; top halfword of R5, multiplies results and
                      ; writes to R0
SMULBB    R0, R4, R5 ; Multiplies the bottom halfword of R4 with the
                      ; bottom halfword of R5, multiplies results and
                      ; writes to R0
SMULTT    R0, R4, R5 ; Multiplies the top halfword of R4 with the top
                      ; halfword of R5, multiplies results and writes
                      ; to R0
SMULTB    R0, R4, R5 ; Multiplies the top halfword of R4 with the
                      ; bottom halfword of R5, multiplies results and
```

		; and writes to R0
SMULWT	R4, R5, R3	; Multiplies R5 with the top halfword of R3, ; extracts top 32 bits and writes to R4
SMULWB	R4, R5, R3	; Multiplies R5 with the bottom halfword of R3, ; extracts top 32 bits and writes to R4.

#### 12.6.6.11 UMULL, UMLAL, SMULL, and SMLAL

Signed and Unsigned Long Multiply, with optional Accumulate, using 32-bit operands and producing a 64-bit result.

##### Syntax

*op{cond} RdLo, RdHi, Rn, Rm*

where:

*op*      is one of:

UMULL Unsigned Long Multiply.

UMLAL Unsigned Long Multiply, with Accumulate.

SMULL Signed Long Multiply.

SMLAL Signed Long Multiply, with Accumulate.

*cond*      is an optional condition code, see “[Conditional Execution](#)” .

*RdHi, RdLo*      are the destination registers. For UMLAL and SMLAL they also hold the accumulating value.

*Rn, Rm*      are registers holding the operands.

##### Operation

The UMULL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The UMLAL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers, adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

The SMULL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The SMLAL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers, adds the 64-bit result to the 64-bit signed integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

##### Restrictions

In these instructions:

- Do not use SP and do not use PC
- *RdHi* and *RdLo* must be different registers.

##### Condition Flags

These instructions do not affect the condition code flags.

##### Examples

UMULL	R0, R4, R5, R6	; Unsigned (R4,R0) = R5 x R6
SMLAL	R4, R5, R3, R8	; Signed (R5,R4) = (R5,R4) + R3 x R8

## 12.6.6.12 SDIV and UDIV

Signed Divide and Unsigned Divide.

Syntax

```
SDIV{cond} {Rd,} Rn, Rm  
UDIV{cond} {Rd,} Rn, Rm
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register. If Rd is omitted, the destination register is Rn.

Rn is the register holding the value to be divided.

Rm is a register holding the divisor.

Operation

SDIV performs a signed integer division of the value in Rn by the value in Rm.

UDIV performs an unsigned integer division of the value in Rn by the value in Rm.

For both instructions, if the value in Rn is not divisible by the value in Rm, the result is rounded towards zero.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SDIV  R0, R2, R4 ; Signed divide, R0 = R2/R4  
UDIV  R8, R8, R1 ; Unsigned divide, R8 = R8/R1
```

## 12.6.7 Saturating Instructions

The table below shows the saturating instructions:

**Table 12-22. Saturating Instructions**

Mnemonic	Description
SSAT	Signed Saturate
SSAT16	Signed Saturate Halfword
USAT	Unsigned Saturate
USAT16	Unsigned Saturate Halfword
QADD	Saturating Add
QSUB	Saturating Subtract
QSUB16	Saturating Subtract 16
QASX	Saturating Add and Subtract with Exchange
QSAX	Saturating Subtract and Add with Exchange
QDADD	Saturating Double and Add
QDSUB	Saturating Double and Subtract
UQADD16	Unsigned Saturating Add 16
UQADD8	Unsigned Saturating Add 8
UQASX	Unsigned Saturating Add and Subtract with Exchange
UQSAX	Unsigned Saturating Subtract and Add with Exchange
UQSUB16	Unsigned Saturating Subtract 16
UQSUB8	Unsigned Saturating Subtract 8

For signed  $n$ -bit saturation, this means that:

- If the value to be saturated is less than  $-2^{n-1}$ , the result returned is  $-2^{n-1}$
- If the value to be saturated is greater than  $2^{n-1}-1$ , the result returned is  $2^{n-1}-1$
- Otherwise, the result returned is the same as the value to be saturated.

For unsigned  $n$ -bit saturation, this means that:

- If the value to be saturated is less than 0, the result returned is 0
- If the value to be saturated is greater than  $2^n-1$ , the result returned is  $2^n-1$
- Otherwise, the result returned is the same as the value to be saturated.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the instruction sets the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. To clear the Q flag to 0, the MSR instruction must be used; see “[MSR](#)” .

To read the state of the Q flag, the MRS instruction must be used; see “[MRS](#)” .

### 12.6.7.1 SSAT and USAT

Signed Saturate and Unsigned Saturate to any bit position, with optional shift before saturating.

#### Syntax

*op{cond} Rd, #n, Rm {, shift #s}*

where:

*op*

is one of:

SSAT Saturates a signed value to a signed range.

USAT Saturates a signed value to an unsigned range.

*cond*

is an optional condition code, see “[Conditional Execution](#)” .

*Rd*

is the destination register.

*n*

specifies the bit position to saturate to:

*n* ranges from 1 to 32 for SSAT

*n* ranges from 0 to 31 for USAT.

*Rm*

is the register containing the value to saturate.

*shift #s*

is an optional shift applied to *Rm* before saturating. It must be one of the following:

ASR #s

where s is in the range 1 to 31.

LSL #s

where s is in the range 0 to 31.

#### Operation

These instructions saturate to a signed or unsigned *n*-bit value.

The SSAT instruction applies the specified shift, then saturates to the signed range  $-2^{n-1} \leq x \leq 2^{n-1}-1$ .

The USAT instruction applies the specified shift, then saturates to the unsigned range  $0 \leq x \leq 2^n-1$ .

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

These instructions do not affect the condition code flags.

If saturation occurs, these instructions set the Q flag to 1.

#### Examples

```
SSAT      R7, #16, R7, LSL #4 ; Logical shift left value in R7 by 4, then  
                                ; saturate it as a signed 16-bit value and  
                                ; write it back to R7  
USATNE   R0, #7, R5          ; Conditionally saturate value in R5 as an  
                                ; unsigned 7 bit value and write it to R0.
```

## 12.6.7.2 SSAT16 and USAT16

Signed Saturate and Unsigned Saturate to any bit position for two halfwords.

### Syntax

*op{cond} Rd, #n, Rm*

where:

*op* is one of:

SSAT16 Saturates a signed halfword value to a signed range.

USAT16 Saturates a signed halfword value to an unsigned range.

*cond*

is an optional condition code, see “[Conditional Execution](#)” .

*Rd*

is the destination register.

*n*

specifies the bit position to saturate to:

*n* ranges from 1

*n* ranges from 0 to 15 for USAT.

to 16 for SSAT

*Rm*

is the register containing the value to saturate.

### Operation

#### The SSAT16 instruction:

Saturates two signed 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two signed 16-bit halfwords to the destination register.

#### The USAT16 instruction:

Saturates two unsigned 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two unsigned halfwords in the destination register.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

These instructions do not affect the condition code flags.

If saturation occurs, these instructions set the Q flag to 1.

### Examples

```
SSAT16    R7, #9, R2 ; Saturates the top and bottom highwords of R2  
; as 9-bit values, writes to corresponding halfword  
; of R7  
USAT16NE  R0, #13, R5 ; Conditionally saturates the top and bottom  
; halfwords of R5 as 13-bit values, writes to  
; corresponding halfword of R0.
```

### 12.6.7.3 QADD and QSUB

Saturating Add and Saturating Subtract, signed.

#### Syntax

```
op{cond} {Rd}, Rn, Rm  
op{cond} {Rd}, Rn, Rm
```

where:

op      is one of:

- QADD Saturating 32-bit add.
- QADD8 Saturating four 8-bit integer additions.
- QADD16 Saturating two 16-bit integer additions.
- QSUB Saturating 32-bit subtraction.
- QSUB8 Saturating four 8-bit integer subtraction.
- QSUB16 Saturating two 16-bit integer subtraction.

cond     is an optional condition code, see “[Conditional Execution](#)” .

Rd      is the destination register.

Rn, Rm    are registers holding the first and second operands.

#### Operation

These instructions add or subtract two, four or eight values from the first and second operands and then writes a signed saturated value in the destination register.

The QADD and QSUB instructions apply the specified add or subtract, and then saturate the result to the signed range -  $2^{n-1} \leq x \leq 2^{n-1}-1$ , where x is given by the number of bits applied in the instruction, 32, 16 or 8.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the QADD and QSUB instructions set the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. The 8-bit and 16-bit QADD and QSUB instructions always leave the Q flag unchanged.

To clear the Q flag to 0, the MSR instruction must be used; see “[MSR](#)” .

To read the state of the Q flag, the MRS instruction must be used; see “[MRS](#)” .

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

These instructions do not affect the condition code flags.

If saturation occurs, these instructions set the Q flag to 1.

#### Examples

```
QADD16  R7, R4, R2 ; Adds halfwords of R4 with corresponding halfword of  
; R2, saturates to 16 bits and writes to  
; corresponding halfword of R7  
QADD8   R3, R1, R6 ; Adds bytes of R1 to the corresponding bytes of R6,  
; saturates to 8 bits and writes to corresponding  
; byte of R3  
QSUB16  R4, R2, R3 ; Subtracts halfwords of R3 from corresponding  
; halfword of R2, saturates to 16 bits, writes to  
; corresponding halfword of R4  
QSUB8   R4, R2, R5 ; Subtracts bytes of R5 from the corresponding byte  
; in R2, saturates to 8 bits, writes to corresponding  
; byte of R4.
```

#### 12.6.7.4 QASX and QSAX

Saturating Add and Subtract with Exchange and Saturating Subtract and Add with Exchange, signed.

##### Syntax

*op{cond} {Rd}, Rm, Rn*

where:

*op* is one of:

QASX Add and Subtract with Exchange and Saturate.

QSAX Subtract and Add with Exchange and Saturate.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rn, Rm* are registers holding the first and second operands.

##### Operation

The QASX instruction:

1. Adds the top halfword of the source operand with the bottom halfword of the second operand.
2. Subtracts the top halfword of the second operand from the bottom highword of the first operand.
3. Saturates the result of the subtraction and writes a 16-bit signed integer in the range  $-2^{15} \leq x \leq 2^{15} - 1$ , where  $x$  equals 16, to the bottom halfword of the destination register.
4. Saturates the results of the sum and writes a 16-bit signed integer in the range  $-2^{15} \leq x \leq 2^{15} - 1$ , where  $x$  equals 16, to the top halfword of the destination register.

The QSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Adds the bottom halfword of the source operand with the top halfword of the second operand.
3. Saturates the results of the sum and writes a 16-bit signed integer in the range  $-2^{15} \leq x \leq 2^{15} - 1$ , where  $x$  equals 16, to the bottom halfword of the destination register.
4. Saturates the result of the subtraction and writes a 16-bit signed integer in the range  $-2^{15} \leq x \leq 2^{15} - 1$ , where  $x$  equals 16, to the top halfword of the destination register.

##### Restrictions

Do not use SP and do not use PC.

##### Condition Flags

These instructions do not affect the condition code flags.

##### Examples

```
QASX  R7, R4, R2 ; Adds top halfword of R4 to bottom halfword of R2,  
; saturates to 16 bits, writes to top halfword of R7  
; Subtracts top highword of R2 from bottom halfword of  
; R4, saturates to 16 bits and writes to bottom halfword  
; of R7  
QSAX  R0, R3, R5 ; Subtracts bottom halfword of R5 from top halfword of  
; R3, saturates to 16 bits, writes to top halfword of R0  
; Adds bottom halfword of R3 to top halfword of R5,  
; saturates to 16 bits, writes to bottom halfword of R0.
```

## 12.6.7.5 QDADD and QDSUB

Saturating Double and Add and Saturating Double and Subtract, signed.

### Syntax

*op{cond} {Rd}, Rm, Rn*

where:

*op* is one of:

QDADD Saturating Double and Add.

QDSUB Saturating Double and Subtract.

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rd* is the destination register.

*Rm, Rn* are registers holding the first and second operands.

### Operation

The QDADD instruction:

- Doubles the second operand value.
- Adds the result of the doubling to the signed saturated value in the first operand.
- Writes the result to the destination register.

The QDSUB instruction:

- Doubles the second operand value.
- Subtracts the doubled value from the signed saturated value in the first operand.
- Writes the result to the destination register.

Both the doubling and the addition or subtraction have their results saturated to the 32-bit signed integer range  $-2^{31} \leq x \leq 2^{31}-1$ . If saturation occurs in either operation, it sets the Q flag in the APSR.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

If saturation occurs, these instructions set the Q flag to 1.

### Examples

```
QDADD    R7, R4, R2 ; Doubles and saturates R4 to 32 bits, adds R2,  
; saturates to 32 bits, writes to R7  
QDSUB    R0, R3, R5 ; Subtracts R3 doubled and saturated to 32 bits  
; from R5, saturates to 32 bits, writes to R0.
```

## 12.6.7.6 UQASX and UQSAX

Saturating Add and Subtract with Exchange and Saturating Subtract and Add with Exchange, unsigned.

### Syntax

*op{cond} {Rd}, Rm, Rn*

where:

type      is one of:

UQASX Add and Subtract with Exchange and Saturate.

UQSAX Subtract and Add with Exchange and Saturate.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Rd      is the destination register.

Rn, Rm      are registers holding the first and second operands.

### Operation

The UQASX instruction:

1. Adds the bottom halfword of the source operand with the top halfword of the second operand.
2. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
3. Saturates the results of the sum and writes a 16-bit unsigned integer in the range  $0 \leq x \leq 2^{16} - 1$ , where  $x$  equals 16, to the top halfword of the destination register.
4. Saturates the result of the subtraction and writes a 16-bit unsigned integer in the range  $0 \leq x \leq 2^{16} - 1$ , where  $x$  equals 16, to the bottom halfword of the destination register.

The UQSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Adds the bottom halfword of the first operand with the top halfword of the second operand.
3. Saturates the result of the subtraction and writes a 16-bit unsigned integer in the range  $0 \leq x \leq 2^{16} - 1$ , where  $x$  equals 16, to the top halfword of the destination register.
4. Saturates the results of the addition and writes a 16-bit unsigned integer in the range  $0 \leq x \leq 2^{16} - 1$ , where  $x$  equals 16, to the bottom halfword of the destination register.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

These instructions do not affect the condition code flags.

### Examples

```
UQASX  R7, R4, R2 ; Adds top halfword of R4 with bottom halfword of R2,  
; saturates to 16 bits, writes to top halfword of R7  
; Subtracts top halfword of R2 from bottom halfword of  
; R4, saturates to 16 bits, writes to bottom halfword of R7  
UQSAX  R0, R3, R5 ; Subtracts bottom halfword of R5 from top halfword of R3,  
; saturates to 16 bits, writes to top halfword of R0  
; Adds bottom halfword of R4 to top halfword of R5  
; saturates to 16 bits, writes to bottom halfword of R0.
```

## 12.6.7.7 UQADD and UQSUB

Saturating Add and Saturating Subtract Unsigned.

Syntax

```
op{cond} {Rd}, Rn, Rm  
op{cond} {Rd}, Rn, Rm
```

where:

op is one of:

- UQADD8 Saturating four unsigned 8-bit integer additions.
- UQADD16 Saturating two unsigned 16-bit integer additions.
- UDSUB8 Saturating four unsigned 8-bit integer subtractions.
- UQSUB16 Saturating two unsigned 16-bit integer subtractions.

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

Rn, Rm are registers holding the first and second operands.

Operation

These instructions add or subtract two or four values and then writes an unsigned saturated value in the destination register.

The UQADD16 instruction:

- Adds the respective top and bottom halfwords of the first and second operands.
- Saturates the result of the additions for each halfword in the destination register to the unsigned range  $0 \leq x \leq 2^{16}-1$ , where x is 16.

The UQADD8 instruction:

- Adds each respective byte of the first and second operands.
- Saturates the result of the addition for each byte in the destination register to the unsigned range  $0 \leq x \leq 2^8-1$ , where x is 8.

The UQSUB16 instruction:

- Subtracts both halfwords of the second operand from the respective halfwords of the first operand.
- Saturates the result of the differences in the destination register to the unsigned range  $0 \leq x \leq 2^{16}-1$ , where x is 16.

The UQSUB8 instructions:

- Subtracts the respective bytes of the second operand from the respective bytes of the first operand.
- Saturates the results of the differences for each byte in the destination register to the unsigned range  $0 \leq x \leq 2^8-1$ , where x is 8.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UQADD16 R7, R4, R2 ; Adds halfwords in R4 to corresponding halfword in R2,  
; saturates to 16 bits, writes to corresponding halfword of R7  
UQADD8  R4, R2, R5 ; Adds bytes of R2 to corresponding byte of R5, saturates  
; to 8 bits, writes to corresponding bytes of R4  
UQSUB16 R6, R3, R0 ; Subtracts halfwords in R0 from corresponding halfword  
; in R3, saturates to 16 bits, writes to corresponding  
; halfword in R6  
UQSUB8  R1, R5, R6 ; Subtracts bytes in R6 from corresponding byte of R5,  
; saturates to 8 bits, writes to corresponding byte of R1.
```

## 12.6.8 Packing and Unpacking Instructions

The table below shows the instructions that operate on packing and unpacking data:

**Table 12-23. Packing and Unpacking Instructions**

Mnemonic	Description
PKH	Pack Halfword
SXTAB	Extend 8 bits to 32 and add
SXTAB16	Dual extend 8 bits to 16 and add
SXTAH	Extend 16 bits to 32 and add
SXTB	Sign extend a byte
SXTB16	Dual extend 8 bits to 16 and add
SXTH	Sign extend a halfword
UXTAB	Extend 8 bits to 32 and add
UXTAB16	Dual extend 8 bits to 16 and add
UXTAH	Extend 16 bits to 32 and add
UXTB	Zero extend a byte
UXTB16	Dual zero extend 8 bits to 16 and add
UXTH	Zero extend a halfword

### 12.6.8.1 PKHBT and PKHTB

Pack Halfword

Syntax

```
op{cond} {Rd}, Rn, Rm {, LSL #imm}  
op{cond} {Rd}, Rn, Rm {, ASR #imm}
```

where:

op      is one of:

PKHBT Pack Halfword, bottom and top with shift.

PKHTB Pack Halfword, top and bottom with shift.

cond     is an optional condition code, see "[Conditional Execution](#)" .

Rd      is the destination register.

Rn      is the first operand register

Rm      is the second operand register holding the value to be optionally shifted.

imm     is the shift length. The type of shift length depends on the instruction:

For PKHBT

LSL a left shift with a shift length from 1 to 31, 0 means no shift.

For PKHTB

ASR an arithmetic shift right with a shift length from 1 to 32,  
a shift of 32-bits is encoded as 0b00000.

Operation

The PKHBT instruction:

1. Writes the value of the bottom halfword of the first operand to the bottom halfword of the destination register.
2. If shifted, the shifted value of the second operand is written to the top halfword of the destination register.

The PKHTB instruction:

1. Writes the value of the top halfword of the first operand to the top halfword of the destination register.
2. If shifted, the shifted value of the second operand is written to the bottom halfword of the destination register.

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

```
PKHBT  R3, R4, R5 LSL #0 ; Writes bottom halfword of R4 to bottom halfword of  
; R3, writes top halfword of R5, unshifted, to top  
; halfword of R3  
PKHTB  R4, R0, R2 ASR #1 ; Writes R2 shifted right by 1 bit to bottom halfword  
; of R4, and writes top halfword of R0 to top  
; halfword of R4.
```

## 12.6.8.2 SXT and UXT

Sign extend and Zero extend.

### Syntax

```
op{cond} {Rd,} Rm {, ROR #n}  
op{cond} {Rd}, Rm {, ROR #n}
```

where:

op       is one of:

SXTB Sign extends an 8-bit value to a 32-bit value.

SXTH Sign extends a 16-bit value to a 32-bit value.

SXTB16 Sign extends two 8-bit values to two 16-bit values.

UXTB Zero extends an 8-bit value to a 32-bit value.

UXTH Zero extends a 16-bit value to a 32-bit value.

UXTB16 Zero extends two 8-bit values to two 16-bit values.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Rd        is the destination register.

Rm        is the register holding the value to extend.

ROR #n    is one of:

ROR #8 Value from Rm is rotated right 8 bits.

### Operation

These instructions do the following:

1. Rotate the value from Rm right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
  - SXTB extracts bits[7:0] and sign extends to 32 bits.
  - UXTB extracts bits[7:0] and zero extends to 32 bits.
  - SXTH extracts bits[15:0] and sign extends to 32 bits.
  - UXTH extracts bits[15:0] and zero extends to 32 bits.
  - SXTB16 extracts bits[7:0] and sign extends to 16 bits, and extracts bits [23:16] and sign extends to 16 bits.
  - UXTB16 extracts bits[7:0] and zero extends to 16 bits, and extracts bits [23:16] and zero extends to 16 bits.

### Restrictions

Do not use SP and do not use PC.

### Condition Flags

These instructions do not affect the flags.

### Examples

```
SXTH R4, R6, ROR #16 ; Rotates R6 right by 16 bits, obtains bottom halfword of  
; of result, sign extends to 32 bits and writes to R4  
UXTB R3, R10          ; Extracts lowest byte of value in R10, zero extends, and  
; writes to R3.
```

### 12.6.8.3 SXTA and UXTA

Signed and Unsigned Extend and Add

Syntax

```
op{cond} {Rd,} Rn, Rm {, ROR #n}  
op{cond} {Rd,} Rn, Rm {, ROR #n}
```

where:

op is one of:

SXTAB Sign extends an 8-bit value to a 32-bit value and add.

SXTAH Sign extends a 16-bit value to a 32-bit value and add.

SXTAB16 Sign extends two 8-bit values to two 16-bit values and add.

UXTAB Zero extends an 8-bit value to a 32-bit value and add.

UXTAH Zero extends a 16-bit value to a 32-bit value and add.

UXTAB16 Zero extends two 8-bit values to two 16-bit values and add.

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

Rn is the first operand register.

Rm is the register holding the value to rotate and extend.

ROR #n is one of:

ROR #8 Value from Rm is rotated right 8 bits.

ROR #16 Value from Rm is rotated right 16 bits.

ROR #24 Value from Rm is rotated right 24 bits.

If ROR #n is omitted, no rotation is performed.

Operation

These instructions do the following:

1. Rotate the value from Rm right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
  - SXTAB extracts bits[7:0] from Rm and sign extends to 32 bits.
  - UXTAB extracts bits[7:0] from Rm and zero extends to 32 bits.
  - SXTAH extracts bits[15:0] from Rm and sign extends to 32 bits.
  - UXTAH extracts bits[15:0] from Rm and zero extends to 32 bits.
  - SXTAB16 extracts bits[7:0] from Rm and sign extends to 16 bits, and extracts bits [23:16] from Rm and sign extends to 16 bits.
  - UXTAB16 extracts bits[7:0] from Rm and zero extends to 16 bits, and extracts bits [23:16] from Rm and zero extends to 16 bits.
3. Adds the signed or zero extended value to the word or corresponding halfword of Rn and writes the result in Rd.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SXTAH R4, R8, R6, ROR #16 ; Rotates R6 right by 16 bits, obtains bottom  
; halfword, sign extends to 32 bits, adds  
; R8, and writes to R4
```

```

UXTAB R3, R4, R10          ; Extracts bottom byte of R10 and zero extends
                             ; to 32 bits, adds R4, and writes to R3.

```

## 12.6.9 Bitfield Instructions

The table below shows the instructions that operate on adjacent sets of bits in registers or bitfields:

**Table 12-24. Packing and Unpacking Instructions**

Mnemonic	Description
BFC	Bit Field Clear
BFI	Bit Field Insert
SBFX	Signed Bit Field Extract
SXTB	Sign extend a byte
SXTH	Sign extend a halfword
UBFX	Unsigned Bit Field Extract
UXTB	Zero extend a byte
UXTH	Zero extend a halfword

### 12.6.9.1 BFC and BFI

Bit Field Clear and Bit Field Insert.

#### Syntax

```

BFC{cond} Rd, #lsb, #width
BFI{cond} Rd, Rn, #lsb, #width

```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .
- Rd is the destination register.
- Rn is the source register.
- lsb is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.
- width is the width of the bitfield and must be in the range 1 to 32-*lsb*.

#### Operation

BFC clears a bitfield in a register. It clears *width* bits in *Rd*, starting at the low bit position *lsb*. Other bits in *Rd* are unchanged.

BFI copies a bitfield into one register from another register. It replaces *width* bits in *Rd* starting at the low bit position *lsb*, with *width* bits from *Rn* starting at bit[0]. Other bits in *Rd* are unchanged.

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

These instructions do not affect the flags.

#### Examples

```

BFC    R4, #8, #12      ; Clear bit 8 to bit 19 (12 bits) of R4 to 0
BFI    R9, R2, #8, #12   ; Replace bit 8 to bit 19 (12 bits) of R9 with
                         ; bit 0 to bit 11 from R2.

```

### 12.6.9.2 SBFX and UBFX

Signed Bit Field Extract and Unsigned Bit Field Extract.

#### Syntax

```
SBFX{cond} Rd, Rn, #lsb, #width  
UBFX{cond} Rd, Rn, #lsb, #width
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Rd is the destination register.  
Rn is the source register.  
lsb is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.  
width is the width of the bitfield and must be in the range 1 to 32-*lsb*.

#### Operation

SBFX extracts a bitfield from one register, sign extends it to 32 bits, and writes the result to the destination register.

UBFX extracts a bitfield from one register, zero extends it to 32 bits, and writes the result to the destination register.

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

These instructions do not affect the flags.

#### Examples

```
SBFX R0, R1, #20, #4 ; Extract bit 20 to bit 23 (4 bits) from R1 and sign  
                      ; extend to 32 bits and then write the result to R0.  
UBFX R8, R11, #9, #10 ; Extract bit 9 to bit 18 (10 bits) from R11 and zero  
                      ; extend to 32 bits and then write the result to R8.
```

### 12.6.9.3 SXT and UXT

Sign extend and Zero extend.

#### Syntax

```
SXTextend{cond} {Rd,} Rm {, ROR #n}  
UXTextend{cond} {Rd}, Rm {, ROR #n}
```

where:

extend      is one of:

B Extends an 8-bit value to a 32-bit value.

H Extends a 16-bit value to a 32-bit value.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Rd      is the destination register.

Rm      is the register holding the value to extend.

ROR #n      is one of:

ROR #8 Value from *Rm* is rotated right 8 bits.

ROR #16 Value from *Rm* is rotated right 16 bits.

ROR #24 Value from *Rm* is rotated right 24 bits.

If ROR #n is omitted, no rotation is performed.

#### Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
  - SXTB extracts bits[7:0] and sign extends to 32 bits.
  - UXTB extracts bits[7:0] and zero extends to 32 bits.
  - SXTH extracts bits[15:0] and sign extends to 32 bits.
  - UXTH extracts bits[15:0] and zero extends to 32 bits.

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

These instructions do not affect the flags.

#### Examples

```
SXTH R4, R6, ROR #16 ; Rotate R6 right by 16 bits, then obtain the lower  
; halfword of the result and then sign extend to  
; 32 bits and write the result to R4.  
UXTB R3, R10 ; Extract lowest byte of the value in R10 and zero  
; extend it, and write the result to R3.
```

## 12.6.10 Branch and Control Instructions

The table below shows the branch and control instructions:

**Table 12-25. Branch and Control Instructions**

Mnemonic	Description
B	Branch
BL	Branch with Link
BLX	Branch indirect with Link
BX	Branch indirect
CBNZ	Compare and Branch if Non Zero
CBZ	Compare and Branch if Zero
IT	If-Then
TBB	Table Branch Byte
TBH	Table Branch Halfword

## 12.6.10.1 B, BL, BX, and BLX

Branch instructions.

Syntax

```
B{cond} label  
BL{cond} label  
BX{cond} Rm  
BLX{cond} Rm
```

where:

B	is branch (immediate).
BL	is branch with link (immediate).
BX	is branch indirect (register).
BLX	is branch indirect with link (register).
cond	is an optional condition code, see “ <a href="#">Conditional Execution</a> ” .
label	is a PC-relative expression. See “ <a href="#">PC-relative Expressions</a> ” .
Rm	is a register that indicates an address to branch to. Bit[0] of the value in <i>Rm</i> must be 1, but the address to branch to is created by changing bit[0] to 0.

Operation

All these instructions cause a branch to *label*, or to the address indicated in *Rm*. In addition:

- The BL and BLX instructions write the address of the next instruction to LR (the link register, R14).
- The BX and BLX instructions result in a UsageFault exception if bit[0] of *Rm* is 0.

*Bcond* label is the only conditional instruction that can be either inside or outside an IT block. All other branch instructions must be conditional inside an IT block, and must be unconditional outside the IT block, see “[IT](#)” .

The table below shows the ranges for the various branch instructions.

**Table 12-26. Branch Ranges**

Instruction	Branch Range
B label	-16 MB to +16 MB
B <sub>cond</sub> label (outside IT block)	-1 MB to +1 MB
B <sub>cond</sub> label (inside IT block)	-16 MB to +16 MB
BL{cond} label	-16 MB to +16 MB
BX{cond} Rm	Any value in register
BLX{cond} Rm	Any value in register

The .W suffix might be used to get the maximum branch range. See “[Instruction Width Selection](#)” .

Restrictions

The restrictions are:

- Do not use PC in the BLX instruction
- For BX and BLX, bit[0] of *Rm* must be 1 for correct execution but a branch occurs to the target address created by changing bit[0] to 0
- When any of these instructions is inside an IT block, it must be the last instruction of the IT block.

*Bcond* is the only conditional instruction that is not required to be inside an IT block. However, it has a longer branch range when it is inside an IT block.

Condition Flags

These instructions do not change the flags.

#### Examples

```
B      loopA    ; Branch to loopA
BLE    ng       ; Conditionally branch to label ng
B.W   target   ; Branch to target within 16MB range
BEQ   target   ; Conditionally branch to target
BEQ.W target   ; Conditionally branch to target within 1MB
BL    funC     ; Branch with link (Call) to function funC, return address
          ; stored in LR
BX    LR       ; Return from function call
BXNE  R0       ; Conditionally branch to address stored in R0
BLX   R0       ; Branch with link and exchange (Call) to a address stored in R0.
```

### 12.6.10.2 CBZ and CBNZ

Compare and Branch on Zero, Compare and Branch on Non-Zero.

#### Syntax

```
CBZ  Rn, label
CBNZ Rn, label
```

where:

Rn is the register holding the operand.

label is the branch destination.

#### Operation

Use the CBZ or CBNZ instructions to avoid changing the condition code flags and to reduce the number of instructions.

CBZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP  Rn, #0
BEQ  label
```

CBNZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP  Rn, #0
BNE  label
```

#### Restrictions

The restrictions are:

- $Rn$  must be in the range of R0 to R7
- The branch destination must be within 4 to 130 bytes after the instruction
- These instructions must not be used inside an IT block.

#### Condition Flags

These instructions do not change the flags.

#### Examples

```
CBZ  R5, target ; Forward branch if R5 is zero
CBNZ R0, target ; Forward branch if R0 is not zero
```

### 12.6.10.3 IT

If-Then condition instruction.

Syntax

IT{x{y{z}}}{cond}

where:

- x specifies the condition switch for the second instruction in the IT block.
- y specifies the condition switch for the third instruction in the IT block.
- z specifies the condition switch for the fourth instruction in the IT block.
- cond specifies the condition for the first instruction in the IT block.

The condition switch for the second, third and fourth instruction in the IT block can be either:

- T Then. Applies the condition *cond* to the instruction.
- E Else. Applies the inverse condition of *cond* to the instruction.

It is possible to use AL (the *always* condition) for *cond* in an IT instruction. If this is done, all of the instructions in the IT block must be unconditional, and each of x, y, and z must be T or omitted but not E.

Operation

The IT instruction makes up to four following instructions conditional. The conditions can be all the same, or some of them can be the logical inverse of the others. The conditional instructions following the IT instruction form the *IT block*.

The instructions in the IT block, including any branches, must specify the condition in the {*cond*} part of their syntax.

The assembler might be able to generate the required IT instructions for conditional instructions automatically, so that the user does not have to write them. See the assembler documentation for details.

A BKPT instruction in an IT block is always executed, even if its condition fails.

Exceptions can be taken between an IT instruction and the corresponding IT block, or within an IT block. Such an exception results in entry to the appropriate exception handler, with suitable return information in LR and stacked PSR.

Instructions designed for use for exception returns can be used as normal to return from the exception, and execution of the IT block resumes correctly. This is the only way that a PC-modifying instruction is permitted to branch to an instruction in an IT block.

Restrictions

The following instructions are not permitted in an IT block:

- IT
- CBZ and CBNZ
- CPSID and CPSIE.

Other restrictions when using an IT block are:

- A branch or any instruction that modifies the PC must either be outside an IT block or must be the last instruction inside the IT block. These are:
  - ADD PC, PC, Rm
  - MOV PC, Rm
  - B, BL, BX, BLX
  - Any LDM, LDR, or POP instruction that writes to the PC
  - TBB and TBH
- Do not branch to any instruction inside an IT block, except when returning from an exception handler
- All conditional instructions except B*cond* must be inside an IT block. B*cond* can be either outside or inside an IT block but has a larger branch range if it is inside one
- Each instruction inside the IT block must specify a condition code suffix that is either the same or logical inverse as for the other instructions in the block.

Your assembler might place extra restrictions on the use of IT blocks, such as prohibiting the use of assembler directives within them.

#### Condition Flags

This instruction does not change the flags.

#### Example

```
ITTE    NE          ; Next 3 instructions are conditional
ANDNE  R0, R0, R1  ; ANDNE does not update condition flags
ADDSNE R2, R2, #1  ; ADDSNE updates condition flags
MOVEQ   R2, R3     ; Conditional move

CMP    R0, #9       ; Convert R0 hex value (0 to 15) into ASCII
                   ; ('0'-'9', 'A'-'F')
ITE    GT          ; Next 2 instructions are conditional
ADDGT  R1, R0, #55  ; Convert 0xA -> 'A'
ADDLE  R1, R0, #48  ; Convert 0x0 -> '0'

IT     GT          ; IT block with only one conditional instruction
ADDGT  R1, R1, #1  ; Increment R1 conditionally

ITTEE  EQ          ; Next 4 instructions are conditional
MOVEQ  R0, R1      ; Conditional move
ADDEQ  R2, R2, #10  ; Conditional add
ANDNE  R3, R3, #1  ; Conditional AND
BNE.W  dloop      ; Branch instruction can only be used in the last
                   ; instruction of an IT block

IT     NE          ; Next instruction is conditional
ADD   R0, R0, R1   ; Syntax error: no condition code used in IT block
```

#### 12.6.10.4 TBB and TBH

Table Branch Byte and Table Branch Halfword.

##### Syntax

```
TBB [Rn, Rm]  
TBH [Rn, Rm, LSL #1]
```

where:

Rn is the register containing the address of the table of branch lengths.

If Rn is PC, then the address of the table is the address of the byte immediately following the TBB or TBH instruction.

Rm is the index register. This contains an index into the table. For halfword tables, LSL #1 doubles the value in Rm to form the right offset into the table.

##### Operation

These instructions cause a PC-relative forward branch using a table of single byte offsets for TBB, or halfword offsets for TBH. Rn provides a pointer to the table, and Rm supplies an index into the table. For TBB the branch offset is twice the unsigned value of the byte returned from the table, and for TBH the branch offset is twice the unsigned value of the halfword returned from the table. The branch occurs to the address at that offset from the address of the byte immediately after the TBB or TBH instruction.

##### Restrictions

The restrictions are:

- Rn must not be SP
- Rm must not be SP and must not be PC
- When any of these instructions is used inside an IT block, it must be the last instruction of the IT block.

##### Condition Flags

These instructions do not change the flags.

##### Examples

```
ADR.W R0, BranchTable_Byte  
TBB [R0, R1] ; R1 is the index, R0 is the base address of the  
; branch table  
Case1  
; an instruction sequence follows  
Case2  
; an instruction sequence follows  
Case3  
; an instruction sequence follows  
BranchTable_Byte  
DCB 0 ; Case1 offset calculation  
DCB ((Case2-Case1)/2) ; Case2 offset calculation  
DCB ((Case3-Case1)/2) ; Case3 offset calculation  
  
TBH [PC, R1, LSL #1] ; R1 is the index, PC is used as base of the  
; branch table  
BranchTable_H  
DCI ((CaseA - BranchTable_H)/2) ; CaseA offset calculation  
DCI ((CaseB - BranchTable_H)/2) ; CaseB offset calculation  
DCI ((CaseC - BranchTable_H)/2) ; CaseC offset calculation  
  
CaseA  
; an instruction sequence follows
```

```

CaseB
; an instruction sequence follows
CaseC
; an instruction sequence follows

```

### 12.6.11 Floating-point Instructions

The table below shows the floating-point instructions.

These instructions are only available if the FPU is included, and enabled, in the system. See “[Enabling the FPU](#)” for information about enabling the floating-point unit.

**Table 12-27. Floating-point Instructions**

Mnemonic	Description
VABS	Floating-point Absolute
VADD	Floating-point Add
VCMP	Compare two floating-point registers, or one floating-point register and zero
VCMPE	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check
VCVT	Convert between floating-point and integer
VCVT	Convert between floating-point and fixed point
VCVTR	Convert between floating-point and integer with rounding
VCVTB	Converts half-precision value to single-precision
VCVTT	Converts single-precision register to half-precision
VDIV	Floating-point Divide
VFMA	Floating-point Fused Multiply Accumulate
VFNMA	Floating-point Fused Negate Multiply Accumulate
VFMS	Floating-point Fused Multiply Subtract
VFNMS	Floating-point Fused Negate Multiply Subtract
VLDM	Load Multiple extension registers
VLDR	Loads an extension register from memory
VLMA	Floating-point Multiply Accumulate
VLMS	Floating-point Multiply Subtract
VMOV	Floating-point Move Immediate
VMOV	Floating-point Move Register
VMOV	Copy ARM core register to single precision
VMOV	Copy 2 ARM core registers to 2 single precision
VMOV	Copies between ARM core register to scalar
VMOV	Copies between Scalar to ARM core register
VMRS	Move to ARM core register from floating-point System Register
VMSR	Move to floating-point System Register from ARM Core register
VMUL	Multiply floating-point
VNEG	Floating-point negate

**Table 12-27. Floating-point Instructions (Continued)**

Mnemonic	Description
VNMLA	Floating-point multiply and add
VNMLS	Floating-point multiply and subtract
VNMUL	Floating-point multiply
VPOP	Pop extension registers
VPUSH	Push extension registers
VSQRT	Floating-point square root
VSTM	Store Multiple extension registers
VSTR	Stores an extension register to memory
VSUB	Floating-point Subtract

#### 12.6.11.1 VABS

Floating-point Absolute.

Syntax

VABS{cond}.F32 Sd, Sm

where:

cond is an optional condition code, see “Conditional Execution” .

Sd, Sm are the destination floating-point value and the operand floating-point value.

Operation

This instruction:

1. Takes the absolute value of the operand floating-point register.
2. Places the results in the destination floating-point register.

Restrictions

There are no restrictions.

Condition Flags

The floating-point instruction clears the sign bit.

Examples

VABS.F32 S4, S6

#### 12.6.11.2 VADD

Floating-point Add

Syntax

VADD{cond}.F32 {Sd,} Sn, Sm

where:

cond is an optional condition code, see “Conditional Execution” .

Sd, is the destination floating-point value.

Sn, Sm are the operand floating-point values.

Operation

This instruction:

1. Adds the values in the two floating-point operand registers.
2. Places the results in the destination floating-point register.

Restrictions

There are no restrictions.

Condition Flags

This instruction does not change the flags.

Examples

VADD.F32 S4, S6, S7

### 12.6.11.3 VCMP, VCMPE

Compares two floating-point registers, or one floating-point register and zero.

#### Syntax

```
VCMP{E}{cond}.F32 Sd, Sm  
VCMP{E}{cond}.F32 Sd, #0.0
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
E If present, any NaN operand causes an Invalid Operation exception.  
Otherwise, only a signaling NaN causes the exception.  
Sd is the floating-point operand to compare.  
Sm is the floating-point operand that is compared with.

#### Operation

This instruction:

1. Compares:
  - Two floating-point registers.
  - One floating-point register and zero.
2. Writes the result to the FPSCR flags.

#### Restrictions

This instruction can optionally raise an Invalid Operation exception if either operand is any type of NaN. It always raises an Invalid Operation exception if either operand is a signaling NaN.

#### Condition Flags

When this instruction writes the result to the FPSCR flags, the values are normally transferred to the ARM flags by a subsequent VMRS instruction, see “[“](#)” .

#### Examples

```
VCMP.F32 S4, #0.0  
VCMP.F32 S4, S2
```

#### 12.6.11.4 VCVT, VCVTR between Floating-point and Integer

Converts a value in a register from floating-point to a 32-bit integer.

##### Syntax

```
VCVT{R}{cond}.Tm.F32 Sd, Sm  
VCVT{cond}.F32.Tm Sd, Sm
```

where:

R                    If R is specified, the operation uses the rounding mode specified by the FPSCR.  
If R is omitted, the operation uses the Round towards Zero rounding mode.

cond                is an optional condition code, see “[Conditional Execution](#)” .

Tm                is the data type for the operand. It must be one of:

S32 signed 32-bit value.

bit value.

Sd, Sm                are the destination register and the operand register.

##### Operation

These instructions:

1. Either
  - Converts a value in a register from floating-point value to a 32-bit integer.
  - Converts from a 32-bit integer to floating-point value.
2. Places the result in a second register.

The floating-point to integer operation normally uses the *Round towards Zero* rounding mode, but can optionally use the rounding mode specified by the FPSCR.

The integer to floating-point operation uses the rounding mode specified by the FPSCR.

##### Restrictions

There are no restrictions.

##### Condition Flags

These instructions do not change the flags.

## 12.6.11.5 VCVT between Floating-point and Fixed-point

Converts a value in a register from floating-point to and from fixed-point.

### Syntax

```
VCVT{cond}.Td.F32 Sd, Sd, #fbits  
VCVT{cond}.F32.Td Sd, Sd, #fbits
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .
- Td is the data type for the fixed-point number. It must be one of:
- S16 signed 16-bit value.
  - U16 unsigned 16-bit value.
  - S32 signed 32-bit value.
  - U32 unsigned 32-bit value.
- Sd is the destination register and the operand register.
- fbits is the number of fraction bits in the fixed-point number:  
If *Td* is S16 or U16, *fbits* must be in the range 0-16.  
If *Td* is S32 or U32, *fbits* must be in the range 1-32.

### Operation

These instructions:

1. Either
  - Converts a value in a register from floating-point to fixed-point.
  - Converts a value in a register from fixed-point to floating-point.
2. Places the result in a second register.

The floating-point values are single-precision.

The fixed-point value can be 16-bit or 32-bit. Conversions from fixed-point values take their operand from the low-order bits of the source register and ignore any remaining bits.

Signed conversions to fixed-point values sign-extend the result value to the destination register width.

Unsigned conversions to fixed-point values zero-extend the result value to the destination register width.

The floating-point to fixed-point operation uses the *Round towards Zero* rounding mode. The fixed-point to floating-point operation uses the *Round to Nearest* rounding mode.

### Restrictions

There are no restrictions.

### Condition Flags

These instructions do not change the flags.

## 12.6.11.6 VCVTB, VCVTT

Converts between a half-precision value and a single-precision value.

### Syntax

```
VCVT{y}{cond}.F32.F16 Sd, Sm  
VCVT{y}{cond}.F16.F32 Sd, Sm
```

where:

y        Specifies which half of the operand register Sm or destination register Sd is used for the operand or destination:  
- If y is B, then the bottom half, bits [15:0], of Sm or Sd is used.  
- If y is T, then the top half, bits [31:16], of Sm or Sd is used.

cond      is an optional condition code, see “[Conditional Execution](#)” .

Sd        is the destination register.

Sm        is the operand register.

### Operation

This instruction with the.F16.32 suffix:

1. Converts the half-precision value in the top or bottom half of a single-precision register to single-precision.
2. Writes the result to a single-precision register.

This instruction with the.F32.F16 suffix:

1. Converts the value in a single-precision register to half-precision.
2. Writes the result into the top or bottom half of a single-precision register, preserving the other half of the target register.

### Restrictions

There are no restrictions.

### Condition Flags

These instructions do not change the flags.

## 12.6.11.7 VDIV

Divides floating-point values.

### Syntax

```
VDIV{cond}.F32 {Sd,} Sn, Sm
```

where:

cond      is an optional condition code, see “[Conditional Execution](#)” .

Sd        is the destination register.

Sn, Sm     are the operand registers.

### Operation

This instruction:

1. Divides one floating-point value by another floating-point value.
2. Writes the result to the floating-point destination register.

### Restrictions

There are no restrictions.

### Condition Flags

These instructions do not change the flags.

## 12.6.11.8 VFMA, VFMS

Floating-point Fused Multiply Accumulate and Subtract.

Syntax

```
VFMA{cond}.F32 {Sd,} Sn, Sm  
VFMS{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Sd is the destination register.

Sn, Sm are the operand registers.

Operation

The VFMA instruction:

1. Multiplies the floating-point values in the operand registers.
2. Accumulates the results into the destination register.

The result of the multiply is not rounded before the accumulation.

The VFMS instruction:

1. Negates the first operand register.
2. Multiplies the floating-point values of the first and second operand registers.
3. Adds the products to the destination register.
4. Places the results in the destination register.

The result of the multiply is not rounded before the addition.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

## 12.6.11.9 VFNMA, VFNMS

Floating-point Fused Negate Multiply Accumulate and Subtract.

### Syntax

```
VFNMA{cond}.F32 {Sd,} Sn, Sm  
VFNMS{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Sd is the destination register.

Sn, Sm are the operand registers.

### Operation

The VFNMA instruction:

1. Negates the first floating-point operand register.
2. Multiplies the first floating-point operand with second floating-point operand.
3. Adds the negation of the floating-point destination register to the product
4. Places the result into the destination register.

The result of the multiply is not rounded before the addition.

The VFNMS instruction:

1. Multiplies the first floating-point operand with second floating-point operand.
2. Adds the negation of the floating-point value in the destination register to the product.
3. Places the result in the destination register.

The result of the multiply is not rounded before the addition.

### Restrictions

There are no restrictions.

### Condition Flags

These instructions do not change the flags.

## 12.6.11.10 VLDM

Floating-point Load Multiple

Syntax

`VLDM{mode}{cond}{.size} Rn{!}, list`

where:

mode	is the addressing mode: - <i>IA</i> Increment After. The consecutive addresses start at the address specified in <i>Rn</i> . - <i>DB</i> Decrement Before. The consecutive addresses end just before the address specified in <i>Rn</i> .
cond	is an optional condition code, see " <a href="#">Conditional Execution</a> " .
size	is an optional data size specifier.
Rn	is the base register. The SP can be used
!	is the command to the instruction to write a modified value back to <i>Rn</i> . This is required if mode == DB, and is optional if mode == IA.
list	is the list of extension registers to be loaded, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction loads:

- Multiple extension registers from consecutive memory locations using an address from an ARM core register as the base address.

Restrictions

The restrictions are:

- If *size* is present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.
- For the base address, the SP can be used.  
In the ARM instruction set, if *!* is not specified the PC can be used.
- list* must contain at least one register. If it contains doubleword registers, it must not contain more than 16 registers.
- If using the *Decrement Before addressing* mode, the write back flag, *!*, must be appended to the base register specification.

Condition Flags

These instructions do not change the flags.

## 12.6.11.11 VLDR

Loads a single extension register from memory

### Syntax

```
VLDR{cond}{.64} Dd, [Rn{#imm}]  
VLDR{cond}{.64} Dd, label  
VLDR{cond}{.64} Dd, [PC, #imm]  
VLDR{cond}{.32} Sd, [Rn {, #imm}]  
VLDR{cond}{.32} Sd, label  
VLDR{cond}{.32} Sd, [PC, #imm]
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .
- 64, 32 are the optional data size specifiers.
- Dd is the destination register for a doubleword load.
- Sd is the destination register for a singleword load.
- Rn is the base register. The SP can be used.
- imm is the + or - immediate offset used to form the address.  
Permitted address values are multiples of 4 in the range 0 to 1020.
- label is the label of the literal data item to be loaded.

### Operation

This instruction:

- Loads a single extension register from memory, using a base address from an ARM core register, with an optional offset.

### Restrictions

There are no restrictions.

### Condition Flags

These instructions do not change the flags.

### 12.6.11.12 VLMA, VLMS

Multiplies two floating-point values, and accumulates or subtracts the results.

#### Syntax

```
VLMA{cond}.F32 Sd, Sn, Sm  
VLMS{cond}.F32 Sd, Sn, Sm
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Sd is the destination floating-point value.  
Sn, Sm are the operand floating-point values.

#### Operation

The floating-point Multiply Accumulate instruction:

1. Multiplies two floating-point values.
2. Adds the results to the destination floating-point value.

The floating-point Multiply Subtract instruction:

1. Multiplies two floating-point values.
2. Subtracts the products from the destination floating-point value.
3. Places the results in the destination register.

#### Restrictions

There are no restrictions.

#### Condition Flags

These instructions do not change the flags.

### 12.6.11.13 VMOV Immediate

Move floating-point Immediate

#### Syntax

```
VMOV{cond}.F32 Sd, #imm
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Sd is the branch destination.  
imm is a floating-point constant.

#### Operation

This instruction copies a constant value to a floating-point register.

#### Restrictions

There are no restrictions.

#### Condition Flags

These instructions do not change the flags.

#### 12.6.11.14 VMOV Register

Copies the contents of one register to another.

##### Syntax

```
VMOV{cond}.F64 Dd, Dm  
VMOV{cond}.F32 Sd, Sm
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Dd is the destination register, for a doubleword operation.  
Dm is the source register, for a doubleword operation.  
Sd is the destination register, for a singleword operation.  
Sm is the source register, for a singleword operation.

##### Operation

This instruction copies the contents of one floating-point register to another.

##### Restrictions

There are no restrictions

##### Condition Flags

These instructions do not change the flags.

#### 12.6.11.15 VMOV Scalar to ARM Core Register

Transfers one word of a doubleword floating-point register to an ARM core register.

##### Syntax

```
VMOV{cond} Rt, Dn[x]
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Rt is the destination ARM core register.  
Dn is the 64-bit doubleword register.  
x Specifies which half of the doubleword register to use:  
- If x is 0, use lower half of doubleword register  
- If x is 1, use upper half of doubleword register.

##### Operation

This instruction transfers:

- One word from the upper or lower half of a doubleword floating-point register to an ARM core register.

##### Restrictions

Rt cannot be PC or SP.

##### Condition Flags

These instructions do not change the flags.

### 12.6.11.16 VMOV ARM Core Register to Single Precision

Transfers a single-precision register to and from an ARM core register.

#### Syntax

```
VMOV{cond} Sn, Rt  
VMOV{cond} Rt, Sn
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Sn is the single-precision floating-point register.  
Rt is the ARM core register.

#### Operation

This instruction transfers:

- The contents of a single-precision register to an ARM core register.
- The contents of an ARM core register to a single-precision register.

#### Restrictions

Rt cannot be PC or SP.

#### Condition Flags

These instructions do not change the flags.

### 12.6.11.17 VMOV Two ARM Core Registers to Two Single Precision

Transfers two consecutively numbered single-precision registers to and from two ARM core registers.

#### Syntax

```
VMOV{cond} Sm, Sm1, Rt, Rt2  
VMOV{cond} Rt, Rt2, Sm, Sm
```

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Sm is the first single-precision register.  
Sm1 is the second single-precision register.  
This is the next single-precision register after Sm.  
Rt is the ARM core register that Sm is transferred to or from.  
Rt2 is the The ARM core register that Sm1 is transferred to or from.

#### Operation

This instruction transfers:

- The contents of two consecutively numbered single-precision registers to two ARM core registers.
- The contents of two ARM core registers to a pair of single-precision registers.

#### Restrictions

- The restrictions are:
- The floating-point registers must be contiguous, one after the other.
- The ARM core registers do not have to be contiguous.
- Rt cannot be PC or SP.

#### Condition Flags

These instructions do not change the flags.

### 12.6.11.18 VMOV ARM Core Register to Scalar

Transfers one word to a floating-point register from an ARM core register.

#### Syntax

`VMOV{cond}{.32} Dd[x], Rt`

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
32 is an optional data size specifier.  
 $Dd[x]$  is the destination, where [x] defines which half of the doubleword is transferred, as follows:  
If  $x$  is 0, the lower half is extracted  
If  $x$  is 1, the upper half is extracted.  
Rt is the source ARM core register.

#### Operation

This instruction transfers one word to the upper or lower half of a doubleword floating-point register from an ARM core register.

#### Restrictions

$Rt$  cannot be PC or SP.

#### Condition Flags

These instructions do not change the flags.

### 12.6.11.19 VMRS

Move to ARM Core register from floating-point System Register.

#### Syntax

`VMRS{cond} Rt, FPSCR`  
`VMRS{cond} APSR_nzcv, FPSCR`

where:

- cond is an optional condition code, see “[Conditional Execution](#)” .  
Rt is the destination ARM core register. This register can be R0-R14.  
APSR\_nzcv Transfer floating-point flags to the APSR flags.

#### Operation

This instruction performs one of the following actions:

- Copies the value of the FPSCR to a general-purpose register.
- Copies the value of the FPSCR flag bits to the APSR N, Z, C, and V flags.

#### Restrictions

$Rt$  cannot be PC or SP.

#### Condition Flags

These instructions optionally change the flags: N, Z, C, V

### 12.6.11.20 VMSR

Move to floating-point System Register from ARM Core register.

#### Syntax

VMSR{*cond*} FPSCR, *Rt*

where:

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Rt* is the general-purpose register to be transferred to the FPSCR.

#### Operation

This instruction moves the value of a general-purpose register to the FPSCR. See “[Floating-point Status Control Register](#)” for more information.

#### Restrictions

The restrictions are:

- *Rt* cannot be PC or SP.

#### Condition Flags

This instruction updates the FPSCR.

### 12.6.11.21 VMUL

Floating-point Multiply.

#### Syntax

VMUL{*cond*} .F32 {*Sd*,} *Sn*, *Sm*

where:

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*Sd* is the destination floating-point value.

*Sn*, *Sm* are the operand floating-point values.

#### Operation

This instruction:

1. Multiplies two floating-point values.
2. Places the results in the destination register.

#### Restrictions

There are no restrictions.

#### Condition Flags

These instructions do not change the flags.

## 12.6.11.22 VNEG

Floating-point Negate.

Syntax

VNEG{cond}.F32 Sd, Sm

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Sd is the destination floating-point value.

Sm is the operand floating-point value.

Operation

This instruction:

1. Negates a floating-point value.
2. Places the results in a second floating-point register.

The floating-point instruction inverts the sign bit.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

## 12.6.11.23 VNMLA, VNMLS, VNMUL

Floating-point multiply with negation followed by add or subtract.

Syntax

VNMLA{cond}.F32 Sd, Sn, Sm  
VNMLS{cond}.F32 Sd, Sn, Sm  
VNMUL{cond}.F32 {Sd, } Sn, Sm

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Sd is the destination floating-point register.

Sn, Sm are the operand floating-point registers.

Operation

The VNMLA instruction:

1. Multiplies two floating-point register values.
2. Adds the negation of the floating-point value in the destination register to the negation of the product.
3. Writes the result back to the destination register.

The VNMLS instruction:

1. Multiplies two floating-point register values.
2. Adds the negation of the floating-point value in the destination register to the product.
3. Writes the result back to the destination register.

The VNMUL instruction:

1. Multiplies together two floating-point register values.
2. Writes the negation of the result to the destination register.

Restrictions

There are no restrictions.

## Condition Flags

These instructions do not change the flags.

### 12.6.11.24 VPOP

Floating-point extension register Pop.

#### Syntax

```
VPOP{cond}{.size} list
```

where:

- |      |  |
|------|--|
| cond | is an optional condition code, see “ <a href="#">Conditional Execution</a> ” .   |
| size | is an optional data size specifier.<br>If present, it must be equal to the size in bits, 32 or 64, of the registers in <i>list</i> .                                     |
| list | is the list of extension registers to be loaded, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets. |

#### Operation

This instruction loads multiple consecutive extension registers from the stack.

#### Restrictions

The list must contain at least one register, and not more than sixteen registers.

#### Condition Flags

These instructions do not change the flags.

### 12.6.11.25 VPUSH

Floating-point extension register Push.

#### Syntax

```
VPUSH{cond}{.size} list
```

where:

- |      |  |
|------|--|
| cond | is an optional condition code, see “ <a href="#">Conditional Execution</a> ” .   |
| size | is an optional data size specifier.<br>If present, it must be equal to the size in bits, 32 or 64, of the registers in <i>list</i> .                                       |
| list | is a list of the extension registers to be stored, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets. |

#### Operation

This instruction:

- Stores multiple consecutive extension registers to the stack.

#### Restrictions

The restrictions are:

- list must contain at least one register, and not more than sixteen.

#### Condition Flags

These instructions do not change the flags.

### 12.6.11.26 VSQRT

Floating-point Square Root.

Syntax

`VSQRT{cond}.F32 Sd, Sm`

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Sd is the destination floating-point value.

Sm is the operand floating-point value.

Operation

This instruction:

- Calculates the square root of the value in a floating-point register.
- Writes the result to another floating-point register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

### 12.6.11.27 VSTM

Floating-point Store Multiple.

Syntax

`VSTM{mode}{cond}{.size} Rn{!}, list`

where:

mode is the addressing mode:

- *IA* Increment After. The consecutive addresses start at the address specified in *Rn*.

This is the default and can be omitted.

- *DB* Decrement Before. The consecutive addresses end just before the address specified in *Rn*.

cond is an optional condition code, see “[Conditional Execution](#)” .

size is an optional data size specifier.

If present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.

*Rn* is the base register. The SP can be used

! is the function that causes the instruction to write a modified value back to *Rn*. Required if mode == DB.

list is a list of the extension registers to be stored, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction:

- Stores multiple extension registers to consecutive memory locations using a base address from an ARM core register.

Restrictions

The restrictions are:

- list must contain at least one register.  
If it contains doubleword registers it must not contain more than 16 registers.
- Use of the PC as *Rn* is deprecated.

#### Condition Flags

These instructions do not change the flags.

### 12.6.11.28 VSTR

Floating-point Store.

#### Syntax

```
VSTR{cond}{.32} Sd, [Rn{, #imm}]
VSTR{cond}{.64} Dd, [Rn{, #imm}]
```

where

cond	is an optional condition code, see “ <a href="#">Conditional Execution</a> ” .
32, 64	are the optional data size specifiers.
Sd	is the source register for a singleword store.
Dd	is the source register for a doubleword store.
Rn	is the base register. The SP can be used.
imm	is the + or - immediate offset used to form the address. Values are multiples of 4 in the range 0-1020. <i>imm</i> can be omitted, meaning an offset of +0.

#### Operation

This instruction:

- Stores a single extension register to memory, using an address from an ARM core register, with an optional offset, defined in *imm*.

#### Restrictions

The restrictions are:

- The use of PC for *Rn* is deprecated.

#### Condition Flags

These instructions do not change the flags.

## 12.6.11.29 VSUB

Floating-point Subtract.

### Syntax

```
VSUB{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Sd is the destination floating-point value.

Sn, Sm are the operand floating-point value.

### Operation

This instruction:

1. Subtracts one floating-point value from another floating-point value.
2. Places the results in the destination floating-point register.

### Restrictions

There are no restrictions.

### Condition Flags

These instructions do not change the flags.

## 12.6.12 Miscellaneous Instructions

The table below shows the remaining Cortex-M4 instructions:

**Table 12-28. Miscellaneous Instructions**

Mnemonic	Description
BKPT	Breakpoint
CPSID	Change Processor State, Disable Interrupts
CPSIE	Change Processor State, Enable Interrupts
DMB	Data Memory Barrier
DSB	Data Synchronization Barrier
ISB	Instruction Synchronization Barrier
MRS	Move from special register to register
MSR	Move from register to special register
NOP	No Operation
SEV	Send Event
SVC	Supervisor Call
WFE	Wait For Event
WFI	Wait For Interrupt

### 12.6.12.1 BKPT

Breakpoint.

Syntax

`BKPT #imm`

where:

imm is an expression evaluating to an integer in the range 0-255 (8-bit value).

Operation

The BKPT instruction causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached.

*imm* is ignored by the processor. If required, a debugger can use it to store additional information about the breakpoint.

The BKPT instruction can be placed inside an IT block, but it executes unconditionally, unaffected by the condition specified by the IT instruction.

Condition Flags

This instruction does not change the flags.

Examples

```
BKPT 0xAB      ; Breakpoint with immediate value set to 0xAB (debugger can  
; extract the immediate value by locating it using the PC)
```

Note: ARM does not recommend the use of the BKPT instruction with an immediate value set to 0xAB for any purpose other than Semi-hosting.

### 12.6.12.2 CPS

Change Processor State.

Syntax

`CPSeffect iflags`

where:

*effect*      is one of:

IE      Clears the special purpose register.

ID      Sets the special purpose register.

*iflags*      is a sequence of one or more flags:

i      Set or clear PRIMASK.

f      Set or clear FAULTMASK.

Operation

CPS changes the PRIMASK and FAULTMASK special register values. See “[Exception Mask Registers](#)” for more information about these registers.

Restrictions

The restrictions are:

- Use CPS only from privileged software, it has no effect if used in unprivileged software
- CPS cannot be conditional and so must not be used inside an IT block.

Condition Flags

This instruction does not change the condition flags.

Examples

```
CPSID i ; Disable interrupts and configurable fault handlers (set PRIMASK)
CPSID f ; Disable interrupts and all fault handlers (set FAULTMASK)
CPSIE i ; Enable interrupts and configurable fault handlers (clear PRIMASK)
CPSIE f ; Enable interrupts and fault handlers (clear FAULTMASK)
```

### 12.6.12.3 DMB

Data Memory Barrier.

Syntax

`DMB{cond}`

where:

*cond*      is an optional condition code, see “[Conditional Execution](#)” .

Operation

DMB acts as a data memory barrier. It ensures that all explicit memory accesses that appear, in program order, before the DMB instruction are completed before any explicit memory accesses that appear, in program order, after the DMB instruction. DMB does not affect the ordering or execution of instructions that do not access memory.

Condition Flags

This instruction does not change the flags.

Examples

```
DMB ; Data Memory Barrier
```

#### 12.6.12.4 DSB

Data Synchronization Barrier.

Syntax

DSB{*cond*}

where:

*cond* is an optional condition code, see “[Conditional Execution](#)” .

Operation

DSB acts as a special data synchronization memory barrier. Instructions that come after the DSB, in program order, do not execute until the DSB instruction completes. The DSB instruction completes when all explicit memory accesses before it complete.

Condition Flags

This instruction does not change the flags.

Examples

```
DSB ; Data Synchronisation Barrier
```

#### 12.6.12.5 ISB

Instruction Synchronization Barrier.

Syntax

ISB{*cond*}

where:

*cond* is an optional condition code, see “[Conditional Execution](#)” .

Operation

ISB acts as an instruction synchronization barrier. It flushes the pipeline of the processor, so that all instructions following the ISB are fetched from cache or memory again, after the ISB instruction has been completed.

Condition Flags

This instruction does not change the flags.

Examples

```
ISB ; Instruction Synchronisation Barrier
```

## 12.6.12.6 MRS

Move the contents of a special register to a general-purpose register.

Syntax

```
MRS{cond} Rd, spec_reg
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Rd is the destination register.

spec\_reg can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI\_MAX, FAULTMASK, or CONTROL.

Operation

Use MRS in combination with MSR as part of a read-modify-write sequence for updating a PSR, for example to clear the Q flag.

In process swap code, the programmers model state of the process being swapped out must be saved, including relevant PSR contents. Similarly, the state of the process being swapped in must also be restored. These operations use MRS in the state-saving instruction sequence and MSR in the state-restoring instruction sequence.

Note: BASEPRI\_MAX is an alias of BASEPRI when used with the MRS instruction.

See “[MSR](#)” .

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

```
MRS R0, PRIMASK ; Read PRIMASK value and write it to R0
```

### 12.6.12.7 MSR

Move the contents of a general-purpose register into the specified special register.

#### Syntax

```
MSR{cond} spec_reg, Rn
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Rn is the source register.

spec\_reg can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI\_MAX, FAULTMASK, or CONTROL.

#### Operation

The register access operation in MSR depends on the privilege level. Unprivileged software can only access the APSR. See “[Application Program Status Register](#)” . Privileged software can access all special registers.

In unprivileged software writes to unallocated or execution state bits in the PSR are ignored.

Note: When the user writes to BASEPRI\_MAX, the instruction writes to BASEPRI only if either:

Rn is non-zero and the current BASEPRI value is 0

Rn is non-zero and less than the current BASEPRI value.

See “[MRS](#)” .

#### Restrictions

Rn must not be SP and must not be PC.

#### Condition Flags

This instruction updates the flags explicitly based on the value in Rn.

#### Examples

```
MSR CONTROL, R1 ; Read R1 value and write it to the CONTROL register
```

### 12.6.12.8 NOP

No Operation.

#### Syntax

```
NOP{cond}
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

#### Operation

NOP does nothing. NOP is not necessarily a time-consuming NOP. The processor might remove it from the pipeline before it reaches the execution stage.

Use NOP for padding, for example to place the following instruction on a 64-bit boundary.

#### Condition Flags

This instruction does not change the flags.

#### Examples

```
NOP ; No operation
```

### 12.6.12.9 SEV

Send Event.

Syntax

SEV{*cond*}

where:

*cond* is an optional condition code, see “[Conditional Execution](#)” .

Operation

SEV is a hint instruction that causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register to 1, see “[Power Management](#)” .

Condition Flags

This instruction does not change the flags.

Examples

```
SEV ; Send Event
```

### 12.6.12.10 SVC

Supervisor Call.

Syntax

SVC{*cond*} #*imm*

where:

*cond* is an optional condition code, see “[Conditional Execution](#)” .

*imm* is an expression evaluating to an integer in the range 0-255 (8-bit value).

Operation

The SVC instruction causes the SVC exception.

*imm* is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

Condition Flags

This instruction does not change the flags.

Examples

```
SVC 0x32 ; Supervisor Call (SVC handler can extract the immediate value  
; by locating it via the stacked PC)
```

### 12.6.12.11 WFE

Wait For Event.

Syntax

```
WFE{cond}
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Operation

WFE is a hint instruction.

If the event register is 0, WFE suspends execution until one of the following events occurs:

- An exception, unless masked by the exception mask registers or the current priority level
- An exception enters the Pending state, if SEVONPEND in the System Control Register is set
- A Debug Entry request, if Debug is enabled
- An event signaled by a peripheral or another processor in a multiprocessor system using the SEV instruction.

If the event register is 1, WFE clears it to 0 and returns immediately.

For more information, see “[Power Management](#)” .

Condition Flags

This instruction does not change the flags.

Examples

```
WFE ; Wait for event
```

### 12.6.12.12 WFI

Wait for Interrupt.

Syntax

```
WFI{cond}
```

where:

cond is an optional condition code, see “[Conditional Execution](#)” .

Operation

WFI is a hint instruction that suspends execution until one of the following events occurs:

- An exception
- A Debug Entry request, regardless of whether Debug is enabled.

Condition Flags

This instruction does not change the flags.

Examples

```
WFI ; Wait for interrupt
```

## 12.7 Cortex-M4 Core Peripherals

### 12.7.1 Peripherals

- Nested Vectored Interrupt Controller (NVIC)  
The Nested Vectored Interrupt Controller (NVIC) is an embedded interrupt controller that supports low latency interrupt processing. See [Section 12.8 "Nested Vectored Interrupt Controller \(NVIC\)"](#)
- System Control Block (SCB)  
The System Control Block (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions. See [Section 12.9 "System Control Block \(SCB\)"](#)
- System Timer (SysTick)  
The System Timer, SysTick, is a 24-bit count-down timer. Use this as a Real-time Operating System (RTOS) tick timer or as a simple counter. See [Section 12.10 "System Timer \(SysTick\)"](#)
- Memory Protection Unit (MPU)  
The Memory Protection Unit (MPU) improves system reliability by defining the memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region. See [Section 12.11 "Memory Protection Unit \(MPU\)"](#)
- Floating-point Unit (FPU)  
The Floating-point Unit (FPU) provides IEEE754-compliant operations on single-precision, 32-bit, floating-point values. See [Section 12.12 "Floating Point Unit \(FPU\)"](#)

### 12.7.2 Address Map

The address map of the *Private peripheral bus* (PPB) is:

**Table 12-29. Core Peripheral Register Regions**

Address	Core Peripheral
0xE000E008-0xE000E00F	System Control Block
0xE000E010-0xE000E01F	System Timer
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller
0xE000ED00-0xE000ED3F	System control block
0xE000ED90-0xE000EDB8	Memory Protection Unit
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller
0xE000EF30-0xE000EF44	Floating-point Unit

In register descriptions:

- The *required privilege* gives the privilege level required to access the register, as follows:
  - Privileged: Only privileged software can access the register.
  - Unprivileged: Both unprivileged and privileged software can access the register.

## 12.8 Nested Vectored Interrupt Controller (NVIC)

This section describes the NVIC and the registers it uses. The NVIC supports:

- 1 to 48 interrupts.
- A programmable priority level of 0-15 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external *Non-maskable interrupt* (NMI)

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling.

### 12.8.1 Level-sensitive Interrupts

The processor supports level-sensitive interrupts. A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically, this happens because the ISR accesses the peripheral, causing it to clear the interrupt request.

When the processor enters the ISR, it automatically removes the pending state from the interrupt (see “[Hardware and Software Control of Interrupts](#)”). For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer requires servicing.

#### 12.8.1.1 Hardware and Software Control of Interrupts

The Cortex-M4 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- The NVIC detects that the interrupt signal is HIGH and the interrupt is not active
- The NVIC detects a rising edge on the interrupt signal
- A software writes to the corresponding interrupt set-pending register bit, see “[Interrupt Set-pending Registers](#)”, or to the NVIC\_STIR register to make an interrupt pending, see “[Software Trigger Interrupt Register](#)”.

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit.  
For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.

### 12.8.2 NVIC Design Hints and Tips

Ensure that the software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers. See the individual register descriptions for the supported access sizes.

A interrupt can enter a pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt.

Before programming SCB\_VTOR to relocate the vector table, ensure that the vector table entries of the new vector table are set up for fault handlers, NMI and all enabled exception like interrupts. For more information, see the “[Vector Table Offset Register](#)”.

### 12.8.2.1 NVIC Programming Hints

The software uses the CPSIE I and CPSID I instructions to enable and disable the interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including:

**Table 12-30. CMSIS Functions for NVIC Control**

CMSIS Interrupt Control Function	Description
void NVIC_SetPriorityGrouping(uint32_t priority_grouping)	Set the priority grouping
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true (IRQ-Number) if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
uint32_t NVIC_GetActive (IRQn_t IRQn)	Return the IRQ number of the active interrupt
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

The input parameter IRQn is the IRQ number. For more information about these functions, see the CMSIS documentation.

To improve software efficiency, the CMSIS simplifies the NVIC register presentation. In the CMSIS:

- The Set-enable, Clear-enable, Set-pending, Clear-pending and Active Bit registers map to arrays of 32-bit integers, so that:
  - The array ISER[0] corresponds to the registers ISER0
  - The array ICER[0] corresponds to the registers ICER0
  - The array ISPR[0] corresponds to the registers ISPR0
  - The array ICPR[0] corresponds to the registers ICPR0
  - The array IABR[0] corresponds to the registers IABR0
- The 4-bit fields of the Interrupt Priority Registers map to an array of 4-bit integers, so that the array IP[0] to IP[47] corresponds to the registers IPR0-IPR12, and the array entry IP[n] holds the interrupt priority for interrupt n.

The CMSIS provides thread-safe code that gives atomic access to the Interrupt Priority Registers. [Table 12-31](#) shows how the interrupts, or IRQ numbers, map onto the interrupt registers and corresponding CMSIS variables that have one bit per interrupt.

**Table 12-31. Mapping of Interrupts to the Interrupt Variables**

Interrupts	CMSIS Array Elements <sup>(1)</sup>				
	Set-enable	Clear-enable	Set-pending	Clear-pending	Active Bit
0-47	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]

Note: 1. Each array element corresponds to a single NVIC register, for example the ICER[0] element corresponds to the ICER0 register.

### 12.8.3 Nested Vectored Interrupt Controller (NVIC) User Interface

**Table 12-32. Nested Vectored Interrupt Controller (NVIC) Register Mapping**

Offset	Register	Name	Access	Reset
0xE000E100	Interrupt Set-enable Register 0	NVIC_ISER0	Read-write	0x00000000
...	...	...	...	...
0xE000E11C	Interrupt Set-enable Register 7	NVIC_ISER7	Read-write	0x00000000
0xE000E180	Interrupt Clear-enable Register0	NVIC_ICER0	Read-write	0x00000000
...	...	...	...	...
0xE000E19C	Interrupt Clear-enable Register 7	NVIC_ICER7	Read-write	0x00000000
0xE000E200	Interrupt Set-pending Register 0	NVIC_ISPR0	Read-write	0x00000000
...	...	...	...	...
0xE000E21C	Interrupt Set-pending Register 7	NVIC_ISPR7	Read-write	0x00000000
0xE000E280	Interrupt Clear-pending Register 0	NVIC_ICPR0	Read-write	0x00000000
...	...	...	...	...
0xE000E29C	Interrupt Clear-pending Register 7	NVIC_ICPR7	Read-write	0x00000000
0xE000E300	Interrupt Active Bit Register 0	NVIC_IABR0	Read-write	0x00000000
...	...	...	...	...
0xE000E31C	Interrupt Active Bit Register 7	NVIC_IABR7	Read-write	0x00000000
0xE000E400	Interrupt Priority Register 0	NVIC_IPR0	Read-write	0x00000000
...	...	...	...	...
0xE000E42C	Interrupt Priority Register 12	NVIC_IPR12	Read-write	0x00000000
0xE000EF00	Software Trigger Interrupt Register	NVIC_STIR	Write-only	0x00000000

### 12.8.3.1 Interrupt Set-enable Registers

**Name:** NVIC\_ISERx [x=0..7]

**Access:** Read-write

**Reset:** 0x0000000000

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

These registers enable interrupts and show which interrupts are enabled.

- **SETENA: Interrupt Set-enable**

Write:

0: No effect.

1: Enables the interrupt.

Read:

0: Interrupt disabled.

1: Interrupt enabled.

Notes: 1. If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority.

2. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, the NVIC never activates the interrupt, regardless of its priority.

### 12.8.3.2 Interrupt Clear-enable Registers

**Name:** NVIC\_ICERx [x=0..7]

**Access:** Read-write

**Reset:** 0x0000000000

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

These registers disable interrupts, and show which interrupts are enabled.

- **CLRENA: Interrupt Clear-enable**

Write:

0: No effect.

1: Disables the interrupt.

Read:

0: Interrupt disabled.

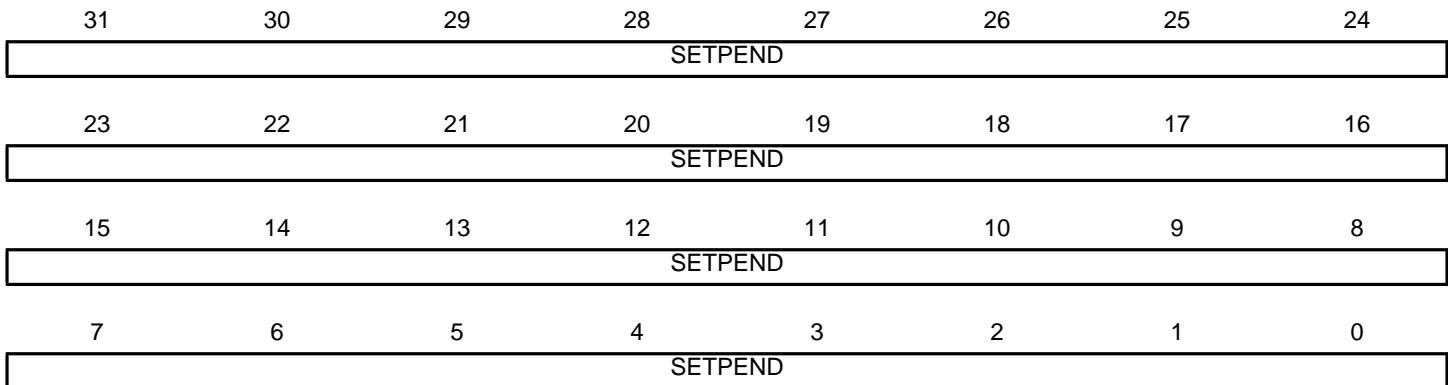
1: Interrupt enabled.

### 12.8.3.3 Interrupt Set-pending Registers

**Name:** NVIC\_ISPRx [x=0..7]

**Access:** Read-write

**Reset:** 0x0000000000



These registers force interrupts into the pending state, and show which interrupts are pending.

- **SETPEND: Interrupt Set-pending**

Write:

0: No effect.

1: Changes the interrupt state to pending.

Read:

0: Interrupt is not pending.

1: Interrupt is pending.

Notes: 1. Writing 1 to an ISPR bit corresponding to an interrupt that is pending has no effect.

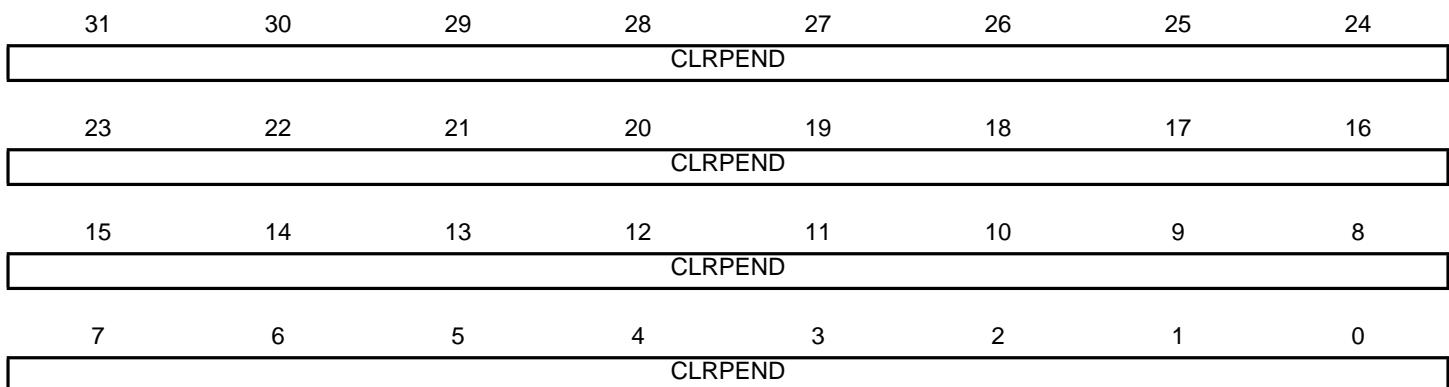
2. Writing 1 to an ISPR bit corresponding to a disabled interrupt sets the state of that interrupt to pending.

#### 12.8.3.4 Interrupt Clear-pending Registers

**Name:** NVIC\_ICPRx [x=0..7]

**Access:** Read-write

**Reset:** 0x0000000000



These registers remove the pending state from interrupts, and show which interrupts are pending.

- **CLRPEND: Interrupt Clear-pending**

Write:

0: No effect.

1: Removes the pending state from an interrupt.

Read:

0: Interrupt is not pending.

1: Interrupt is pending.

Note: Writing 1 to an ICPR bit does not affect the active state of the corresponding interrupt.

### 12.8.3.5 Interrupt Active Bit Registers

**Name:** NVIC\_IABRx [x=0..7]

**Access:** Read-write

**Reset:** 0x0000000000

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

These registers indicate which interrupts are active.

- **ACTIVE: Interrupt Active Flags**

0: Interrupt is not active.

1: Interrupt is active.

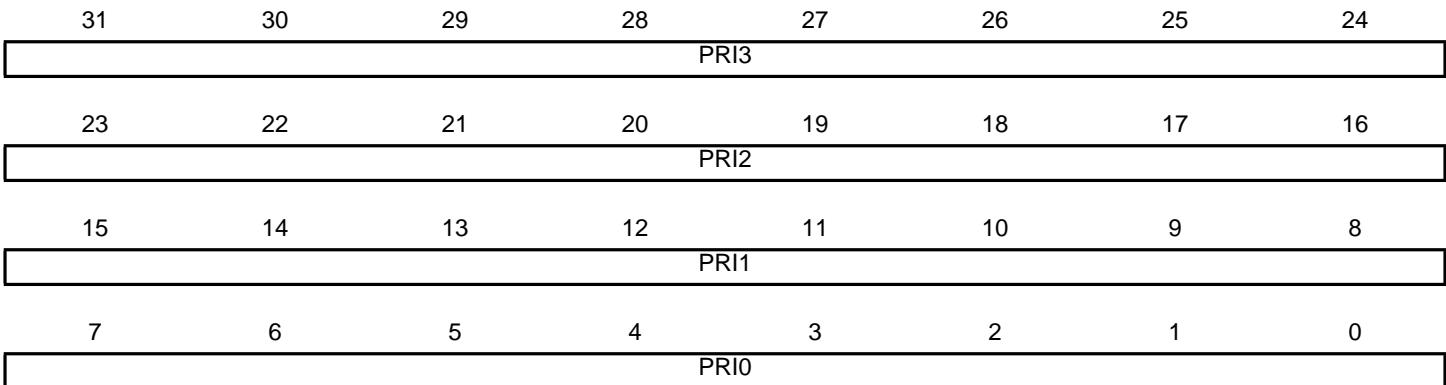
Note: A bit reads as one if the status of the corresponding interrupt is active, or active and pending.

### 12.8.3.6 Interrupt Priority Registers

**Name:** NVIC\_IPRx [x=0..12]

**Access:** Read-write

**Reset:** 0x000000000



The NVIC\_IPR0-NVIC\_IPR12 registers provide a 4-bit priority field for each interrupt. These registers are byte-accessible. Each register holds four priority fields, that map up to four elements in the CMSIS interrupt priority array IP[0] to IP[47]

- **PRI3: Priority (4m+3)**

Priority, Byte Offset 3, refers to register bits [31:24].

- **PRI2: Priority (4m+2)**

Priority, Byte Offset 2, refers to register bits [23:16].

- **PRI1: Priority (4m+1)**

Priority, Byte Offset 1, refers to register bits [15:8].

- **PRI0: Priority (4m)**

Priority, Byte Offset 0, refers to register bits [7:0].

Notes:

1. Each priority field holds a priority value, 0-15. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:4] of each field; bits[3:0] read as zero and ignore writes.
2. for more information about the IP[0] to IP[47] interrupt priority array, that provides the software view of the interrupt priorities, see [Table 12-30, "CMSIS Functions for NVIC Control"](#).
3. The corresponding IPR number  $n$  is given by  $n = m \text{ DIV } 4$ .
4. The byte offset of the required Priority field in this register is  $m \text{ MOD } 4$ .

#### 12.8.3.7 Software Trigger Interrupt Register

**Name:** NVIC\_STIR

**Access:** Write-only

**Reset:** 0x0000000000

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

Write to this register to generate an interrupt from the software.

- **INTID: Interrupt ID**

Interrupt ID of the interrupt to trigger, in the range 0-239. For example, a value of 0x03 specifies interrupt IRQ3.

## 12.9 System Control Block (SCB)

The System Control Block (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions.

Ensure that the software uses aligned accesses of the correct size to access the system control block registers:

- Except for the SCB\_CFSR and SCB\_SHPR1-SCB\_SHPR3 registers, it must use aligned word accesses
- For the SCB\_CFSR and SCB\_SHPR1-SCB\_SHPR3 registers, it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to system control block registers.

In a fault handler, to determine the true faulting address:

1. Read and save the MMFAR or SCB\_BFAR value.
2. Read the MMARVALID bit in the MMFSR subregister, or the BFARVALID bit in the BFSR subregister. The SCB\_MMFAR or SCB\_BFAR address is valid only if this bit is 1.

The software must follow this sequence because another higher priority exception might change the SCB\_MMFAR or SCB\_BFAR value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the SCB\_MMFAR or SCB\_BFAR value.

## 12.9.1 System Control Block (SCB) User Interface

**Table 12-33. System Control Block (SCB) Register Mapping**

Offset	Register	Name	Access	Reset
0xE000E008	Auxiliary Control Register	SCB_ACTLR	Read-write	0x00000000
0xE000ED00	CPUID Base Register	SCB_CPUID	Read-only	0x410FC240
0xE000ED04	Interrupt Control and State Register	SCB_ICSR	Read-write <sup>(1)</sup>	0x00000000
0xE000ED08	Vector Table Offset Register	SCB_VTOR	Read-write	0x00000000
0xE000ED0C	Application Interrupt and Reset Control Register	SCB_AIRCR	Read-write	0xFA050000
0xE000ED10	System Control Register	SCB_SCR	Read-write	0x00000000
0xE000ED14	Configuration and Control Register	SCB_CCR	Read-write	0x00000200
0xE000ED18	System Handler Priority Register 1	SCB_SHPR1	Read-write	0x00000000
0xE000ED1C	System Handler Priority Register 2	SCB_SHPR2	Read-write	0x00000000
0xE000ED20	System Handler Priority Register 3	SCB_SHPR3	Read-write	0x00000000
0xE000ED24	System Handler Control and State Register	SCB_SHCSR	Read-write	0x00000000
0xE000ED28	Configurable Fault Status Register	SCB_CFSR <sup>(2)</sup>	Read-write	0x00000000
0xE000ED2C	HardFault Status Register	SCB_HFSR	Read-write	0x00000000
0xE000ED34	MemManage Fault Address Register	SCB_MMFAR	Read-write	Unknown
0xE000ED38	BusFault Address Register	SCB_BFAR	Read-write	Unknown
0xE000ED3C	Auxiliary Fault Status Register	SCB_AFSR	Read-write	0x00000000

Notes:

1. See the register description for more information.

2. This register contains the subregisters: [“MMFSR: Memory Management Fault Status Subregister”](#) (0xE000ED28 - 8 bits), [“BFSR: Bus Fault Status Subregister”](#) (0xE000ED29 - 8 bits), [“UFSR: Usage Fault Status Subregister”](#) (0xE000ED2A - 16 bits).

### 12.9.1.1 Auxiliary Control Register

**Name:** SCB\_ACTLR

**Access:** Read-write

**Reset:** 0x0000000000

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

The SCB\_ACTLR register provides disable bits for the following processor functions:

- IT folding
- Write buffer use for accesses to the default memory map
- Interruption of multi-cycle instructions.

By default, this register is set to provide optimum performance from the Cortex-M4 processor, and does not normally require modification.

#### • **DISOOFP: Disable Out Of Order Floating Point**

Disables floating point instructions that complete out of order with respect to integer instructions.

#### • **DISFPCA: Disable FPCA**

Disables an automatic update of CONTROL.FPCA.

#### • **DISFOLD: Disable Folding**

When set to 1, disables the IT folding.

**Note:** In some situations, the processor can start executing the first instruction in an IT block while it is still executing the IT instruction. This behavior is called IT folding, and it improves the performance. However, IT folding can cause jitter in looping. If a task must avoid jitter, set the DISFOLD bit to 1 before executing the task, to disable the IT folding.

#### • **DISDEFWBUF: Disable Default Write Buffer**

When set to 1, it disables the write buffer use during default memory map accesses. This causes BusFault to be precise but decreases the performance, as any store to memory must complete before the processor can execute the next instruction.

This bit only affects write buffers implemented in the Cortex-M4 processor.

#### • **DISMCYCINT: Disable Multiple Cycle Interruption**

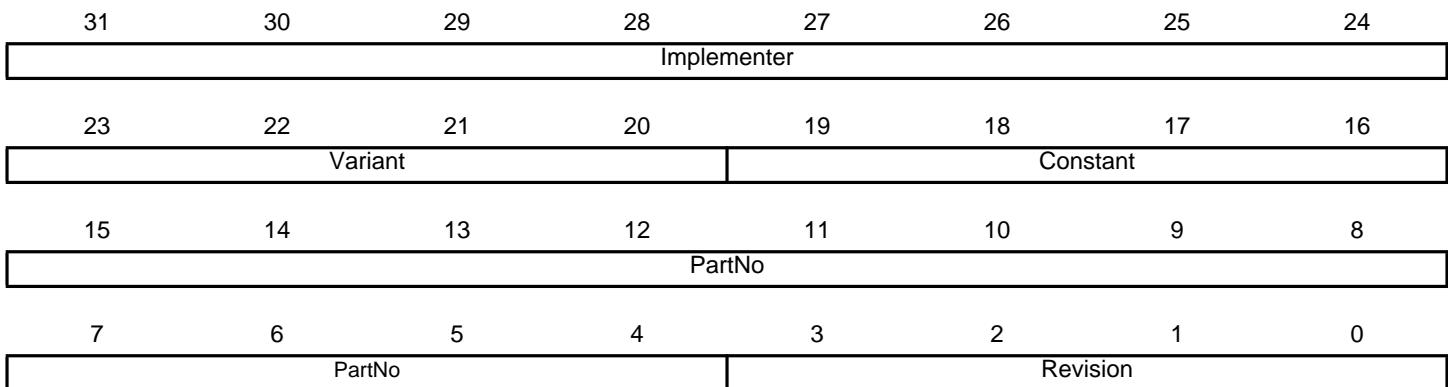
When set to 1, it disables the interruption of load multiple and store multiple instructions. This increases the interrupt latency of the processor, as any LDM or STM must complete before the processor can stack the current state and enter the interrupt handler.

### 12.9.1.2 CPUID Base Register

**Name:** SCB\_CPUID

**Access:** Read-write

**Reset:** 0x000000000



The SCB\_CPUID register contains the processor part number, version, and implementation information.

- **Implementer: Implementer Code**

0x41: ARM.

- **Variant: Variant Number**

It is the r value in the rnpn product revision identifier:

0x0: Revision 0.

- **Constant**

Reads as 0xF.

- **PartNo: Part Number of the Processor**

0xC24 = Cortex-M4.

- **Revision: Revision Number**

It is the p value in the rnpn product revision identifier:

0x0: Patch 0.

### 12.9.1.3 Interrupt Control and State Register

**Name:** SCB\_ICSR  
**Access:** Read-write  
**Reset:** 0x000000000

31	30	29	28	27	26	25	24
NMIPENDSET	–	PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	–	
23	22	21	20	19	18	17	16
–	ISR PENDING			VECTPENDING			
15	14	13	12	11	10	9	8
	VECTPENDING		RETTTOBASE		–		VECTACTIVE
7	6	5	4	3	2	1	0
			VECTACTIVE				

The SCB\_ICSR register provides a set-pending bit for the Non-Maskable Interrupt (NMI) exception, and set-pending and clear-pending bits for the PendSV and SysTick exceptions.

It indicates:

- The exception number of the exception being processed, and whether there are preempted active exceptions,
- The exception number of the highest priority pending exception, and whether any interrupts are pending.

#### • NMIPENDSET: NMI Set-pending

Write:

PendSV set-pending bit.

Write:

0: No effect.

1: Changes NMI exception state to pending.

Read:

0: NMI exception is not pending.

1: NMI exception is pending.

As NMI is the highest-priority exception, the processor normally enters the NMI exception handler as soon as it registers a write of 1 to this bit. Entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.

#### • PENDSVSET: PendSV Set-pending

Write:

0: No effect.

1: Changes PendSV exception state to pending.

Read:

0: PendSV exception is not pending.

1: PendSV exception is pending.

Writing 1 to this bit is the only way to set the PendSV exception state to pending.

#### • PENDSVCLR: PendSV Clear-pending

Write:

0: No effect.

1: Removes the pending state from the PendSV exception.

- **PENDSTSET: SysTick Exception Set-pending**

Write:

0: No effect.

1: Changes SysTick exception state to pending.

Read:

0: SysTick exception is not pending.

1: SysTick exception is pending.

- **PENDSTCLR: SysTick Exception Clear-pending**

Write:

0: No effect.

1: Removes the pending state from the SysTick exception.

This bit is Write-only. On a register read, its value is Unknown.

- **ISRPENDING: Interrupt Pending Flag (Excluding NMI and Faults)**

0: Interrupt not pending.

1: Interrupt pending.

- **VECTPENDING: Exception Number of the Highest Priority Pending Enabled Exception**

0: No pending exceptions.

Nonzero: The exception number of the highest priority pending enabled exception.

The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.

- **RETTOBASE: Preempted Active Exceptions Present or Not**

0: There are preempted active exceptions to execute.

1: There are no active exceptions, or the currently-executing exception is the only active exception.

- **VECTACTIVE: Active Exception Number Contained**

0: Thread mode.

Nonzero: The exception number of the currently active exception. The value is the same as IPSR bits [8:0]. See “[Interrupt Program Status Register](#)” .

Subtract 16 from this value to obtain the IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers, see “[Interrupt Program Status Register](#)” .

Note: When the user writes to the SCB\_ICSR register, the effect is unpredictable if:

- Writing 1 to the PENDSVSET bit and writing 1 to the PENDSVCLR bit
- Writing 1 to the PENDSTSET bit and writing 1 to the PENDSTCLR bit.

#### 12.9.1.4 Vector Table Offset Register

**Name:** SCB\_VTOR

**Access:** Read-write

**Reset:** 0x0000000000

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

The SCB\_VTOR register indicates the offset of the vector table base address from memory address 0x00000000.

- **TBLOFF: Vector Table Base Offset**

It contains bits [29:7] of the offset of the table base from the bottom of the memory map.

Bit [29] determines whether the vector table is in the code or SRAM memory region:

0: Code.

1: SRAM.

It is sometimes called the TBLBASE bit.

**Note:** When setting TBLOFF, the offset must be aligned to the number of exception entries in the vector table. Configure the next statement to give the information required for your implementation; the statement reminds the user of how to determine the alignment requirement. The minimum alignment is 32 words, enough for up to 16 interrupts. For more interrupts, adjust the alignment by rounding up to the next power of two. For example, if 21 interrupts are required, the alignment must be on a 64-word boundary because the required table size is 37 words, and the next power of two is 64.

Table alignment requirements mean that bits[6:0] of the table offset are always zero.

### 12.9.1.5 Application Interrupt and Reset Control Register

**Name:** SCB\_AIRCR

**Access:** Read-write

**Reset:** 0x000000000

31	30	29	28	27	26	25	24
VECTKEYSTAT/VECTKEY							
23	22	21	20	19	18	17	16
VECTKEYSTAT/VECTKEY							
15	14	13	12	11	10	9	8
ENDIANNES		—			PRIGROUP		
7	6	5	4	3	2	1	0
		—		SYSRESETREQ	VECTCLRACTIVE	VECTRESET	

The SCB\_AIRCR register provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. To write to this register, write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

- **VECTKEYSTAT: Register Key**

Read:

Reads as 0xFA05.

- **VECTKEY: Register Key**

Write:

Writes 0x5FA to VECTKEY, otherwise the write is ignored.

- **ENDIANNES: Data Endianness**

0: Little-endian.

1: Big-endian.

- **PRIGROUP: Interrupt Priority Grouping**

This field determines the split of group priority from subpriority. It shows the position of the binary point that splits the PRI\_n fields in the Interrupt Priority Registers into separate *group priority* and *subpriority* fields. The table below shows how the PRIGROUP value controls this split:

PRIGROUP	Interrupt Priority Level Value, PRI_N[7:0]			Number of	
	Binary Point <sup>(1)</sup>	Group Priority Bits	Subpriority Bits	Group Priorities	Subpriorities
0b000	bxxxxxx.y	[7:1]	None	128	2
0b001	bxxxxxx.yy	[7:2]	[4:0]	64	4
0b010	bxxxxx.yyy	[7:3]	[4:0]	32	8
0b011	bxxxx.yyyy	[7:4]	[4:0]	16	16
0b100	bxxx.yyyyy	[7:5]	[4:0]	8	32

PRIGROUP	Interrupt Priority Level Value, PRI_N[7:0]			Number of	
	Binary Point <sup>(1)</sup>	Group Priority Bits	Subpriority Bits	Group Priorities	Subpriorities
0b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
0b110	bx.yyyyyy	[7]	[6:0]	2	128
0b111	b.yyyyyy	None	[7:0]	1	256

Note: 1. PRI\_n[7:0] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit. Determining preemption of an exception uses only the group priority field.

- **SYSRESETREQ: System Reset Request**

0: No system reset request.

1: Asserts a signal to the outer system that requests a reset.

This is intended to force a large system reset of all major components except for debug. This bit reads as 0.

- **VECTCLRACTIVE**

Reserved for Debug use. This bit reads as 0. When writing to the register, write 0 to this bit, otherwise the behavior is unpredictable.

- **VECTRESET**

Reserved for Debug use. This bit reads as 0. When writing to the register, write 0 to this bit, otherwise the behavior is unpredictable.

### 12.9.1.6 System Control Register

**Name:** SCB\_SCR  
**Access:** Read-write  
**Reset:** 0x0000000000

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
	—		SEVONPEND	—	SLEEPDEEP	SLEEPONEXIT	—

- **SEVONPEND: Send Event on Pending Bit**

0: Only enabled interrupts or events can wake up the processor; disabled interrupts are excluded.

1: Enabled events and all interrupts, including disabled interrupts, can wake up the processor.

When an event or an interrupt enters the pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE.

The processor also wakes up on execution of an SEV instruction or an external event.

- **SLEEPDEEP: Sleep or Deep Sleep**

Controls whether the processor uses sleep or deep sleep as its low power mode:

0: Sleep.

1: Deep sleep.

- **SLEEPONEXIT: Sleep-on-exit**

Indicates sleep-on-exit when returning from the Handler mode to the Thread mode:

0: Do not sleep when returning to Thread mode.

1: Enter sleep, or deep sleep, on return from an ISR.

Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

### 12.9.1.7 Configuration and Control Register

**Name:** SCB\_CCR  
**Access:** Read-write  
**Reset:** 0x0000000000

31	30	29	28	27	26	25	24
				—			
23	22	21	20	19	18	17	16
				—			
15	14	13	12	11	10	9	8
			—			STKALIGN	BFHFNIGN
7	6	5	4	3	2	1	0
	—		DIV_0_TRP	UNALIGN_TRP	—	USERSETMPE ND	NONBASETHR DENA

The SCB\_CCR register controls the entry to the Thread mode and enables the handlers for NMI, hard fault and faults escalated by FAULTMASK to ignore BusFaults. It also enables the division by zero and unaligned access trapping, and the access to the NVIC\_STIR register by unprivileged software (see “[Software Trigger Interrupt Register](#)” ).

- **STKALIGN: Stack Alignment**

Indicates the stack alignment on exception entry:

0: 4-byte aligned.

1: 8-byte aligned.

On exception entry, the processor uses bit [9] of the stacked PSR to indicate the stack alignment. On return from the exception, it uses this stacked bit to restore the correct stack alignment.

- **BFHFNIGN: Bus Faults Ignored**

Enables handlers with priority -1 or -2 to ignore data bus faults caused by load and store instructions. This applies to the hard fault and FAULTMASK escalated handlers:

0: Data bus faults caused by load and store instructions cause a lock-up.

1: Handlers running at priority -1 and -2 ignore data bus faults caused by load and store instructions.

Set this bit to 1 only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them.

- **DIV\_0\_TRP: Division by Zero Trap**

Enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0:

0: Do not trap divide by 0.

1: Trap divide by 0.

When this bit is set to 0, a divide by zero returns a quotient of 0.

- **UNALIGN\_TRP: Unaligned Access Trap**

Enables unaligned access traps:

0: Do not trap unaligned halfword and word accesses.

1: Trap unaligned halfword and word accesses.

If this bit is set to 1, an unaligned access generates a usage fault.

Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of whether UNALIGN\_TRP is set to 1.

- **USERSETMPEND**

Enables unprivileged software access to the NVIC\_STIR register, see “[Software Trigger Interrupt Register](#)” :

0: Disable.

1: Enable.

- **NONEBASETHRDENA: Thread Mode Enable**

Indicates how the processor enters Thread mode:

0: The processor can enter the Thread mode only when no exception is active.

1: The processor can enter the Thread mode from any level under the control of an EXC\_RETURN value, see “[Exception Return](#)”

.

#### 12.9.1.8 System Handler Priority Registers

The SCB\_SHPR1-SCB\_SHPR3 registers set the priority level, 0 to 15 of the exception handlers that have configurable priority. They are byte-accessible.

The system fault handlers and the priority field and register for each handler are:

**Table 12-34. System Fault Handler Priority Fields**

Handler	Field	Register Description
Memory management fault (MemManage)	PRI_4	“System Handler Priority Register 1”
Bus fault (BusFault)	PRI_5	
Usage fault (UsageFault)	PRI_6	
SVCALL	PRI_11	“System Handler Priority Register 2”
PendSV	PRI_14	“System Handler Priority Register 3”
SysTick	PRI_15	

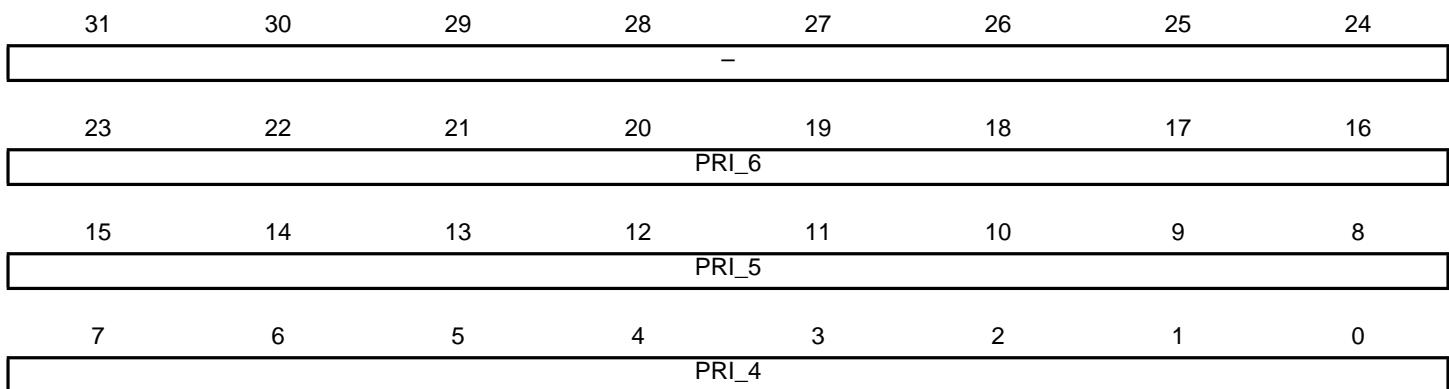
Each PRI\_N field is 8 bits wide, but the processor implements only bits [7:4] of each field, and bits [3:0] read as zero and ignore writes.

### 12.9.1.9 System Handler Priority Register 1

**Name:** SCB\_SHPR1

**Access:** Read-write

**Reset:** 0x0000000000



- **PRI\_6: Priority**

Priority of system handler 6, UsageFault.

- **PRI\_5: Priority**

Priority of system handler 5, BusFault.

- **PRI\_4: Priority**

Priority of system handler 4, MemManage.

### 12.9.1.10 System Handler Priority Register 2

**Name:** SCB\_SHPR2

**Access:** Read-write

**Reset:** 0x0000000000

31	30	29	28	27	26	25	24
PRI_11							
23	22	21	20	19	18	17	16
—							
15	14	13	12	11	10	9	8
—							
7	6	5	4	3	2	1	0
—							

- **PRI\_11: Priority**

Priority of system handler 11, SVCall.

### 12.9.1.11 System Handler Priority Register 3

**Name:** SCB\_SHPR3

**Access:** Read-write

**Reset:** 0x0000000000

31	30	29	28	27	26	25	24
PRI_15							
23	22	21	20	19	18	17	16
PRI_14							
15	14	13	12	11	10	9	8
—							
7	6	5	4	3	2	1	0
—							

- **PRI\_15: Priority**

Priority of system handler 15, SysTick exception.

- **PRI\_14: Priority**

Priority of system handler 14, PendSV.

### 12.9.1.12 System Handler Control and State Register

**Name:** SCB\_SHCSR

**Access:** Read-write

**Reset:** 0x000000000

31	30	29	28	27	26	25	24
				—			
23	22	21	20	19	18	17	16
		—		USGFAULTENA	BUSFAULTENA	MEMFAULTENA	
15	14	13	12	11	10	9	8
SVCALLPENDE D	BUSFAULTPEN DED	MEMFAULTPEN DED	USGFAULTPEN DED	SYSTICKACT	PENDSVACT	—	MONITORACT
7	6	5	4	3	2	1	0
SVCALLAVCT		—	USGFAULTACT	—	BUSFAULTACT	MEMFAULTACT	

The SHCSR register enables the system handlers, and indicates the pending status of the bus fault, memory management fault, and SVC exceptions; it also indicates the active status of the system handlers.

- **USGFAULTENA: Usage Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **BUSFAULTENA: Bus Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **MEMFAULTENA: Memory Management Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **SVCALLPENDED: SVC Call Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **BUSFAULTPENDED: Bus Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **MEMFAULTPENDED: Memory Management Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **USGFAULTPENDED: Usage Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **SYSTICKACT: SysTick Exception Active**

Read:

0: The exception is not active.

1: The exception is active.

Note: The user can write to these bits to change the active status of the exceptions.

- Caution: A software that changes the value of an active bit in this register without a correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure that the software writing to this register retains and subsequently restores the current active status.
- Caution: After enabling the system handlers, to change the value of a bit in this register, the user must use a read-modify-write procedure to ensure that only the required bit is changed.

- **PENDSVACT: PendSV Exception Active**

0: The exception is not active.

1: The exception is active.

- **MONITORACT: Debug Monitor Active**

0: Debug monitor is not active.

1: Debug monitor is active.

- **SVCALLACT: SVC Call Active**

0: SVC call is not active.

1: SVC call is active.

- **USGFAULTACT: Usage Fault Exception Active**

0: Usage fault exception is not active.

1: Usage fault exception is active.

- **BUSFAULTACT: Bus Fault Exception Active**

0: Bus fault exception is not active.

1: Bus fault exception is active.

- **MEMFAULTACT: Memory Management Fault Exception Active**

0: Memory management fault exception is not active.

1: Memory management fault exception is active.

If the user disables a system handler and the corresponding fault occurs, the processor treats the fault as a hard fault.

The user can write to this register to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

### 12.9.1.13 Configurable Fault Status Register

**Name:** SCB\_CFSR

**Access:** Read-write

**Reset:** 0x0000000000

31	30	29	28	27	26	25	24
—					DIVBYZERO	UNALIGNED	
23	22	21	20	19	18	17	16
—			NOCP	INVPC	INVSTATE	UNDEFINSTR	
15	14	13	12	11	10	9	8
BFRVALID	—	STKERR	UNSTKERR	IMPRECISERR	PRECISERR	IBUSERR	
7	6	5	4	3	2	1	0
MMARVALID	—	MLSPERR	MSTKERR	MUNSTKERR	—	DACCVIOL	IACCVIOL

- **IACCVIOL: Instruction Access Violation Flag**

This is part of "[MMFSR: Memory Management Fault Status Subregister](#)" .

0: No instruction access violation fault.

1: The processor attempted an instruction fetch from a location that does not permit execution.

This fault occurs on any access to an XN region, even when the MPU is disabled or not present.

When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has not written a fault address to the SCB\_MMFAR register.

- **DACCVIOL: Data Access Violation Flag**

This is part of "[MMFSR: Memory Management Fault Status Subregister](#)" .

0: No data access violation fault.

1: The processor attempted a load or store at a location that does not permit the operation.

When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has loaded the SCB\_MMFAR register with the address of the attempted access.

- **MUNSTKERR: Memory Manager Fault on Unstacking for a Return From Exception**

This is part of "[MMFSR: Memory Management Fault Status Subregister](#)" .

0: No unstacking fault.

1: Unstack for an exception return has caused one or more access violations.

This fault is chained to the handler. This means that when this bit is 1, the original return stack is still present. The processor has not adjusted the SP from the failing return, and has not performed a new save. The processor has not written a fault address to the SCB\_MMFAR register.

- **MSTKERR: Memory Manager Fault on Stacking for Exception Entry**

This is part of "[MMFSR: Memory Management Fault Status Subregister](#)" .

0: No stacking fault.

1: Stacking for an exception entry has caused one or more access violations.

When this bit is 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor has not written a fault address to SCB\_MMFAR register.

- **MLSPERR: MemManage during Lazy State Preservation**

This is part of “[MMFSR: Memory Management Fault Status Subregister](#)” .

0: No MemManage fault occurred during the floating-point lazy state preservation.

1: A MemManage fault occurred during the floating-point lazy state preservation.

- **MMARVALID: Memory Management Fault Address Register (SCB\_MMFAR) Valid Flag**

This is part of “[MMFSR: Memory Management Fault Status Subregister](#)” .

0: The value in SCB\_MMFAR is not a valid fault address.

1: SCB\_MMFAR register holds a valid fault address.

If a memory management fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems on return to a stacked active memory management fault handler whose SCB\_MMFAR value has been overwritten.

- **IBUSERR: Instruction Bus Error**

This is part of “[BFSR: Bus Fault Status Subregister](#)” .

0: No instruction bus error.

1: Instruction bus error.

The processor detects the instruction bus error on prefetching an instruction, but it sets the IBUSERR flag to 1 only if it attempts to issue the faulting instruction.

When the processor sets this bit to 1, it does not write a fault address to the BFAR register.

- **PRECISERR: Precise Data Bus Error**

This is part of “[BFSR: Bus Fault Status Subregister](#)” .

0: No precise data bus error.

1: A data bus error has occurred, and the PC value stacked for the exception return points to the instruction that caused the fault.

When the processor sets this bit to 1, it writes the faulting address to the SCB\_BFAR register.

- **IMPRECISERR: Imprecise Data Bus Error**

This is part of “[BFSR: Bus Fault Status Subregister](#)” .

0: No imprecise data bus error.

1: A data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error.

When the processor sets this bit to 1, it does not write a fault address to the SCB\_BFAR register.

This is an asynchronous fault. Therefore, if it is detected when the priority of the current process is higher than the bus fault priority, the bus fault becomes pending and becomes active only when the processor returns from all higher priority processes. If a precise fault occurs before the processor enters the handler for the imprecise bus fault, the handler detects that both this bit and one of the precise fault status bits are set to 1.

- **UNSTKERR: Bus Fault on Unstacking for a Return From Exception**

This is part of “[BFSR: Bus Fault Status Subregister](#)” .

0: No unstacking fault.

1: Unstack for an exception return has caused one or more bus faults.

This fault is chained to the handler. This means that when the processor sets this bit to 1, the original return stack is still present. The processor does not adjust the SP from the failing return, does not perform a new save, and does not write a fault address to the BFAR.

- **STKERR: Bus Fault on Stacking for Exception Entry**

This is part of “[BFSR: Bus Fault Status Subregister](#)” .

0: No stacking fault.

1: Stacking for an exception entry has caused one or more bus faults.

When the processor sets this bit to 1, the SP is still adjusted but the values in the context area on the stack might be incorrect.

The processor does not write a fault address to the SCB\_BFAR register.

- **BFARVALID: Bus Fault Address Register (BFAR) Valid flag**

This is part of “[BFSR: Bus Fault Status Subregister](#)” .

0: The value in SCB\_BFAR is not a valid fault address.

1: SCB\_BFAR holds a valid fault address.

The processor sets this bit to 1 after a bus fault where the address is known. Other faults can set this bit to 0, such as a memory management fault occurring later.

If a bus fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems if returning to a stacked active bus fault handler whose SCB\_BFAR value has been overwritten.

- **UNDEFINSTR: Undefined Instruction Usage Fault**

This is part of “[UFSR: Usage Fault Status Subregister](#)” .

0: No undefined instruction usage fault.

1: The processor has attempted to execute an undefined instruction.

When this bit is set to 1, the PC value stacked for the exception return points to the undefined instruction.

An undefined instruction is an instruction that the processor cannot decode.

- **INVSTATE: Invalid State Usage Fault**

This is part of “[UFSR: Usage Fault Status Subregister](#)” .

0: No invalid state usage fault.

1: The processor has attempted to execute an instruction that makes illegal use of the EPSR.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that attempted the illegal use of the EPSR.

This bit is not set to 1 if an undefined instruction uses the EPSR.

- **INVPC: Invalid PC Load Usage Fault**

This is part of “[UFSR: Usage Fault Status Subregister](#)” . It is caused by an invalid PC load by EXC\_RETURN:

0: No invalid PC load usage fault.

1: The processor has attempted an illegal load of EXC\_RETURN to the PC, as a result of an invalid context, or an invalid EXC\_RETURN value.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that tried to perform the illegal load of the PC.

- **NOCP: No Coprocessor Usage Fault**

This is part of “[UFSR: Usage Fault Status Subregister](#)” . The processor does not support coprocessor instructions:

0: No usage fault caused by attempting to access a coprocessor.

1: The processor has attempted to access a coprocessor.

- **UNALIGNED: Unaligned Access Usage Fault**

This is part of “[UFSR: Usage Fault Status Subregister](#)” .

0: No unaligned access fault, or unaligned access trapping not enabled.

1: The processor has made an unaligned memory access.

Enable trapping of unaligned accesses by setting the UNALIGN\_TRP bit in the SCB\_CCR register to 1. See “[Configuration and Control Register](#)” . Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of the setting of UNALIGN\_TRP.

- **DIVBYZERO: Divide by Zero Usage Fault**

This is part of “[UFSR: Usage Fault Status Subregister](#)” .

0: No divide by zero fault, or divide by zero trapping not enabled.

1: The processor has executed an SDIV or UDIV instruction with a divisor of 0.

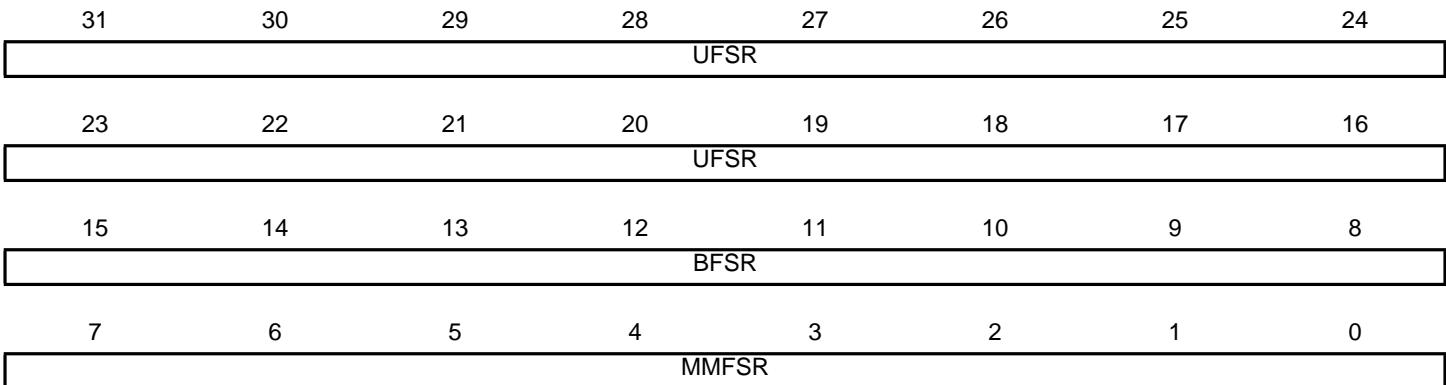
When the processor sets this bit to 1, the PC value stacked for the exception return points to the instruction that performed the divide by zero. Enable trapping of divide by zero by setting the DIV\_0\_TRP bit in the SCB\_CCR register to 1. See “[Configuration and Control Register](#)” .

#### 12.9.1.14 Configurable Fault Status Register (Byte Access)

**Name:** SCB\_CFSR (BYTE)

**Access:** Read-write

**Reset:** 0x000000000



- **MMFSR: Memory Management Fault Status Subregister**

The flags in the MMFSR subregister indicate the cause of memory access faults. See bitfield [7..0] description in [Section 12.9.1.13](#).

- **BFSR: Bus Fault Status Subregister**

The flags in the BFSR subregister indicate the cause of a bus access fault. See bitfield [14..8] description in [Section 12.9.1.13](#).

- **UFSR: Usage Fault Status Subregister**

The flags in the UFSR subregister indicate the cause of a usage fault. See bitfield [31..15] description in [Section 12.9.1.13](#).

**Note:** The UFSR bits are sticky. This means that as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

The SCB\_CFSR register indicates the cause of a memory management fault, bus fault, or usage fault. It is byte accessible. The user can access the SCB\_CFSR register or its subregisters as follows:

- Access complete SCB\_CFSR with a word access to 0xE000ED28
- Access MMFSR with a byte access to 0xE000ED28
- Access MMFSR and BFSR with a halfword access to 0xE000ED28
- Access BFSR with a byte access to 0xE000ED29
- Access UFSR with a halfword access to 0xE000ED2A.

### 12.9.1.15 Hard Fault Status Register

**Name:** SCB\_HFSR

**Access:** Read-write

**Reset:** 0x000000000

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

The HFSR register gives information about events that activate the hard fault handler. This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0.

- **DEBUGEVT: Reserved for Debug Use**

When writing to the register, write 0 to this bit, otherwise the behavior is unpredictable.

- **FORCED: Forced Hard Fault**

It indicates a forced hard fault, generated by escalation of a fault with configurable priority that cannot be handles, either because of priority or because it is disabled:

0: No forced hard fault.

1: Forced hard fault.

When this bit is set to 1, the hard fault handler must read the other fault status registers to find the cause of the fault.

- **VECTTBL: Bus Fault on a Vector Table**

It indicates a bus fault on a vector table read during an exception processing:

0: No bus fault on vector table read.

1: Bus fault on vector table read.

This error is always handled by the hard fault handler.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that was preempted by the exception.

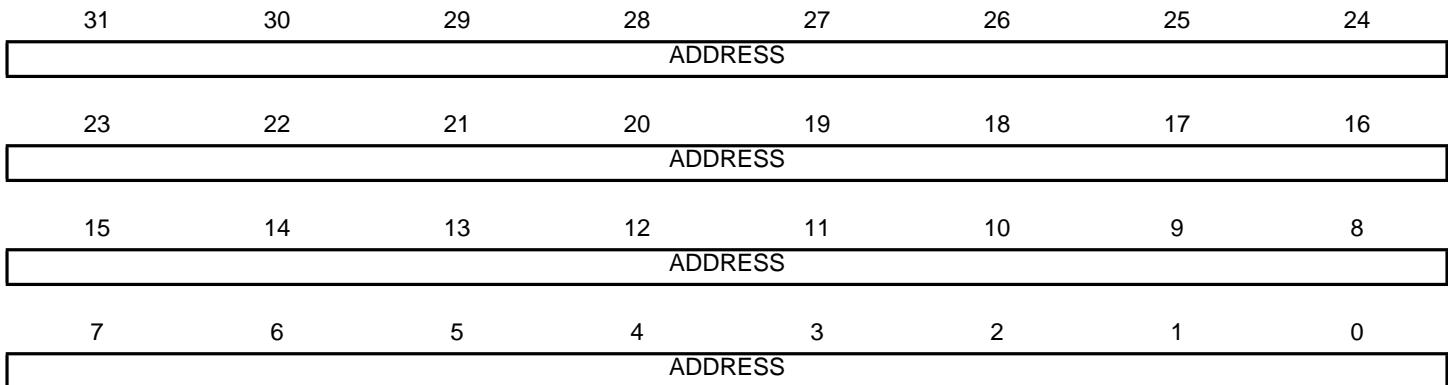
**Note:** The HFSR bits are sticky. This means that, as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

### 12.9.1.16 MemManage Fault Address Register

**Name:** SCB\_MMFAR

**Access:** Read-write

**Reset:** 0x0000000000



The MMFAR register contains the address of the location that generated a memory management fault.

- **ADDRESS**

When the MMARVALID bit of the MMFSR subregister is set to 1, this field holds the address of the location that generated the memory management fault.

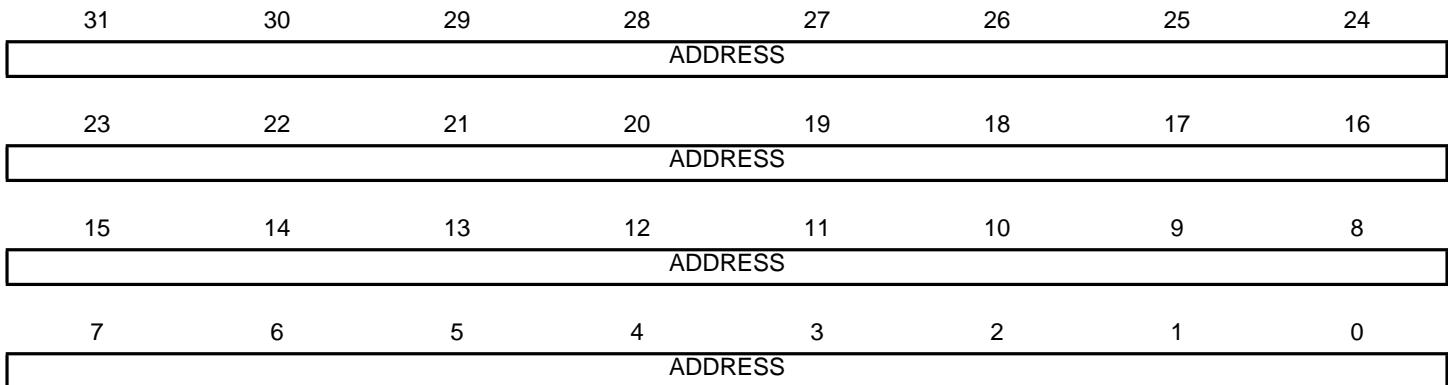
- Notes:
1. When an unaligned access faults, the address is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size.
  2. Flags in the MMFSR subregister indicate the cause of the fault, and whether the value in the SCB\_MMFAR register is valid. See "[MMFSR: Memory Management Fault Status Subregister](#)".

### 12.9.1.17 Bus Fault Address Register

**Name:** SCB\_BFAR

**Access:** Read-write

**Reset:** 0x000000000



The BFAR register contains the address of the location that generated a bus fault.

- **ADDRESS**

When the BFARVALID bit of the BFSR subregister is set to 1, this field holds the address of the location that generated the bus fault.

- Notes:
1. When an unaligned access faults, the address in the SCB\_BFAR register is the one requested by the instruction, even if it is not the address of the fault.
  2. Flags in the BFSR indicate the cause of the fault, and whether the value in the SCB\_BFAR register is valid. See “[BFSR: Bus Fault Status Subregister](#)” .

## 12.10 System Timer (SysTick)

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads (wraps to) the value in the SYST\_RVR register on the next clock edge, then counts down on subsequent clocks.

When the processor is halted for debugging, the counter does not decrement.

The SysTick counter runs on the processor clock. If this clock signal is stopped for low power mode, the SysTick counter stops.

Ensure that the software uses aligned word accesses to access the SysTick registers.

The SysTick counter reload and current value are undefined at reset; the correct initialization sequence for the SysTick counter is:

1. Program the reload value.
2. Clear the current value.
3. Program the Control and Status register.

### 12.10.1 System Timer (SysTick) User Interface

Table 12-35. System Timer (SYST) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E010	SysTick Control and Status Register	SYST_CSR	Read-write	0x00000004
0xE000E014	SysTick Reload Value Register	SYST_RVR	Read-write	Unknown
0xE000E018	SysTick Current Value Register	SYST_CVR	Read-write	Unknown
0xE000E01C	SysTick Calibration Value Register	SYST_CALIB	Read-only	0xC0000000

### 12.10.1.1 SysTick Control and Status

**Name:** SYST\_CSR

**Access:** Read-write

**Reset:** 0x0000000000

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

The SysTick SYST\_CSR register enables the SysTick features.

- **COUNTFLAG: Count Flag**

Returns 1 if the timer counted to 0 since the last time this was read.

- **CLKSOURCE: Clock Source**

Indicates the clock source:

0: External Clock.

1: Processor Clock.

- **TICKINT**

Enables a SysTick exception request:

0: Counting down to zero does not assert the SysTick exception request.

1: Counting down to zero asserts the SysTick exception request.

The software can use COUNTFLAG to determine if SysTick has ever counted to zero.

- **ENABLE**

Enables the counter:

0: Counter disabled.

1: Counter enabled.

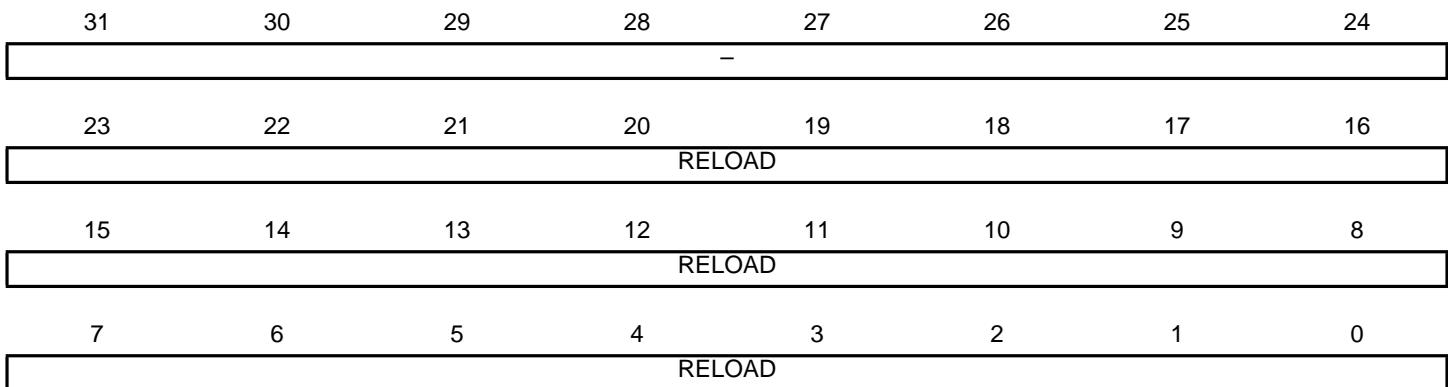
When ENABLE is set to 1, the counter loads the RELOAD value from the SYST\_RVR register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

### 12.10.1.2 SysTick Reload Value Registers

**Name:** SYST\_RVR

**Access:** Read-write

**Reset:** 0x0000000000



The SYST\_RVR register specifies the start value to load into the SYST\_CVR register.

- **RELOAD**

Value to load into the SYST\_CVR register when the counter is enabled and when it reaches 0.

The RELOAD value can be any value in the range 0x00000001-0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

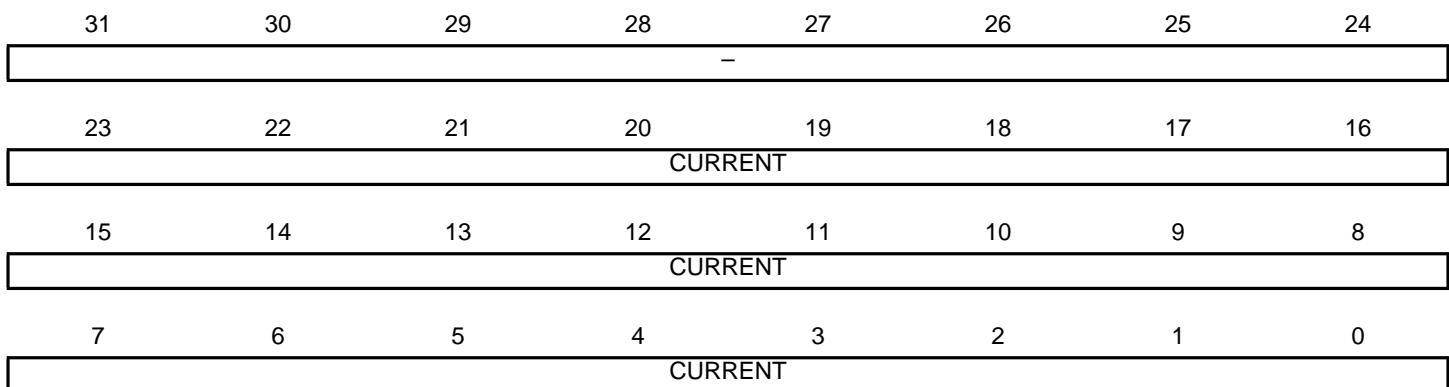
The RELOAD value is calculated according to its use: For example, to generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. If the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

### 12.10.1.3 SysTick Current Value Register

**Name:** SYST\_CVR

**Access:** Read-write

**Reset:** 0x0000000000



The SysTick SYST\_CVR register contains the current value of the SysTick counter.

- **CURRENT**

Reads return the current value of the SysTick counter.

A write of any value clears the field to 0, and also clears the SYST\_CSR.COUNTFLAG bit to 0.

#### 12.10.1.4 SysTick Calibration Value Register

**Name:** SYST\_CALIB

**Access:** Read-write

**Reset:** 0x0000000000

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

The SysTick SYST\_CSR register indicates the SysTick calibration properties.

- **NOREF: No Reference Clock**

It indicates whether the device provides a reference clock to the processor:

0: Reference clock provided.

1: No reference clock provided.

If your device does not provide a reference clock, the SYST\_CSR.CLKSOURCE bit reads-as-one and ignores writes.

- **SKEW**

It indicates whether the TENMS value is exact:

0: TENMS value is exact.

1: TENMS value is inexact, or not given.

An inexact TENMS value can affect the suitability of SysTick as a software real-time clock.

- **TENMS: Ten Milliseconds**

The reload value for 10 ms (100 Hz) timing is subject to system clock skew errors. If the value reads as zero, the calibration value is not known.

## 12.11 Memory Protection Unit (MPU)

The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:

- Independent attribute settings for each region
- Overlapping regions
- Export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M4 MPU defines:

- Eight separate memory regions, 0-7
- A background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M4 MPU memory map is unified. This means that instruction accesses and data accesses have the same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault. This causes a fault exception, and might cause the termination of the process in an OS environment.

In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

The configuration of MPU regions is based on memory types (see "[Memory Regions, Types and Attributes](#)").

[Table](#) shows the possible MPU region attributes. These include Share ability and cache behavior attributes that are not relevant to most microcontroller implementations. See "[MPU Configuration for a Microcontroller](#)" for guidelines for programming such an implementation.

Memory Attributes Summary

Memory Type	Shareability	Other Attributes	Description
Strongly- ordered	-	-	All accesses to Strongly-ordered memory occur in program order. All Strongly-ordered regions are assumed to be shared.
Device	Shared	-	Memory-mapped peripherals that several processors share.
	Non-shared	-	Memory-mapped peripherals that only a single processor uses.
Normal	Shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that is shared between several processors.
	Non-shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that only a single processor uses.

### 12.11.1 MPU Access Permission Attributes

This section describes the MPU access permission attributes. The access permission bits (TEX, C, B, S, AP, and XN) of the MPU\_RASR control the access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The table below shows the encodings for the TEX, C, B, and S access permission bits.

**Table 12-36. TEX, C, B, and S Encoding**

TEX	C	B	S	Memory Type	Shareability	Other Attributes
b000	0	0	x <sup>(1)</sup>	Strongly-ordered	Shareable	-
		1	x <sup>(1)</sup>	Device	Shareable	-
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
					Shareable	
		1	1	Normal	Not shareable	
					Shareable	
	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
					Shareable	
b001	1	x <sup>(1)</sup>	Reserved encoding		-	
	1	0	x <sup>(1)</sup>	Implementation defined attributes.		-
		1	0	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
					Shareable	
b010	0	0	x <sup>(1)</sup>	Device	Not shareable	Nonshared Device.
		1	x <sup>(1)</sup>	Reserved encoding		-
	1	x <sup>(1)</sup>	x <sup>(1)</sup>	Reserved encoding		-
b1B B	A	A	0	Normal	Not shareable	Cached memory BB = outer policy, AA = inner policy.
			1		Shareable	

Note: 1. The MPU ignores the value of this bit.

Table 12-37 shows the cache policy for memory attribute encodings with a TEX value in the range 4-7.

**Table 12-37. Cache Policy for Memory Attribute Encoding**

Encoding, AA or BB	Corresponding Cache Policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

[Table 12-38](#) shows the AP encodings that define the access permissions for privileged and unprivileged software.

**Table 12-38. AP Encoding**

AP[2:0]	Privileged Permissions	Unprivileged Permissions	Description
000	No access	No access	All accesses generate a permission fault
001	RW	No access	Access from privileged software only
010	RW	RO	Writes by unprivileged software generate a permission fault
011	RW	RW	Full access
100	Unpredictable	Unpredictable	Reserved
101	RO	No access	Reads by privileged software only
110	RO	RO	Read only, by privileged or unprivileged software
111	RO	RO	Read only, by privileged or unprivileged software

#### 12.11.1.1 MPU Mismatch

When an access violates the MPU permissions, the processor generates a memory management fault, see “[Exceptions and Interrupts](#)”. The MMFSR indicates the cause of the fault. See “[MMFSR: Memory Management Fault Status Subregister](#)” for more information.

#### 12.11.1.2 Updating an MPU Region

To update the attributes for an MPU region, update the MPU\_RNR, MPU\_RBAR and MPU\_RASR registers. Each register can be programmed separately, or a multiple-word write can be used to program all of these registers. MPU\_RBAR and MPU\_RASR aliases can be used to program up to four regions simultaneously using an STM instruction.

#### 12.11.1.3 Updating an MPU Region Using Separate Words

Simple code to configure one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPU_RNR           ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]         ; Region Number
STR R4, [R0, #0x4]         ; Region Base Address
STRH R2, [R0, #0x8]        ; Region Size and Enable
STRH R3, [R0, #0xA]        ; Region Attribute
```

Disable a region before writing new region settings to the MPU, if the region being changed was previously enabled. For example:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPU_RNR           ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]         ; Region Number
BIC R2, R2, #1             ; Disable
STRH R2, [R0, #0x8]        ; Region Size and Enable
STR R4, [R0, #0x4]         ; Region Base Address
STRH R3, [R0, #0xA]        ; Region Attribute
ORR R2, #1                 ; Enable
STRH R2, [R0, #0x8]        ; Region Size and Enable
```

The software must use memory barrier instructions:

- Before the MPU setup, if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings
- After the MPU setup, if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanisms cause memory barrier behavior.

The software does not need any memory barrier instructions during an MPU setup, because it accesses the MPU through the PPB, which is a Strongly-Ordered memory region.

For example, if the user wants all of the memory access behavior to take effect immediately after the programming sequence, a DSB instruction and an ISB instruction must be used. A DSB is required after changing MPU settings, such as at the end of a context switch. An ISB is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then an ISB is not required.

#### 12.11.1.4 Updating an MPU Region Using Multi-word Writes

The user can program directly using multi-word writes, depending on how the information is divided. Consider the following reprogramming:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR      ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]    ; Region Number
STR R2, [R0, #0x4]    ; Region Base Address
STR R3, [R0, #0x8]    ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR      ; 0xE000ED98, MPU region number register
STM R0, {R1-R3}       ; Region Number, address, attribute, size and enable
```

This can be done in two words for pre-packed information. This means that the MPU\_RBAR contains the required region number and had the VALID bit set to 1. See “[MPU Region Base Address Register](#)”. Use this when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPU_RBAR     ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]    ; Region base address and
                      ; region number combined with VALID (bit 4) set to 1
STR R2, [R0, #0x4]    ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPU_RBAR     ; 0xE000ED9C, MPU Region Base register
STM R0, {R1-R2}       ; Region base address, region number and VALID bit,
                      ; and Region Attribute, Size and Enable
```

### 12.11.1.5 Subregions

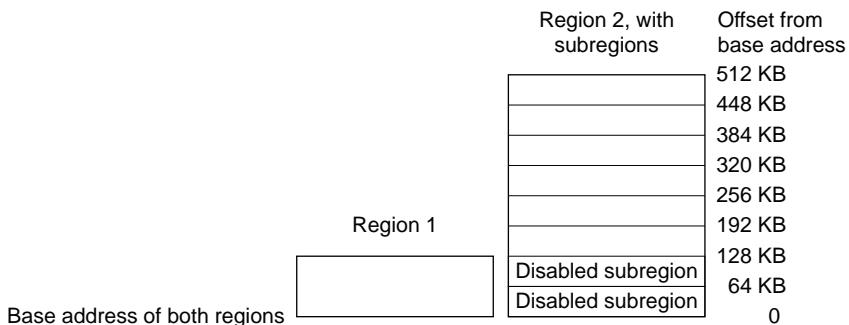
Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU\_RASR register to disable a subregion. See “[MPU Region Attribute and Size Register](#)”. The least significant bit of SRD controls the first subregion, and the most significant bit controls the last subregion. Disabling a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion, the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions. With regions of these sizes, the SRD field must be set to 0x00, otherwise the MPU behavior is unpredictable.

### 12.11.1.6 Example of SRD Use

Two regions with the same base address overlap. Region 1 is 128 KB, and region 2 is 512 KB. To ensure the attributes from region 1 apply to the first 128 KB region, set the SRD field for region 2 to b00000011 to disable the first two subregions, as in [Figure 12-13](#) below:

**Figure 12-13. SRD Use**



### 12.11.1.7 MPU Design Hints And Tips

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

Ensure the software uses aligned accesses of the correct size to access MPU registers:

- Except for the MPU\_RASR register, it must use aligned word accesses
- For the MPU\_RASR register, it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to MPU registers.

When setting up the MPU, and if the MPU has previously been programmed, disable unused regions to prevent any previous region settings from affecting the new MPU setup.

#### *MPU Configuration for a Microcontroller*

Usually, a microcontroller system has only a single processor and no caches. In such a system, program the MPU as follows:

**Table 12-39. Memory Region Attributes for a Microcontroller**

Memory Region	TEX	C	B	S	Memory Type and Attributes
Flash memory	b000	1	0	0	Normal memory, non-shareable, write-through
Internal SRAM	b000	1	0	1	Normal memory, shareable, write-through
External SRAM	b000	1	1	1	Normal memory, shareable, write-back, write-allocate
Peripherals	b000	0	1	1	Device memory, shareable

In most microcontroller implementations, the shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable. The values given are for typical situations. In special systems, such as multiprocessor designs or designs with a separate DMA engine, the

shareability attribute might be important. In these cases, refer to the recommendations of the memory device manufacturer.

### 12.11.2 Memory Protection Unit (MPU) User Interface

**Table 12-40. Memory Protection Unit (MPU) Register Mapping**

Offset	Register	Name	Access	Reset
0xE000ED90	MPU Type Register	MPU_TYPE	Read-only	0x00000800
0xE000ED94	MPU Control Register	MPU_CTRL	Read-write	0x00000000
0xE000ED98	MPU Region Number Register	MPU_RNR	Read-write	0x00000000
0xE000ED9C	MPU Region Base Address Register	MPU_RBAR	Read-write	0x00000000
0xE000EDA0	MPU Region Attribute and Size Register	MPU_RASR	Read-write	0x00000000
0xE000EDA4	Alias of RBAR, see MPU Region Base Address Register	MPU_RBAR_A1	Read-write	0x00000000
0xE000EDA8	Alias of RASR, see MPU Region Attribute and Size Register	MPU_RASR_A1	Read-write	0x00000000
0xE000EDAC	Alias of RBAR, see MPU Region Base Address Register	MPU_RBAR_A2	Read-write	0x00000000
0xE000EDB0	Alias of RASR, see MPU Region Attribute and Size Register	MPU_RASR_A2	Read-write	0x00000000
0xE000EDB4	Alias of RBAR, see MPU Region Base Address Register	MPU_RBAR_A3	Read-write	0x00000000
0xE000EDB8	Alias of RASR, see MPU Region Attribute and Size Register	MPU_RASR_A3	Read-write	0x00000000

### 12.11.2.1 MPU Type Register

**Name:** MPU\_TYPE

**Access:** Read-write

**Reset:** 0x00000800

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

The MPU\_TYPE register indicates whether the MPU is present, and if so, how many regions it supports.

- **IREGION: Instruction Region**

Indicates the number of supported MPU instruction regions.

Always contains 0x00. The MPU memory map is unified and is described by the DREGION field.

- **DREGION: Data Region**

Indicates the number of supported MPU data regions:

0x08 = Eight MPU regions.

- **SEPARATE: Separate Instruction**

Indicates support for unified or separate instruction and date memory maps:

0: Unified.

### 12.11.2.2 MPU Control Register

**Name:** MPU\_CTRL

**Access:** Read-write

**Reset:** 0x00000800

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
		—			PRIVDEFENA	HFNMIENA	ENABLE

The MPU CTRL register enables the MPU, enables the default memory map background region, and enables the use of the MPU when in the hard fault, Non-maskable Interrupt (NMI), and FAULTMASK escalated handlers.

- **PRIVDEFENA: Privileged Default Memory Map Enabled**

Enables privileged software access to the default memory map:

0: If the MPU is enabled, disables the use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault.

1: If the MPU is enabled, enables the use of the default memory map as a background region for privileged software accesses.

When enabled, the background region acts as a region number -1. Any region that is defined and enabled has priority over this default map.

If the MPU is disabled, the processor ignores this bit.

- **HFNMIENA: Hard Fault and NMI Enabled**

Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers.

When the MPU is enabled:

0: MPU is disabled during hard fault, NMI, and FAULTMASK handlers, regardless of the value of the ENABLE bit.

1: The MPU is enabled during hard fault, NMI, and FAULTMASK handlers.

When the MPU is disabled, if this bit is set to 1, the behavior is unpredictable.

- **ENABLE**

Enables the MPU:

0: MPU disabled.

1: MPU enabled.

When ENABLE and PRIVDEFENA are both set to 1:

- For privileged accesses, the *default memory map* is as described in “[Memory Model](#)” . Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a memory management fault.

XN and Strongly-ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set to 1, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFENA bit is set to 1. If the PRIVDEFENA bit is set to 1 and no regions are enabled, then only privileged software can operate.

When the ENABLE bit is set to 0, the system uses the default memory map. This has the same memory attributes as if the MPU is not implemented. The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFENA is set to 1.

Unless HFNMIEA is set to 1, the MPU is not enabled when the processor is executing the handler for an exception with priority -1 or -2. These priorities are only possible when handling a hard fault or NMI exception, or when FAULTMASK is enabled. Setting the HFNMIEA bit to 1 enables the MPU when operating with these two priorities.

### 12.11.2.3 MPU Region Number Register

**Name:** MPU\_RNR

**Access:** Read-write

**Reset:** 0x00000800

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
				REGION			

The MPU\_RNR selects which memory region is referenced by the MPU\_RBAR and MPU\_RASR registers.

- **REGION**

Indicates the MPU region referenced by the MPU\_RBAR and MPU\_RASR registers.

The MPU supports 8 memory regions, so the permitted values of this field are 0-7.

Normally, the required region number is written to this register before accessing the MPU\_RBAR or MPU\_RASR. However, the region number can be changed by writing to the MPU\_RBAR with the VALID bit set to 1; see “[MPU Region Base Address Register](#)”. This write updates the value of the REGION field.

#### 12.11.2.4 MPU Region Base Address Register

**Name:** MPU\_RBAR

**Access:** Read-write

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
ADDR							
23	22	21	20	19	18	17	16
ADDR							
15	14	13	12	11	10	9	N
ADDR							
N-1	6	5	4	3	2	1	0
—	VALID					REGION	

Note: If the region size is 32B, the ADDR field is bits [31:5] and there is no Reserved field.

The MPU\_RBAR defines the base address of the MPU region selected by the MPU\_RNR, and can update the value of the MPU\_RNR.

Write MPU\_RBAR with the VALID bit set to 1 to change the current region number and update the MPU\_RNR.

- **ADDR: Region Base Address**

The value of N depends on the region size. The ADDR field is bits[31:N] of the MPU\_RBAR. The region size, as specified by the SIZE field in the MPU\_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4 GB, in the MPU\_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU\_RNR not changed, and the processor updates the base address for the region specified in the MPU\_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU\_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

For the behavior on writes, see the description of the VALID field.

On reads, returns the current region number, as specified by the MPU\_RNR.

### 12.11.2.5 MPU Region Attribute and Size Register

**Name:** MPU\_RASR

**Access:** Read-write

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
–	–	XN	–	–	AP	–	–
23	22	21	20	19	18	17	16
–	–	TEX	–	S	C	B	–
15	14	13	12	11	10	9	8
–	–	SRD	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	SIZE	–	–	–	–	ENABLE

The MPU\_RASR defines the region size and memory attributes of the MPU region specified by the MPU\_RNR, and enables that region and any subregions.

MPU\_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

#### • **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

#### • **AP: Access Permission**

See [Table 12-38](#).

#### • **TEX, C, B: Memory Access Attributes**

See [Table 12-36](#).

#### • **S: Shareable**

See [Table 12-36](#).

#### • **SRD: Subregion Disable**

For each bit in this field:

0: Corresponding sub-region is enabled.

1: Corresponding sub-region is disabled.

See “[Subregions](#)” for more information.

Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.

#### • **SIZE: Size of the MPU Protection Region**

The minimum permitted value is 3 (b00010).

The SIZE field defines the size of the MPU memory region specified by the MPU\_RNR. as follows:

$$(\text{Region size in bytes}) = 2^{(\text{SIZE}+1)}$$

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. The table below gives an example of SIZE values, with the corresponding region size and value of N in the MPU\_RBAR.

SIZE Value	Region Size	Value of N <sup>(1)</sup>	Note
b00100 (4)	32 B	5	Minimum permitted size
b01001 (9)	1 KB	10	-
b10011 (19)	1 MB	20	-
b11101 (29)	1 GB	30	-
b11111 (31)	4 GB	b01100	Maximum possible size

Note: 1. In the MPU\_RBAR, see "[MPU Region Base Address Register](#)"

- **ENABLE: Region Enable**

Note: For information about access permission, see "[MPU Access Permission Attributes](#)".

## 12.12 Floating Point Unit (FPU)

The Cortex-M4F FPU implements the FPv4-SP floating-point extension.

The FPU fully supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

The FPU provides floating-point computation functionality that is compliant with the ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic, referred to as the IEEE 754 standard.

The FPU contains 32 single-precision extension registers, which can also be accessed as 16 doubleword registers for load, store, and move operations.

### 12.12.1 Enabling the FPU

The FPU is disabled from reset. It must be enabled before any floating-point instructions can be used. Example 4-1 shows an example code sequence for enabling the FPU in both privileged and user modes. The processor must be in privileged mode to read from and write to the CPACR.

Example of Enabling the FPU:

```
; CPACR is located at address 0xE000ED88
LDR.W R0, =0xE000ED88
; Read CPACR
LDR R1, [R0]
; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR R1, R1, #(0xF << 20)
; Write back the modified value to the CPACR
STR R1, [R0]; wait for store to complete
DSB
;reset pipeline now the FPU is enabled
ISB
```

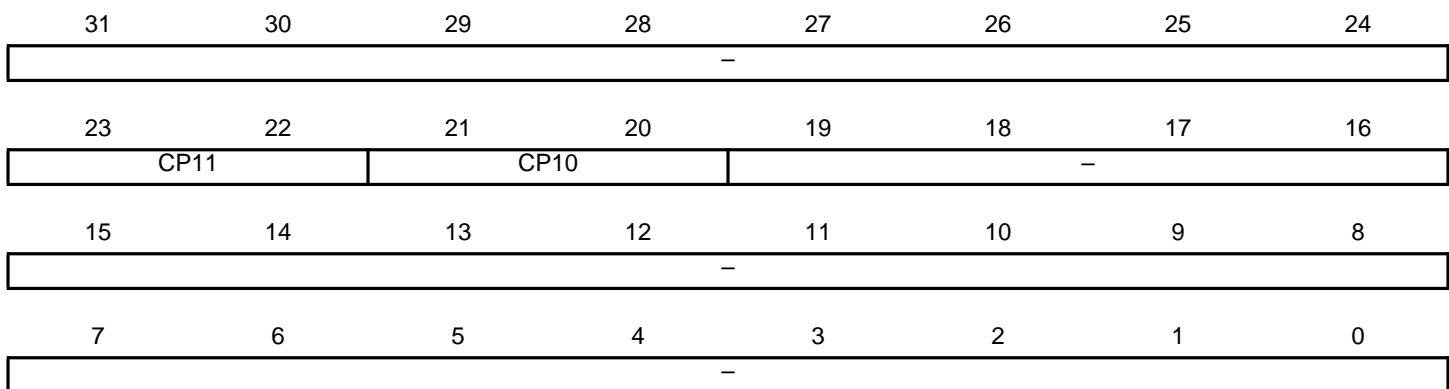
## 12.12.2 Floating Point Unit (FPU) User Interface

Table 12-41. Floating Point Unit (FPU) Register Mapping

Offset	Register	Name	Access	Reset
0xE000ED88	Coprocessor Access Control Register	CPACR	Read-write	0x00000000
0xE000EF34	Floating-point Context Control Register	FPCCR	Read-write	0xC0000000
0xE000EF38	Floating-point Context Address Register	FPCAR	Read-write	–
–	Floating-point Status Control Register	FPSCR	Read-write	–
0xE000E01C	Floating-point Default Status Control Register	FPDSCR	Read-write	0x00000000

#### 12.12.2.1 Coprocessor Access Control Register

**Name:** CPACR  
**Access:** Read-write  
**Reset:** 0x0000000000



The CPACR register specifies the access privileges for coprocessors.

- **CPn: Access Privileges for Coprocessor n [2n+1:2n], for n Values 10 and 11.**

The possible values of each field are:

0b00 = Access denied. Any attempted access generates a NOCP UsageFault.

0b01 = Privileged access only. An unprivileged access generates a NOCP fault.

0b10 = Reserved. The result of any access is unpredictable.

0b11 = Full access.

### 12.12.2.2 Floating-point Context Control Register

**Name:** FPCCR  
**Access:** Read-write  
**Reset:** 0x0000000000

31	30	29	28	27	26	25	24
ASPEN	LSPEN			—			
23	22	21	20	19	18	17	16
				—			
15	14	13	12	11	10	9	8
			—				MONRDY
7	6	5	4	3	2	1	0
—	BFRDY	MMRDY	HFRDY	THREAD	—	USER	LSPACT

The FPCCR register sets or returns FPU control data.

- **ASPEN: Automatic Hardware State Preservation And Restoration**

Enables CONTROL bit [2] setting on execution of a floating-point instruction. This results in an automatic hardware state preservation and restoration, for floating-point context, on exception entry and exit.

- 0: Disable CONTROL bit [2] setting on execution of a floating-point instruction.  
 1: Enable CONTROL bit [2] setting on execution of a floating-point instruction.

- **LSPEN: Automatic Lazy State Preservation**

- 0: Disable automatic lazy state preservation for floating-point context.  
 1: Enable automatic lazy state preservation for floating-point context.

- **MONRDY: Debug Monitor Ready**

- 0: DebugMonitor is disabled or the priority did not permit to set MON\_PEND when the floating-point stack frame was allocated.  
 1: DebugMonitor is enabled and the priority permitted to set MON\_PEND when the floating-point stack frame was allocated.

- **BFRDY: Bus Fault Ready**

- 0: BusFault is disabled or the priority did not permit to set the BusFault handler to the pending state when the floating-point stack frame was allocated.  
 1: BusFault is enabled and the priority permitted to set the BusFault handler to the pending state when the floating-point stack frame was allocated.

- **MMRDY: Memory Management Ready**

- 0: MemManage is disabled or the priority did not permit to set the MemManage handler to the pending state when the floating-point stack frame was allocated.  
 1: MemManage is enabled and the priority permitted to set the MemManage handler to the pending state when the floating-point stack frame was allocated.

- **HFRDY: Hard Fault Ready**

- 0: The priority did not permit to set the HardFault handler to the pending state when the floating-point stack frame was allocated.  
 1: The priority permitted to set the HardFault handler to the pending state when the floating-point stack frame was allocated.

- **THREAD: Thread Mode**

0: The mode was not the Thread Mode when the floating-point stack frame was allocated.

1: The mode was the Thread Mode when the floating-point stack frame was allocated.

- **USER: User Privilege Level**

0: The privilege level was not User when the floating-point stack frame was allocated.

1: The privilege level was User when the floating-point stack frame was allocated.

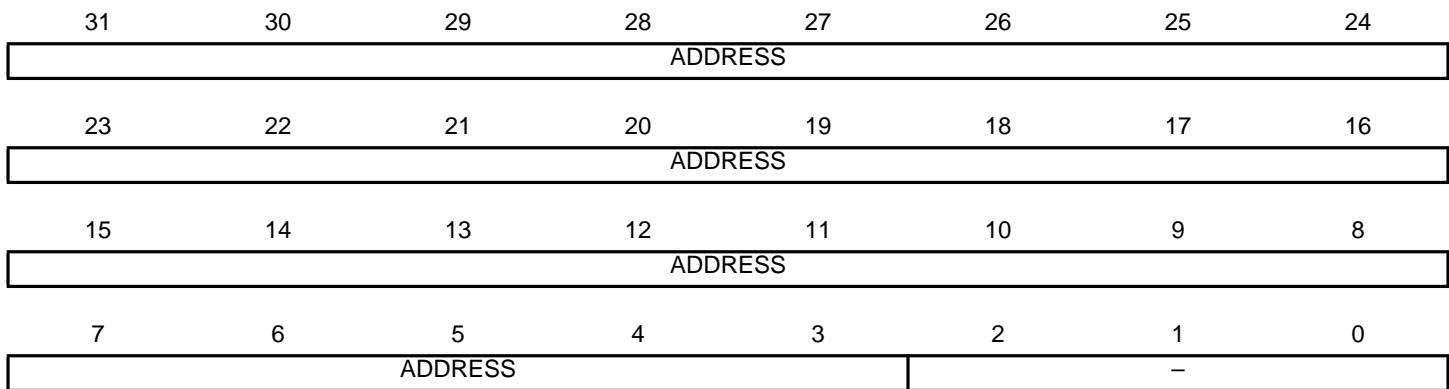
- **LSPACT: Lazy State Preservation Active**

0: The lazy state preservation is not active.

1: The lazy state preservation is active. The floating-point stack frame has been allocated but saving the state to it has been deferred.

### 12.12.2.3 Floating-point Context Address Register

**Name:** FPCAR  
**Access:** Read-write  
**Reset:** 0x0000000000



The FPCAR register holds the location of the unpopulated floating-point register space allocated on an exception stack frame.

- **ADDRESS**

The location of the unpopulated floating-point register space allocated on an exception stack frame.

#### 12.12.2.4 Floating-point Status Control Register

**Name:** FPSCR

**Access:** Read-write

**Reset:** 0x000000000

31	30	29	28	27	26	25	24
N	Z	C	V	-	AHP	DN	FZ
23	22	21	20	19	18	17	16
RMode				-			
15	14	13	12	11	10	9	8
				-			
7	6	5	4	3	2	1	0
IDC		-	IXC	UFC	OFC	DZC	IOC

The FPSCR register provides all necessary User level control of the floating-point system.

- **N: Negative Condition Code Flag**

Floating-point comparison operations update this flag.

- **Z: Zero Condition Code Flag**

Floating-point comparison operations update this flag.

- **C: Carry Condition Code Flag**

Floating-point comparison operations update this flag.

- **V: Overflow Condition Code Flag**

Floating-point comparison operations update this flag.

- **AHP: Alternative Half-precision Control**

0: IEEE half-precision format selected.

1: Alternative half-precision format selected.

- **DN: Default NaN Mode Control**

0: NaN operands propagate through to the output of a floating-point operation.

1: Any operation involving one or more NaNs returns the Default NaN.

- **FZ: Flush-to-zero Mode Control**

0: Flush-to-zero mode disabled. The behavior of the floating-point system is fully compliant with the IEEE 754 standard.

1: Flush-to-zero mode enabled.

- **RMode: Rounding Mode Control**

The encoding of this field is:

0b00 Round to Nearest (RN) mode

0b01 Round towards Plus Infinity (RP) mode.

0b10 Round towards Minus Infinity (RM) mode.

0b11 Round towards Zero (RZ) mode.

The specified rounding mode is used by almost all floating-point instructions.

- **IDC: Input Denormal Cumulative Exception**

IDC is a cumulative exception bit for floating-point exception; see also bits [4:0].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **IXC: Inexact Cumulative Exception**

IXC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **UFC: Underflow Cumulative Exception**

UFC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **OFC: Overflow Cumulative Exception**

OFC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **DZC: Division by Zero Cumulative Exception**

DZC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **IOC: Invalid Operation Cumulative Exception**

IOC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

### 12.12.2.5 Floating-point Default Status Control Register

**Name:** FPDSCR  
**Access:** Read-write  
**Reset:** 0x0000000000

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

The FPDSCR register holds the default values for the floating-point status control data.

- **AHP**

Default value for FPSCR.AHP.

- **DN**

Default value for FPSCR.DN.

- **FZ**

Default value for FPSCR.FZ.

- **RMode**

Default value for FPSCR.RMode.

## 12.13 Glossary

This glossary describes some of the terms used in technical documents from ARM.

Abort	A mechanism that indicates to a processor that the value associated with a memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory.
Aligned	A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.
Banked register	A register that has multiple physical copies, where the state of the processor determines which copy is used. The Stack Pointer, SP (R13) is a banked register.
Base register	In instruction descriptions, a register specified by a load or store instruction that is used to hold the base value for the instruction's address calculation. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the address that is sent to memory. <i>See also “Index register”</i>
Big-endian (BE)	Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory. <i>See also “Byte-invariant” , “Endianness” , “Little-endian (LE)” .</i>
Big-endian memory	Memory in which: a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address, a byte at a halfword-aligned address is the most significant byte within the halfword at that address. <i>See also “Little-endian memory” .</i>
Breakpoint	A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is to be halted. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints are removed after the program is successfully tested.

Byte-invariant	In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access. An ARM byte-invariant implementation also supports unaligned halfword and word memory accesses. It expects multi-word accesses to be word-aligned.
Cache	A block of on-chip or off-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used instructions, data, or instructions and data. This is done to greatly increase the average speed of memory accesses and so improve processor performance.
Condition field	A four-bit field in an instruction that specifies a condition under which the instruction can execute.
Conditional execution	If the condition code flags indicate that the corresponding condition is true when the instruction starts executing, it executes normally. Otherwise, the instruction does nothing.
Context	The environment that each process operates in for a multitasking operating system. In ARM processors, this is limited to mean the physical address range that it can access in memory and the associated memory access permissions.
Coprocessor	A processor that supplements the main processor. Cortex-M4 does not support any coprocessors.
Debugger	A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.
Direct Memory Access (DMA)	An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.
Doubleword	A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.
Doubleword-aligned	A data item having a memory address that is divisible by eight.
Endianness	Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping. <i>See also “Little-endian (LE)” and “Big-endian (BE)”</i>

## Exception

An event that interrupts program execution. When an exception occurs, the processor suspends the normal program flow and starts execution at the address indicated by the corresponding exception vector. The indicated address contains the first instruction of the handler for the exception.

An exception can be an interrupt request, a fault, or a software-generated system exception. Faults include attempting an invalid memory access, attempting to execute an instruction in an invalid processor state, and attempting to execute an undefined instruction.

## Exception service routine

See “[Interrupt handler](#)” .

## Exception vector

See “[Interrupt vector](#)” .

## Flat address mapping

A system of organizing memory in which each physical address in the memory space is the same as the corresponding virtual address.

## Halfword

A 16-bit data item.

## Illegal instruction

An instruction that is architecturally Undefined.

## Implementation-defined

The behavior is not architecturally defined, but is defined and documented by individual implementations.

## Implementation-specific

The behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

## Index register

In some load and store instruction descriptions, the value of this register is used as an offset to be added to or subtracted from the base register value to form the address that is sent to memory. Some addressing modes optionally enable the index register value to be shifted prior to the addition or subtraction.

See also “[Base register](#)” .

## Instruction cycle count

The number of cycles that an instruction occupies the Execute stage of the pipeline.

## Interrupt handler

A program that control of the processor is passed to when an interrupt occurs.

## Interrupt vector

One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.

Little-endian (LE)	Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory.  See also “ <a href="#">Big-endian (BE)</a> ” , “ <a href="#">Byte-invariant</a> ” , “ <a href="#">Endianness</a> ” .
Little-endian memory	Memory in which: a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address, a byte at a halfword-aligned address is the least significant byte within the halfword at that address.  See also “ <a href="#">Big-endian memory</a> ” .
Load/store architecture	A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.
Memory Protection Unit (MPU)	Hardware that controls access permissions to blocks of memory. An MPU does not perform any address translation.
Prefetching	In pipelined processors, the process of fetching instructions from memory to fill up the pipeline before the preceding instructions have finished executing. Prefetching an instruction does not mean that the instruction has to be executed.
Preserved	Preserved by writing the same value back that has been previously read from the same field on the same processor.
Read	Reads are defined as memory operations that have the semantics of a load. Reads include the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP.
Region	A partition of memory space.
Reserved	A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.
Thread-safe	In a multi-tasking environment, thread-safe functions use safeguard mechanisms when accessing shared resources, to ensure correct operation without the risk of shared access conflicts.
Thumb instruction	One or two halfwords that specify an operation for a processor to perform. Thumb instructions must be halfword-aligned.

Unaligned	A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.
Undefined	Indicates an instruction that generates an Undefined instruction exception.
Unpredictable	One cannot rely on the behavior. Unpredictable behavior must not represent security holes. Unpredictable behavior must not halt or hang the processor, or any parts of the system.
Warm reset	Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if debugging features of a processor.
WA	See " <a href="#">Write-allocate (WA)</a> " .
WB	See " <a href="#">Write-back (WB)</a> " .
Word	A 32-bit data item.
Write	Writes are defined as operations that have the semantics of a store. Writes include the Thumb instructions STM, STR, STRH, STRB, and PUSH.
Write-allocate (WA)	In a write-allocate cache, a cache miss on storing data causes a cache line to be allocated into the cache.
Write-back (WB)	In a write-back cache, data is only written to main memory when it is forced out of the cache on line replacement following a cache miss. Otherwise, writes by the processor only update the cache. This is also known as copyback.
Write buffer	A block of high-speed memory, arranged as a FIFO buffer, between the data cache and main memory, whose purpose is to optimize stores to main memory.
Write-through (WT)	In a write-through cache, data is written to main memory at the same time as the cache is updated.

## 13. Debug and Test Features

### 13.1 Description

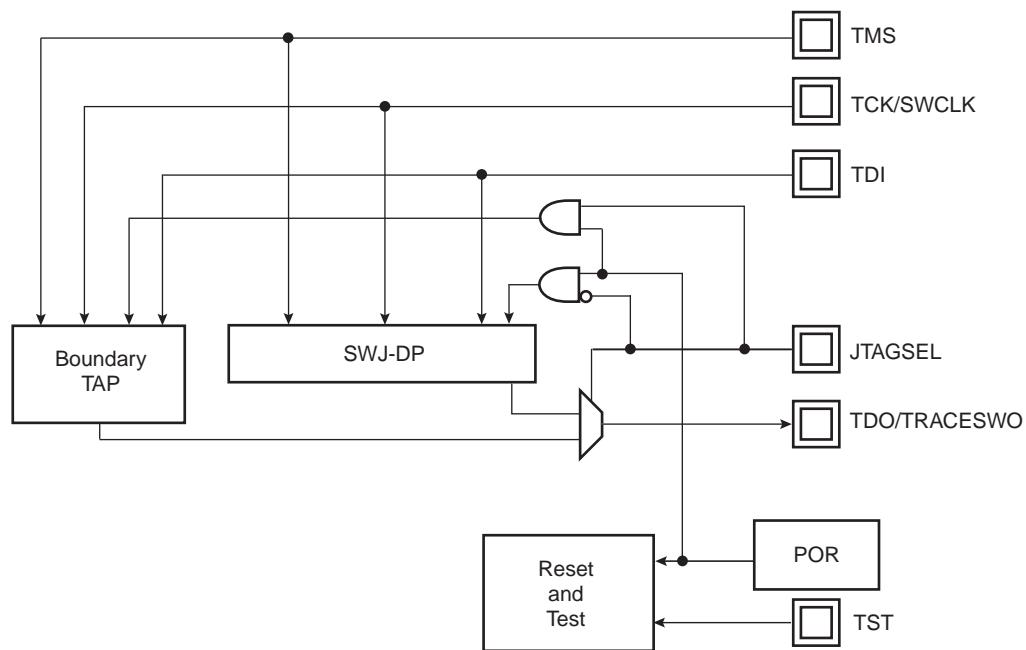
The SAM4 Series Microcontrollers feature a number of complementary debug and test capabilities. The Serial Wire/JTAG Debug Port (SWJ-DP) combining a Serial Wire Debug Port (SW-DP) and JTAG Debug (JTAG-DP) port is used for standard debugging functions, such as downloading code and single-stepping through programs. It also embeds a serial wire trace.

### 13.2 Embedded Characteristics

- Debug access to all memory and registers in the system, including Cortex-M4 register bank when the core is running, halted, or held in reset.
- Serial Wire Debug Port (SW-DP) and Serial Wire JTAG Debug Port (SWJ-DP) debug access
- Flash Patch and Breakpoint (FPB) unit for implementing breakpoints and code patches
- Data Watchpoint and Trace (DWT) unit for implementing watchpoints, data tracing, and system profiling
- Instrumentation Trace Macrocell (ITM) for support of printf style debugging
- IEEE1149.1 JTAG Boundary-can on All Digital Pins

### 13.3 Debug and Test Block Diagram

Figure 13-1. Debug and Test Block Diagram

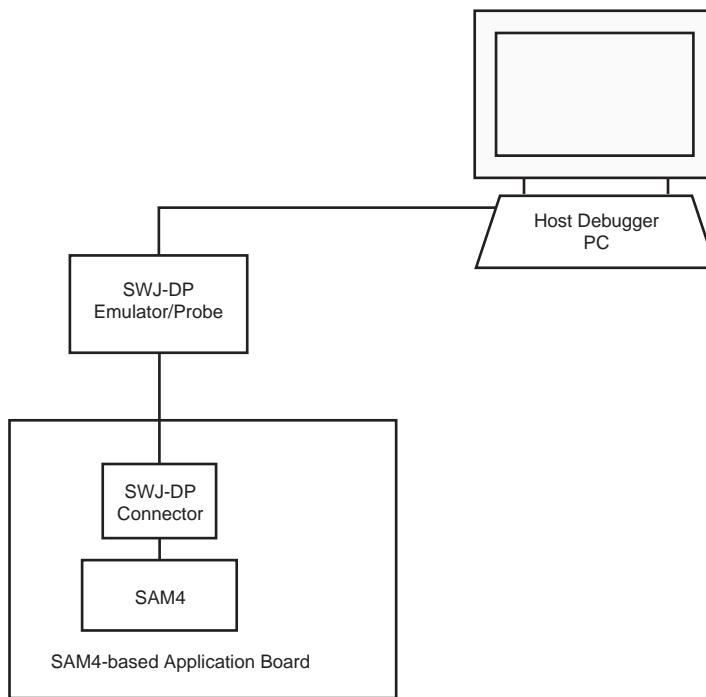


## 13.4 Application Examples

### 13.4.1 Debug Environment

Figure 13-2 shows a complete debug environment example. The SWJ-DP interface is used for standard debugging functions, such as downloading code and single-stepping through the program and viewing core and peripheral registers.

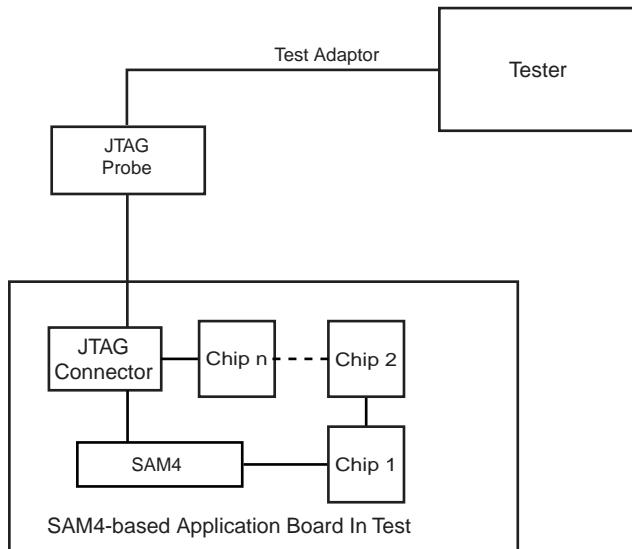
Figure 13-2. Application Debug Environment Example



### 13.4.2 Test Environment

Figure 13-3 shows a test environment example (JTAG Boundary scan). Test vectors are sent and interpreted by the tester. In this example, the “board in test” is designed using a number of JTAG-compliant devices. These devices can be connected to form a single scan chain.

**Figure 13-3. Application Test Environment Example**



## 13.5 Debug and Test Pin Description

**Table 13-1. Debug and Test Signal List**

Signal Name	Function	Type	Active Level
<b>Reset/Test</b>			
NRST	Microcontroller Reset	Input/Output	Low
TST	Test Select	Input	
<b>SWD/JTAG</b>			
TCK/SWCLK	Test Clock/Serial Wire Clock	Input	
TDI	Test Data In	Input	
TDO/TRACESWO	Test Data Out/Trace Asynchronous Data Out	Output	
TMS/SWDIO	Test Mode Select/Serial Wire Input/Output	Input	
JTAGSEL	JTAG Selection	Input	High

## 13.6 Functional Description

### 13.6.1 Test Pin

The TST pin is used for JTAG Boundary Scan Manufacturing Test or Fast Flash programming mode of the SAM4E series. The TST pin integrates a permanent pull-down resistor of about  $15\text{ k}\Omega$  to GND, so that it can be left unconnected for normal operations. To enter fast programming mode, see the Fast Flash Programming Interface (FFPI) section. For more on the manufacturing and test mode, refer to the “Debug and Test” section of the product datasheet.

### 13.6.2 NRST Pin

The NRST pin is bidirectional. It is handled by the on-chip reset controller and can be driven low to provide a reset signal to the external components or asserted low externally to reset the microcontroller. It will reset the Core and the peripherals except the Backup region (RTC, RTT and Supply Controller). There is no constraint on the length of the reset pulse and the reset controller can guarantee a minimum pulse length. The NRST pin integrates a permanent pull-up resistor to VDDIO of about  $100\text{ k}\Omega$ . By default, the NRST pin is configured as an input.

### 13.6.3 ERASE Pin

The ERASE pin is used to reinitialize the Flash content (and some of its NVM bits) to an erased state (all bits read as logic level 1). It integrates a pull-down resistor of about  $100\text{ k}\Omega$  to GND, so that it can be left unconnected for normal operations.

This pin is debounced by SCLK to improve the glitch tolerance. When the ERASE pin is tied high during less than 100 ms, it is not taken into account. The pin must be tied high during more than 220 ms to perform a Flash erase operation.

The ERASE pin is a system I/O pin and can be used as a standard I/O. At startup, the ERASE pin is not configured as a PIO pin. If the ERASE pin is used as a standard I/O, start-up level of this pin must be low to prevent unwanted erasing. Also, if the ERASE pin is used as a standard I/O output, asserting the pin to low does not erase the Flash. For details, please refer to the “Peripheral Signal Multiplexing on I/O Lines” section of this datasheet.

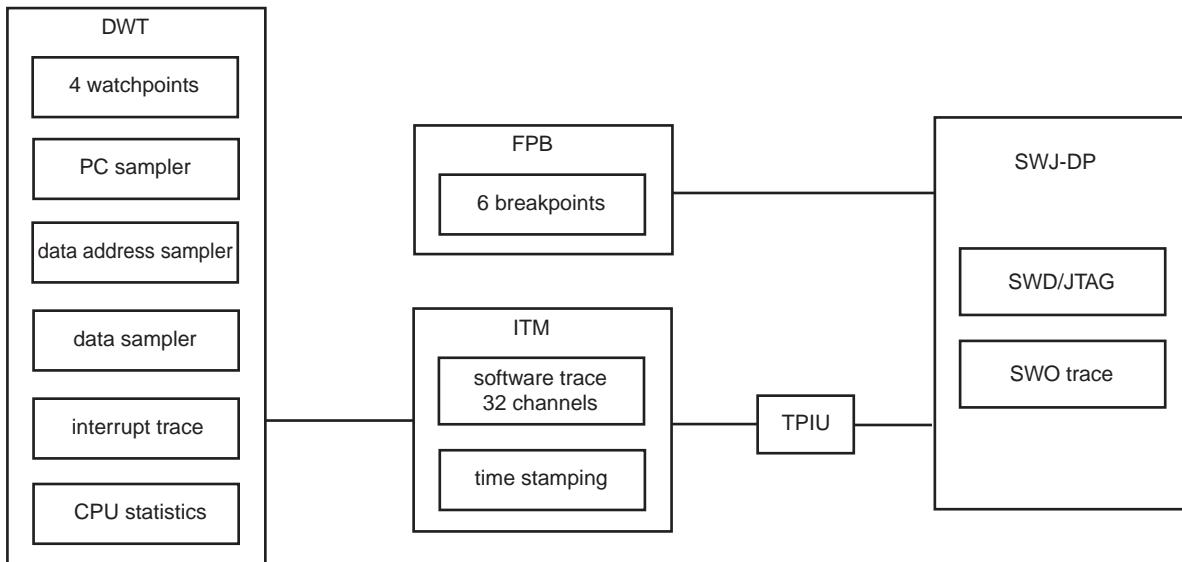
### 13.6.4 Debug Architecture

Figure 13-4 shows the Debug Architecture used in the SAM4. The Cortex-M4 embeds four functional units for debug:

- SWJ-DP (Serial Wire/JTAG Debug Port)
- FPB (Flash Patch Breakpoint)
- DWT (Data Watchpoint and Trace)
- ITM (Instrumentation Trace Macrocell)
- TPIU (Trace Port Interface Unit)

The debug architecture information that follows is mainly dedicated to developers of SWJ-DP Emulators/Probes and debugging tool vendors for Cortex-M4 based microcontrollers. For further details on SWJ-DP see the Cortex-M4 technical reference manual.

**Figure 13-4. Debug Architecture**



### 13.6.5 Serial Wire JTAG Debug Port (SWJ-DP) Pins

The SWJ-DP pins are TCK/SWCLK, TMS/SWDIO, TDO/SWO, TDI and commonly provided on a standard 20-pin JTAG connector defined by ARM. For more details about voltage reference and reset state, please refer to the “Signal Description” section of this datasheet.

At start-up, SWJ-DP pins are configured in SWJ-DP mode to allow connection with debugging probe. Please refer to the “Debug and Test” section of the product datasheet.

SWJ-DP pins can be used as standard I/Os to provide users more general input/output pins when the debug port is not needed in the end application. Mode selection between SWJ-DP mode (System IO mode) and general IO mode is performed through the AHB Matrix Special Function Registers (MATRIX\_SFR). Configuration of the pad for pull-up, triggers, debouncing and glitch filters is possible regardless of the mode.

The JTAGSEL pin is used to select the JTAG boundary scan when asserted at a high level. It integrates a permanent pull-down resistor of about 15 kΩ to GND, so that it can be left unconnected for normal operations.

By default, the JTAG Debug Port is active. If the debugger host wants to switch to the Serial Wire Debug Port, it must provide a dedicated JTAG sequence on TMS/SWDIO and TCK/SWCLK which disables the JTAG-DP and enables the SW-DP. When the Serial Wire Debug Port is active, TDO/TRACESWO can be used for trace.

The asynchronous TRACE output (TRACESWO) is multiplexed with TDO. So the asynchronous trace can only be used with SW-DP, not JTAG-DP. For more information about SW-DP and JTAG-DP switching, please refer to the “Debug and Test” section of the datasheet.

**Table 13-2. SWJ-DP Pin List**

Pin Name	JTAG Port	Serial Wire Debug Port
TMS/SWDIO	TMS	SWDIO
TCK/SWCLK	TCK	SWCLK
TDI	TDI	-
TDO/TRACESWO	TDO	TRACESWO (optional: trace)

SW-DP or JTAG-DP mode is selected when JTAGSEL is low. It is not possible to switch directly between SWJ-DP and JTAG boundary scan operations. A chip reset must be performed after JTAGSEL is changed.

### 13.6.5.1 SW-DP and JTAG-DP Selection Mechanism

Debug port selection mechanism is done by sending specific **SWDIOTMS** sequence. The JTAG-DP is selected by default after reset.

- Switch from JTAG-DP to SW-DP. The sequence is:
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
  - Send the 16-bit sequence on **SWDIOTMS** = 0111100111100111 (0x79E7 MSB first)
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
- Switch from SWD to JTAG. The sequence is:
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
  - Send the 16-bit sequence on **SWDIOTMS** = 0011110011100111 (0x3CE7 MSB first)
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1

### 13.6.6 FPB (Flash Patch Breakpoint)

The FPB:

- Implements hardware breakpoints
- Patches code and data from code space to system space.

The FPB unit contains:

- Two literal comparators for matching against literal loads from Code space, and remapping to a corresponding area in System space.
- Six instruction comparators for matching against instruction fetches from Code space and remapping to a corresponding area in System space.
- Alternatively, comparators can also be configured to generate a Breakpoint instruction to the processor core on a match.

### 13.6.7 DWT (Data Watchpoint and Trace)

The DWT contains four comparators which can be configured to generate the following:

- PC sampling packets at set intervals
- PC or Data watchpoint packets
- Watchpoint event to halt core

The DWT contains counters for the items that follow:

- Clock cycle (CYCCNT)
- Folded instructions
- Load Store Unit (LSU) operations
- Sleep Cycles
- CPI (all instruction cycles except for the first cycle)
- Interrupt overhead

### 13.6.8 ITM (Instrumentation Trace Macrocell)

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 by three different sources with several priority levels:

- **Software trace:** Software can write directly to ITM stimulus registers. This can be done thanks to the “print” function. For more information, refer to [Section 13.6.8.1 “How to Configure the ITM”](#).
- **Hardware trace:** The ITM emits packets generated by the DWT.
- **Time stamping:** Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp.

### 13.6.8.1 How to Configure the ITM

The following example describes how to output trace data in asynchronous trace mode.

- Configure the TPIU for asynchronous trace mode (refer to [Section 13.6.8.3 “5.4.3. How to Configure the TPIU”](#))
- Enable the write accesses into the ITM registers by writing “0xC5ACCE55” into the Lock Access Register (Address: 0xE0000FB0)
- Write 0x00010015 into the Trace Control Register:
  - Enable ITM
  - Enable Synchronization packets
  - Enable SWO behavior
  - Fix the ATB ID to 1
- Write 0x1 into the Trace Enable Register:
  - Enable the Stimulus port 0
- Write 0x1 into the Trace Privilege Register:
  - Stimulus port 0 only accessed in privileged mode (Clearing a bit in this register will result in the corresponding stimulus port being accessible in user mode.)
- Write into the Stimulus port 0 register: TPIU (Trace Port Interface Unit)

The TPIU acts as a bridge between the on-chip trace data and the Instruction Trace Macrocell (ITM).

The TPIU formats and transmits trace data off-chip at frequencies asynchronous to the core.

### 13.6.8.2 Asynchronous Mode

The TPIU is configured in asynchronous mode, trace data are output using the single TRACESWO pin. The TRACESWO signal is multiplexed with the TDO signal of the JTAG Debug Port. As a consequence, asynchronous trace mode is only available when the Serial Wire Debug mode is selected since TDO signal is used in JTAG debug mode.

Two encoding formats are available for the single pin output:

- Manchester encoded stream. This is the reset value.
- NRZ\_based UART byte structure

### 13.6.8.3 5.4.3. How to Configure the TPIU

This example only concerns the asynchronous trace mode.

- Set the TRCENA bit to 1 into the Debug Exception and Monitor Register (0xE000EDFC) to enable the use of trace and debug blocks.
- Write 0x2 into the Selected Pin Protocol Register
  - Select the Serial Wire Output – NRZ
- Write 0x100 into the Formatter and Flush Control Register
- Set the suitable clock prescaler value into the Async Clock Prescaler Register to scale the baud rate of the asynchronous output (this can be done automatically by the debugging tool).

### 13.6.9 IEEE® 1149.1 JTAG Boundary Scan

IEEE 1149.1 JTAG Boundary Scan allows pin-level access independent of the device packaging technology.

IEEE 1149.1 JTAG Boundary Scan is enabled when TST is tied to low, while JTAGSEL is high during power-up, and must be kept in this state during the whole boundary scan operation. The SAMPLE, EXTEST and BYPASS functions are implemented. In SWD/JTAG debug mode, the ARM processor responds with a non-JTAG chip ID that identifies the processor. This is not IEEE 1149.1 JTAG-compliant.

It is not possible to switch directly between JTAG Boundary Scan and SWJ Debug Port operations. A chip reset must be performed after JTAGSEL is changed.

A Boundary-scan Descriptor Language (BSDL) file to set up the test is provided on the [Atmel website at http://www.atmel.com](#).

### 13.6.9.1 JTAG Boundary-scan Register

The Boundary-scan Register (BSR) contains a number of bits which correspond to active pins and associated control signals.

Each SAM4 input/output pin corresponds to a 3-bit register in the BSR. The OUTPUT bit contains data that can be forced on the pad. The INPUT bit facilitates the observability of data applied to the pad. The CONTROL bit selects the direction of the pad.

For more information, please refer to BDSL files available for the SAM4 Series.

### 13.6.10 ID Code Register

Access: Read-only

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

- **VERSION[31:28]: Product Version Number**

Set to 0x0.

- **PART NUMBER[27:12]: Product Part Number**

Chip Name	Chip ID
SAM4E	0xA3CC_0CE0

- **MANUFACTURER IDENTITY[11:1]**

Set to 0x01F.

- **Bit[0] Required by IEEE Std. 1149.1.**

Set to 0x1.

Chip Name	JTAG ID Code
SAM4E	0x05B3_703F

## 14. Chip Identifier (CHIPID)

### 14.1 Description

Chip Identifier (CHIPID) registers permit recognition of the device and its revision. These registers provide the sizes and types of the on-chip memories, as well as the set of embedded peripherals.

Two chip identifier registers are embedded: CHIPID\_CIDR (Chip ID Register) and CHIPID\_EXID (Extension ID). Both registers contain a hard-wired value that is read-only. The first register contains the following fields:

- EXT - shows the use of the extension identifier register
- NVPTYP and NVPSIZ - identifies the type of embedded non-volatile memory and its size
- ARCH - identifies the set of embedded peripherals
- SRAMSIZ - indicates the size of the embedded SRAM
- EPROC - indicates the embedded ARM processor
- VERSION - gives the revision of the silicon

The second register is device-dependent and reads 0 if the bit EXT is 0.

### 14.2 Embedded Characteristics

- Chip ID Registers
  - Identification of the Device Revision, Sizes of the Embedded Memories, Set of Peripherals, Embedded Processor

Table 14-1. SAM4E Chip IDs Registers

Chip Name	CHIPID_CIDR	CHIPID_EXID
SAM4E16E	0xA3CC_0CE0	0x0012_0200
SAM4E8E	0xA3CC_0CE0	0x0012_0208
SAM4E16C	0xA3CC_0CE0	0x0012_0201
SAM4E8C	0xA3CC_0CE0	0x0012_0209

## 14.3 Power Management Controller (PMC) User Interface

Table 14-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0	Chip ID Register	CHIPID_CIDR	Read-only	—
0x4	Chip ID Extension Register	CHIPID_EXID	Read-only	—

### 14.3.1 Chip ID Register

**Name:** CHIPID\_CIDR

**Address:** 0x400E0740

**Access:** Read-only

31	30	29	28	27	26	25	24
EXT	NVPTYP						
23	22	21	20	19	18	17	16
ARCH				SRAMSIZ			
15	14	13	12	11	10	9	8
NVPSIZ2				NVPSIZ			
7	6	5	4	3	2	1	0
EPROC				VERSION			

- **VERSION: Version of the Device**

Current version of the device.

- **EPROC: Embedded Processor**

Value	Name	Description
1	ARM946ES	ARM946ES
2	ARM7TDMI	ARM7TDMI
3	CM3	Cortex-M3
4	ARM920T	ARM920T
5	ARM926EJS	ARM926EJS
6	CA5	Cortex-A5
7	CM4	Cortex-M4

- **NVPSIZ: Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8 Kbytes
2	16K	16 Kbytes
3	32K	32 Kbytes
4	–	Reserved
5	64K	64 Kbytes
6	–	Reserved
7	128K	128 Kbytes
8	–	Reserved
9	256K	256 Kbytes
10	512K	512 Kbytes
11	–	Reserved
12	1024K	1024 Kbytes

Value	Name	Description
13	–	Reserved
14	2048K	2048 Kbytes
15	–	Reserved

- **NVPSIZ2: Second Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8 Kbytes
2	16K	16 Kbytes
3	32K	32 Kbytes
4	–	Reserved
5	64K	64 Kbytes
6	–	Reserved
7	128K	128 Kbytes
8	–	Reserved
9	256K	256 Kbytes
10	512K	512 Kbytes
11	–	Reserved
12	1024K	1024 Kbytes
13	–	Reserved
14	2048K	2048 Kbytes
15	–	Reserved

- **SRAMSIZ: Internal SRAM Size**

Value	Name	Description
0	48K	48 Kbytes
1	192K	192 Kbytes
2	2K	2 Kbytes
3	6K	6 Kbytes
4	24K	24 Kbytes
5	4K	4 Kbytes
6	80K	80 Kbytes
7	160K	160 Kbytes
8	8K	8 Kbytes
9	16K	16 Kbytes
10	32K	32 Kbytes
11	64K	64 Kbytes
12	128K	128 Kbytes

Value	Name	Description
13	256K	256 Kbytes
14	96K	96 Kbytes
15	512K	512 Kbytes

- **ARCH: Architecture Identifier**

Value	Name	Description
0x19	AT91SAM9xx	AT91SAM9xx Series
0x29	AT91SAM9XExx	AT91SAM9XExx Series
0x34	AT91x34	AT91x34 Series
0x37	CAP7	CAP7 Series
0x39	CAP9	CAP9 Series
0x3B	CAP11	CAP11 Series
0x3C	SAM4E	SAM4E Series
0x40	AT91x40	AT91x40 Series
0x42	AT91x42	AT91x42 Series
0x45	AT91SAM4SH2	AT91SAM4SH2 Series
0x55	AT91x55	AT91x55 Series
0x60	AT91SAM7Axx	AT91SAM7Axx Series
0x61	AT91SAM7AQxx	AT91SAM7AQxx Series
0x63	AT91x63	AT91x63 Series
0x64	SAM4CxC	SAM4CxC Series (100-pin version)
0x70	AT91SAM7Sxx	AT91SAM7Sxx Series
0x71	AT91SAM7XCxx	AT91SAM7XCxx Series
0x72	AT91SAM7SExx	AT91SAM7SExx Series
0x73	AT91SAM7Lxx	AT91SAM7Lxx Series
0x75	AT91SAM7XXX	AT91SAM7XXX Series
0x76	AT91SAM7SLxx	AT91SAM7SLxx Series
0x80	SAM3UxC	SAM3UxC Series (100-pin version)
0x81	SAM3UxE	SAM3UxE Series (144-pin version)
0x83	SAM3AxC	SAM3AxC Series (100-pin version)
0x84	SAM3XxC	SAM3XxC Series (100-pin version)
0x85	SAM3XxE	SAM3XxE Series (144-pin version)
0x86	SAM3XxG	SAM3XxG Series (208/217-pin version)
0x92	AT91x92	AT91x92 Series
0x99	SAM3SDxB	SAM3SDxB Series (64-pin version)
0x9A	SAM3SDxC	SAM3SDxC Series (100-pin version)
0xA5	SAM5A	SAM5A
0xB0	SAM4LxA	SAM4LxA Series (48-pin version)

Value	Name	Description
0xB1	SAM4LxB	SAM4LxB Series (64-pin version)
0xB2	SAM4LxC	SAM4LxC Series (100-pin version)
0xF0	AT75Cxx	AT75Cxx Series

- **NVPTYP: Nonvolatile Program Memory Type**

Value	Name	Description
0	ROM	ROM
1	ROMLESS	ROMless or on-chip Flash
4	SRAM	SRAM emulating ROM
2	FLASH	Embedded Flash Memory
3	ROM_FLASH	ROM and Embedded Flash Memory <ul style="list-style-type: none"> <li>• NVPSIZ is ROM size</li> <li>• NVPSIZ2 is Flash size</li> </ul>

- **EXT: Extension Flag**

0 = Chip ID has a single register definition without extension

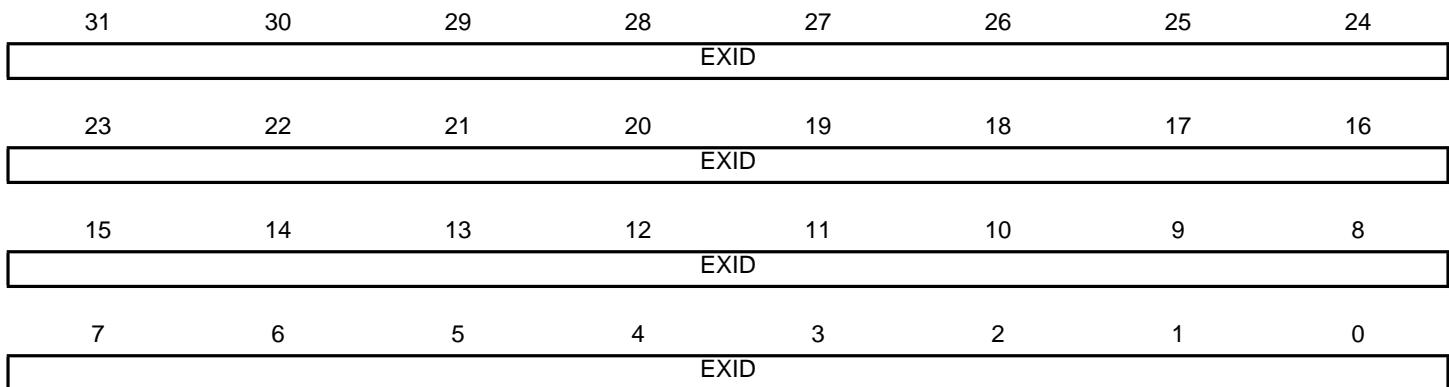
1 = An extended Chip ID exists.

#### 14.3.2 Chip ID Extension Register

**Name:** CHIPID\_EXID

**Address:** 0x400E0744

**Access:** Read-only



- **EXID: Chip ID Extension**

Reads 0 if the EXT bit in CHIPID\_CIDR is 0.

CHIPID\_EXID [1:0]: Package Type

Value	Name	Description
0	Package Type	Package 144
1	Package Type	Package 100

CHIPID\_EXID [4:2]: Flash Size

Value	Name	Description
0	Flash Size	1024 Kbytes
2	Flash Size	512 Kbytes

CHIPID\_EXID [31:5]: Product Number

Value	Name	Description
0x0012_020	Product Number	SAM4E Product Series



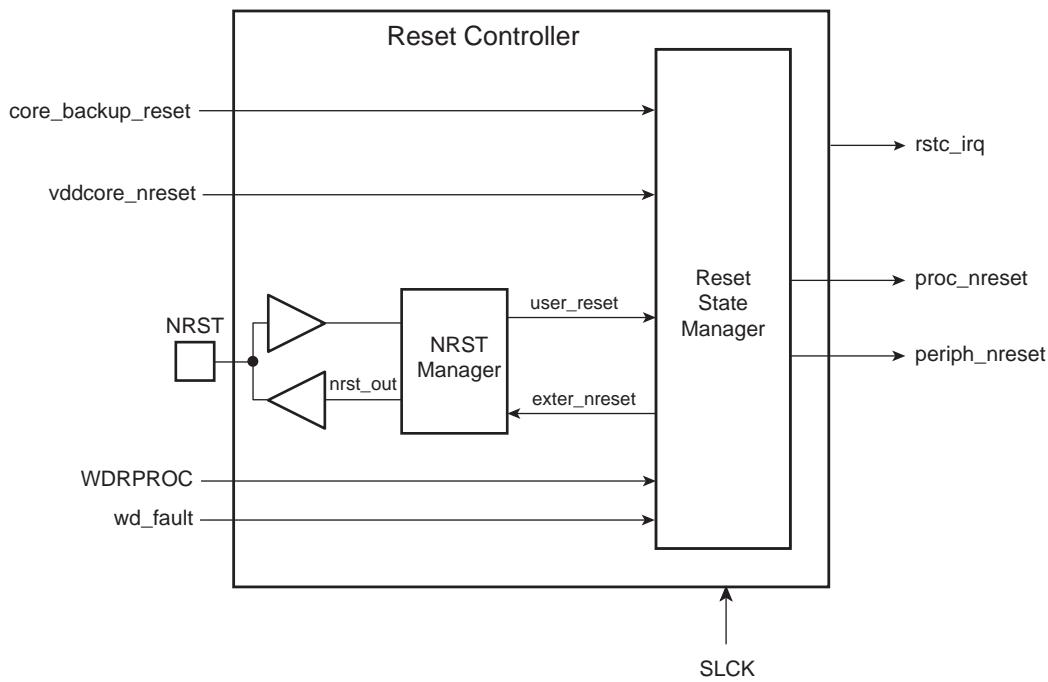
## 15. Reset Controller (RSTC)

### 15.1 Embedded Characteristics

- Manages all Resets of the System, Including
  - External Devices through the NRST Pin
  - Processor Reset
  - Peripheral Set Reset
- Based on Embedded Power-on Cell
- Reset Source Status
  - Status of the Last Reset
  - Either Software Reset, User Reset, Watchdog Reset
- External Reset Signal Shaping

### 15.2 Block Diagram

Figure 15-1. Reset Controller Block Diagram



### 15.3 Functional Description

#### 15.3.1 Reset Controller Overview

The Reset Controller is made up of an NRST Manager and a Reset State Manager. It runs at Slow Clock and generates the following reset signals:

- proc\_nreset: Processor reset line. It also resets the Watchdog Timer
- periph\_nreset: Affects the whole set of embedded peripherals
- nrst\_out: Drives the NRST pin

These reset signals are asserted by the Reset Controller, either on external events or on software action. The Reset State Manager controls the generation of reset signals and provides a signal to the NRST Manager when an assertion of the NRST pin is required.

The NRST Manager shapes the NRST assertion during a programmable time, thus controlling external device resets.

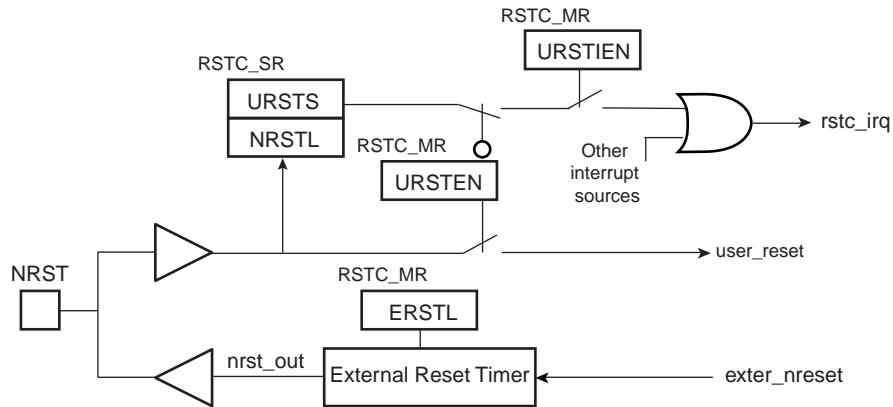
The Reset Controller Mode Register (RSTC\_MR), allowing the configuration of the Reset Controller, is powered with VDDIO, so that its configuration is saved as long as VDDIO is on.

### 15.3.2 NRST Manager

After power-up, NRST is an output during the ERSTL time period defined in the RSTC\_MR. When ERSTL has elapsed, the pin behaves as an input and all the system is held in reset if NRST is tied to GND by an external signal.

The NRST Manager samples the NRST input pin and drives this pin low when required by the Reset State Manager. Figure 15-2 shows the block diagram of the NRST Manager.

**Figure 15-2. NRST Manager**



#### 15.3.2.1 NRST Signal or Interrupt

The NRST Manager samples the NRST pin at Slow Clock speed. When the line is detected low, a User Reset is reported to the Reset State Manager.

However, the NRST Manager can be programmed to not trigger a reset when an assertion of NRST occurs. Writing the bit URSTEN at 0 in RSTC\_MR disables the User Reset trigger.

The level of the pin NRST can be read at any time in the bit NRSTL (NRST level) in RSTC\_SR. As soon as the pin NRST is asserted, the bit URSTS in RSTC\_SR is set. This bit clears only when RSTC\_SR is read.

The Reset Controller can also be programmed to generate an interrupt instead of generating a reset. To do so, the bit URSTIEN in RSTC\_MR must be written at 1.

#### 15.3.2.2 NRST External Reset Control

The Reset State Manager asserts the signal ext\_nreset to assert the NRST pin. When this occurs, the "nrst\_out" signal is driven low by the NRST Manager for a time programmed by the field ERSTL in RSTC\_MR. This assertion duration, named EXTERNAL\_RESET\_LENGTH, lasts  $2^{(ERSTL+1)}$  Slow Clock cycles. This gives the approximate duration of an assertion between 60 µs and 2 seconds. Note that ERSTL at 0 defines a two-cycle duration for the NRST pulse.

This feature allows the Reset Controller to shape the NRST pin level, and thus to guarantee that the NRST line is driven low for a time compliant with potential external devices connected on the system reset.

As the ERSTL field is within RSTC\_MR register, which is backed-up, it can be used to shape the system power-up reset for devices requiring a longer startup time than the Slow Clock Oscillator.

### 15.3.3 Brownout Manager

The Brownout manager is embedded within the Supply Controller, please refer to the product Supply Controller section for a detailed description.

### 15.3.4 Reset States

The Reset State Manager handles the different reset sources and generates the internal reset signals. It reports the reset status in the field RSTTYP of the Status Register (RSTC\_SR). The update of the field RSTTYP is performed when the processor reset is released.

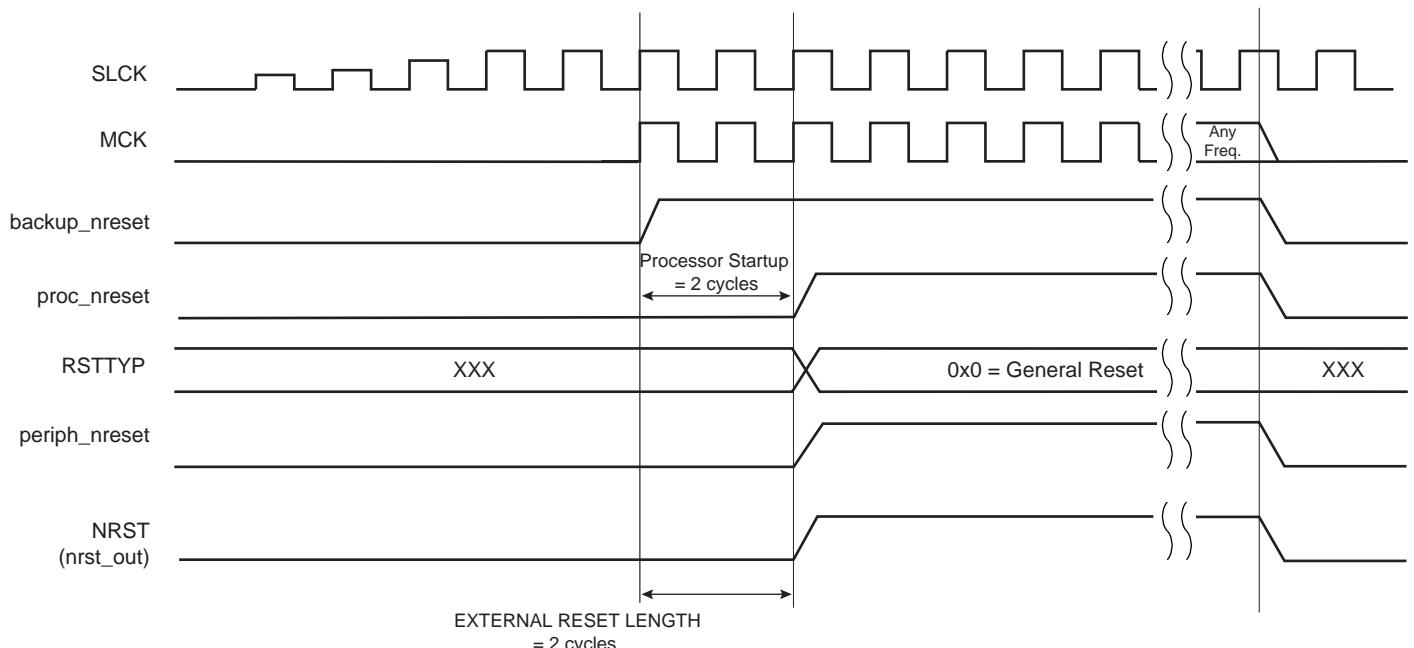
#### 15.3.4.1 General Reset

A general reset occurs when a Power-on-reset is detected, a Brownout or a Voltage regulation loss is detected by the Supply controller. The vddcore\_nreset signal is asserted by the Supply Controller when a general reset occurs.

All the reset signals are released and the field RSTTYP in RSTC\_SR reports a General Reset. As the RSTC\_MR is reset, the NRST line rises 2 cycles after the vddcore\_nreset, as ERSTL defaults at value 0x0.

Figure 15-3 shows how the General Reset affects the reset signals.

Figure 15-3. General Reset State



#### 15.3.4.2 Backup Reset

A Backup reset occurs when the chip returns from Backup Mode. The core\_backup\_reset signal is asserted by the Supply Controller when a Backup reset occurs.

The field RSTTYP in RSTC\_SR is updated to report a Backup Reset.

#### 15.3.4.3 User Reset

The User Reset is entered when a low level is detected on the NRST pin and the bit URSTEN in RSTC\_MR is at 1. The NRST input signal is resynchronized with SLCK to insure proper behavior of the system.

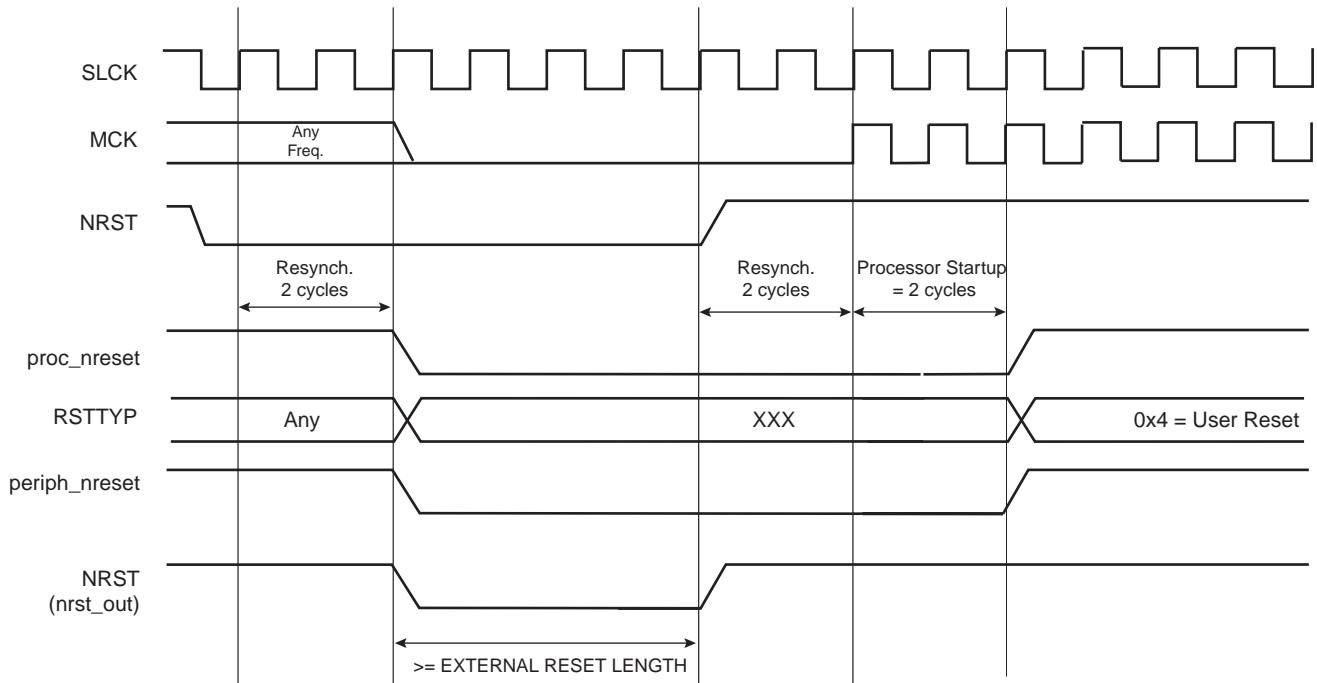
The User Reset is entered as soon as a low level is detected on NRST. The Processor Reset and the Peripheral Reset are asserted.

The User Reset is left when NRST rises, after a two-cycle resynchronization time and a 3-cycle processor startup. The processor clock is re-enabled as soon as NRST is confirmed high.

When the processor reset signal is released, the RSTTYP field of the Status Register (RSTC\_SR) is loaded with the value 0x4, indicating a User Reset.

The NRST Manager guarantees that the NRST line is asserted for EXTERNAL\_RESET\_LENGTH Slow Clock cycles, as programmed in the field ERSTL. However, if NRST does not rise after EXTERNAL\_RESET\_LENGTH because it is driven low externally, the internal reset lines remain asserted until NRST actually rises.

**Figure 15-4. User Reset State**



#### 15.3.4.4 Software Reset

The Reset Controller offers several commands used to assert the different reset signals. These commands are performed by writing the Control Register (RSTC\_CR) with the following bits at 1:

- PROCRST: Writing PROCRST at 1 resets the processor and the watchdog timer
- PERRST: Writing PERRST at 1 resets all the embedded peripherals including the memory system, and, in particular, the Remap Command. The Peripheral Reset is generally used for debug purposes.  
Except for debug purposes, PERRST must always be used in conjunction with PROCRST (PERRST and PROCRST set both at 1 simultaneously).
- EXTRST: Writing EXTRST at 1 asserts low the NRST pin during a time defined by the field ERSTL in the Mode Register (RSTC\_MR).

The software reset is entered if at least one of these bits is set by the software. All these commands can be performed independently or simultaneously. The software reset lasts 3 Slow Clock cycles.

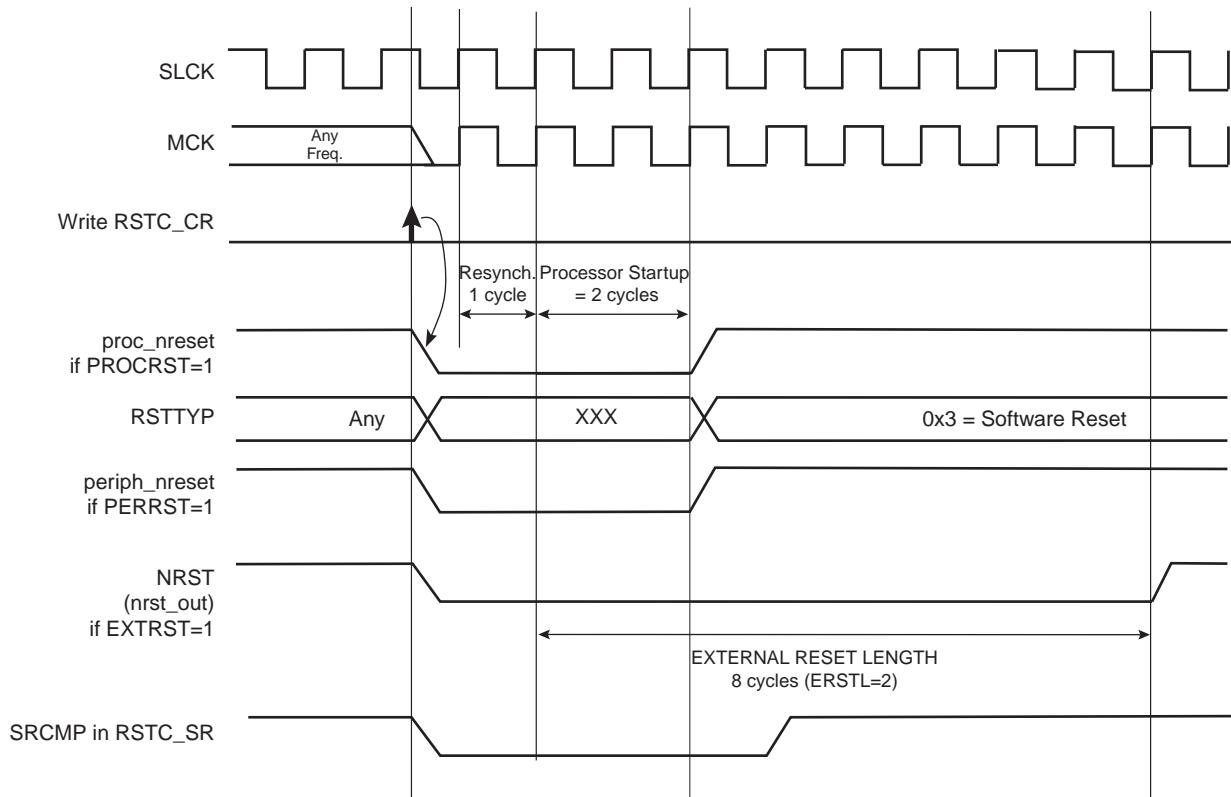
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset is left, i.e.; synchronously to SLCK.

If EXTRST is set, the nrst\_out signal is asserted depending on the programming of the field ERSTL. However, the resulting falling edge on NRST does not lead to a User Reset.

If and only if the PROCRST bit is set, the Reset Controller reports the software status in the field RSTTYP of the Status Register (RSTC\_SR). Other Software Resets are not reported in RSTTYP.

As soon as a software operation is detected, the bit SRCMP (Software Reset Command in Progress) is set in the Status Register (RSTC\_SR). It is cleared as soon as the software reset is left. No other software reset can be performed while the SRCMP bit is set, and writing any value in RSTC\_CR has no effect.

**Figure 15-5. Software Reset**



#### 15.3.4.5 Watchdog Reset

The Watchdog Reset is entered when a watchdog fault occurs. This state lasts 3 Slow Clock cycles.

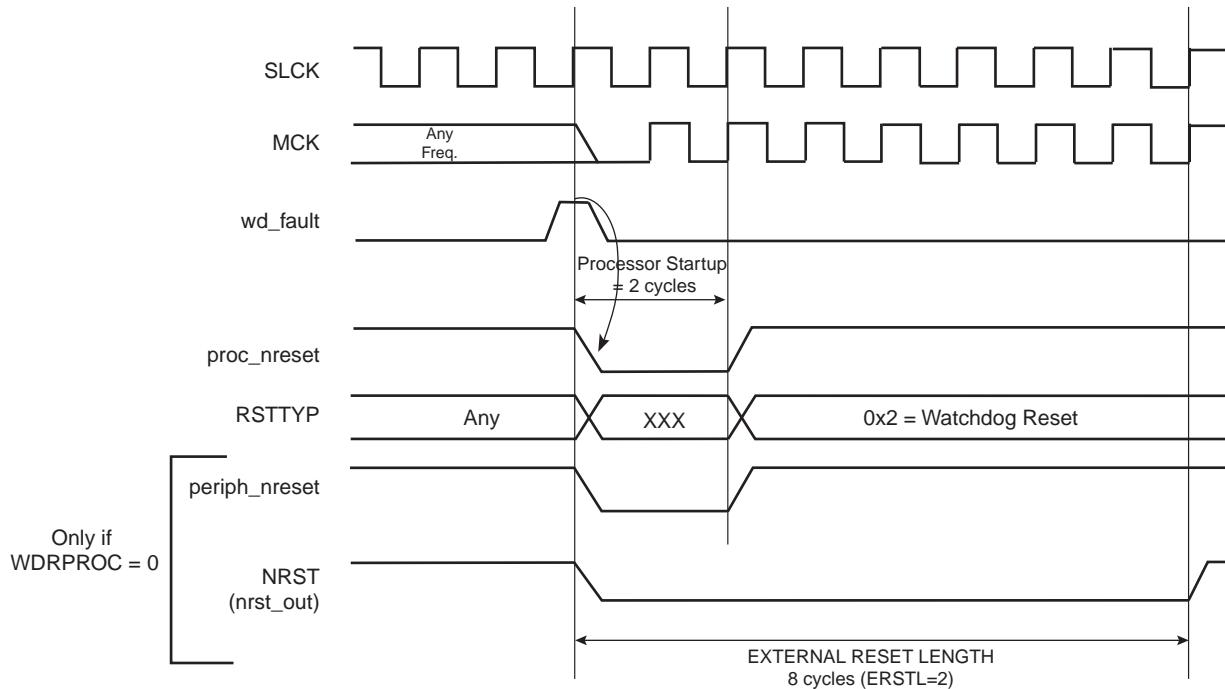
When in Watchdog Reset, assertion of the reset signals depends on the WDRPROC bit in WDT\_MR:

- If WDRPROC is 0, the Processor Reset and the Peripheral Reset are asserted. The NRST line is also asserted, depending on the programming of the field ERSTL. However, the resulting low level on NRST does not result in a User Reset state.
- If WDRPROC = 1, only the processor reset is asserted.

The Watchdog Timer is reset by the proc\_nreset signal. As the watchdog fault always causes a processor reset if WDRSTEN is set, the Watchdog Timer is always reset after a Watchdog Reset, and the Watchdog is enabled by default and with a period set to a maximum.

When the WDRSTEN in WDT\_MR bit is reset, the watchdog fault has no impact on the reset controller.

**Figure 15-6. Watchdog Reset**



### 15.3.5 Reset State Priorities

The Reset State Manager manages the following priorities between the different reset sources, given in descending order:

- General Reset
- Backup Reset
- Watchdog Reset
- Software Reset
- User Reset

Particular cases are listed below:

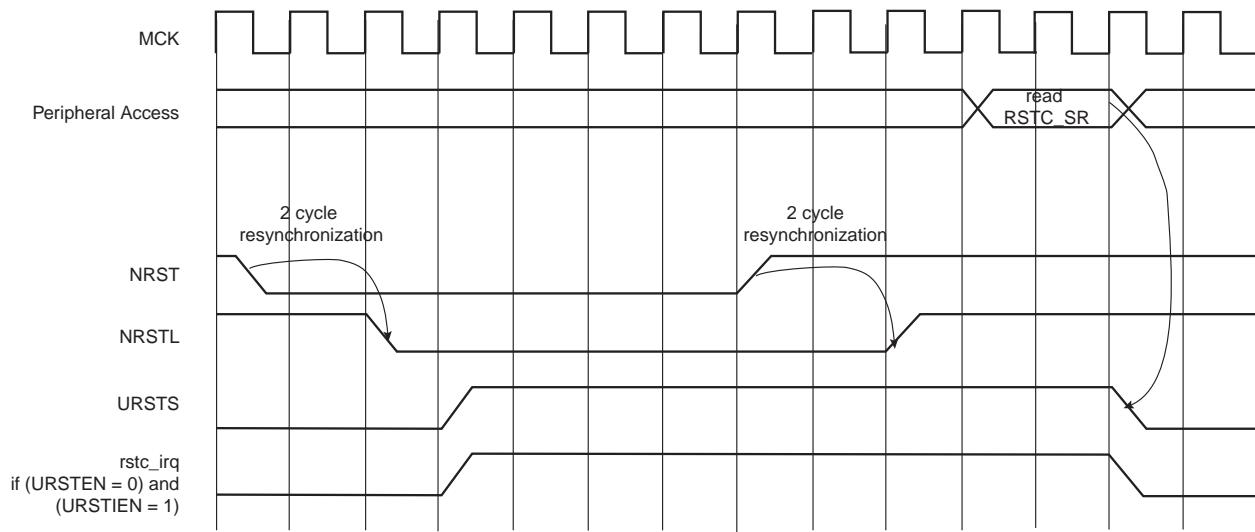
- When in User Reset:
  - A watchdog event is impossible because the Watchdog Timer is being reset by the proc\_nreset signal.
  - A software reset is impossible, since the processor reset is being activated.
- When in Software Reset:
  - A watchdog event has priority over the current state.
  - The NRST has no effect.
- When in Watchdog Reset:
  - The processor reset is active and so a Software Reset cannot be programmed.
  - A User Reset cannot be entered.

### 15.3.6 Reset Controller Status Register

The Reset Controller status register (RSTC\_SR) provides several status fields:

- RSTTYP field: This field gives the type of the last reset, as explained in previous sections.
- SRCMP bit: This field indicates that a Software Reset Command is in progress and that no further software reset should be performed until the end of the current one. This bit is automatically cleared at the end of the current software reset.
- NRSTL bit: The NRSTL bit of the Status Register gives the level of the NRST pin sampled on each MCK rising edge.
- URSTS bit: A high-to-low transition of the NRST pin sets the URSTS bit of the RSTC\_SR register. This transition is also detected on the Master Clock (MCK) rising edge (see [Figure 15-7](#)). If the User Reset is disabled (URSTEN = 0) and if the interruption is enabled by the URSTIEN bit in the RSTC\_MR register, the URSTS bit triggers an interrupt. Reading the RSTC\_SR status register resets the URSTS bit and clears the interrupt.

**Figure 15-7. Reset Controller Status and Interrupt**



## 15.4 Reset Controller (RSTC) User Interface

Table 15-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RSTC_CR	Write-only	-
0x04	Status Register	RSTC_SR	Read-only	0x0000_0000
0x08	Mode Register	RSTC_MR	Read-write	0x0000 0001

### 15.4.1 Reset Controller Control Register

**Name:** RSTC\_CR

**Address:** 0x400E1800

**Access:** Write-only

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

- **PROCRST: Processor Reset**

0 = No effect.

1 = If KEY is correct, resets the processor.

- **PERRST: Peripheral Reset**

0 = No effect.

1 = If KEY is correct, resets the peripherals.

- **EXTRST: External Reset**

0 = No effect.

1 = If KEY is correct, asserts the NRST pin and resets the processor and the peripherals.

- **KEY: System Reset Key**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

### 15.4.2 Reset Controller Status Register

**Name:** RSTC\_SR

**Address:** 0x400E1804

**Access:** Read-only

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

- **URSTS: User Reset Status**

0 = No high-to-low edge on NRST happened since the last read of RSTC\_SR.

1 = At least one high-to-low transition of NRST has been detected since the last read of RSTC\_SR.

- **RSTTYP: Reset Type**

Value	Name	Description
0	General Reset	First power-up Reset
1	Backup Reset	Return from Backup Mode
2	Watchdog Reset	Watchdog fault occurred
3	Software Reset	Processor reset required by the software
4	User Reset	NRST pin detected low

Reports the cause of the last processor reset. Reading this RSTC\_SR does not reset this field.

- **NRSTL: NRST Pin Level**

Registers the NRST Pin Level at Master Clock (MCK).

- **SRCMP: Software Reset Command in Progress**

0 = No software command is being performed by the reset controller. The reset controller is ready for a software command.

1 = A software reset command is being performed by the reset controller. The reset controller is busy.

### 15.4.3 Reset Controller Mode Register

**Name:** RSTC\_MR

**Address:** 0x400E1808

**Access:** Read-write

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

- **URSTEN: User Reset Enable**

0 = The detection of a low level on the pin NRST does not generate a User Reset.

1 = The detection of a low level on the pin NRST triggers a User Reset.

- **URSTIEN: User Reset Interrupt Enable**

0 = USRTS bit in RSTC\_SR at 1 has no effect on rstc\_irq.

1 = USRTS bit in RSTC\_SR at 1 asserts rstc\_irq if URSTEN = 0.

- **ERSTL: External Reset Length**

This field defines the external reset length. The external reset is asserted during a time of  $2^{(ERSTL+1)}$  Slow Clock cycles. This allows assertion duration to be programmed between 60 µs and 2 seconds.

- **KEY: Write Access Password**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.



## 16. Real-time Timer (RTT)

### 16.1 Description

The Real-time Timer is built around a 32-bit counter used to count roll-over events of the programmable 16-bit prescaler which enables counting elapsed seconds from a 32 kHz slow clock source. It generates a periodic interrupt and/or triggers an alarm on a programmed value.

It can be configured to be driven by the 1 Hz signal generated by the RTC, thus taking advantage of a calibrated 1 Hz clock.

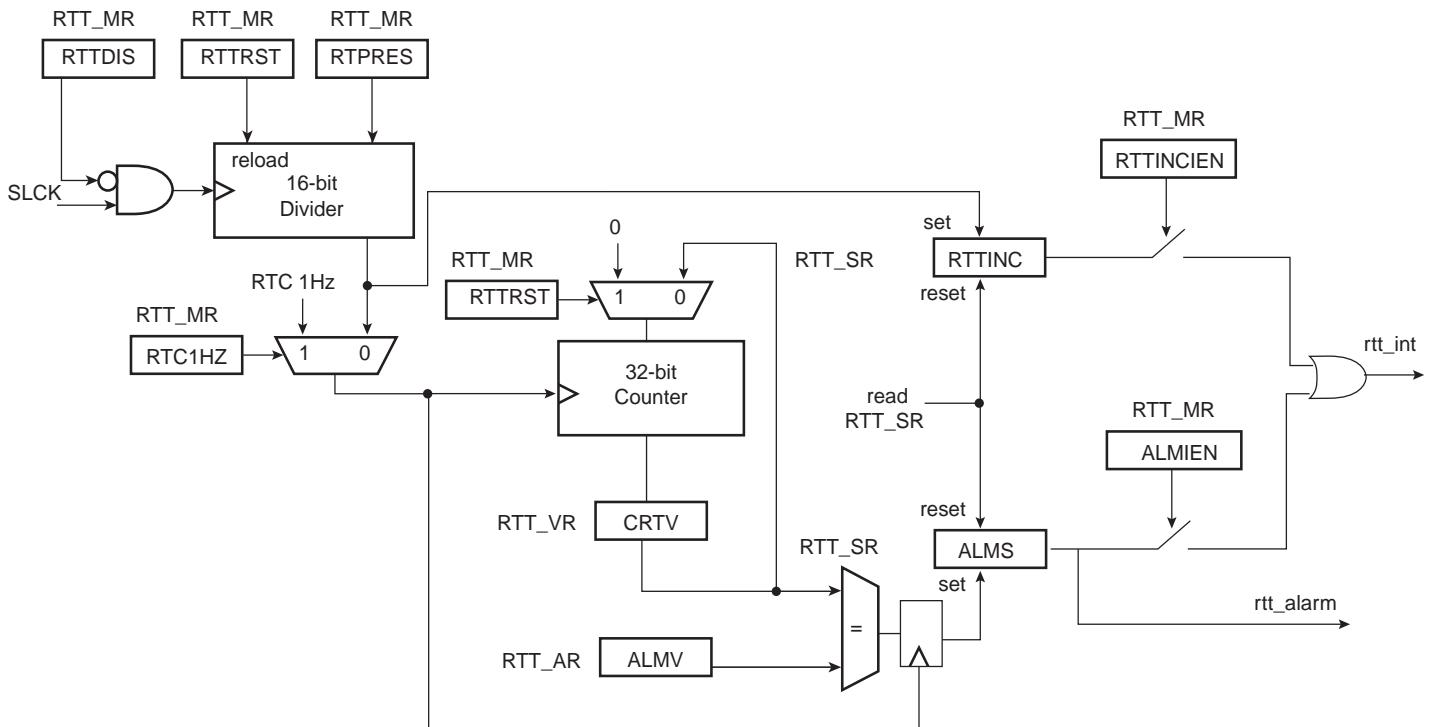
The slow clock source can be fully disabled to reduce power consumption when RTT is not required.

### 16.2 Embedded Characteristics

- 32-bit Free-running Counter on prescaled slow clock or RTC calibrated 1 Hz clock
- 16-bit Configurable Prescaler
- Interrupt on Alarm

### 16.3 Block Diagram

Figure 16-1. Real-time Timer



## 16.4 Functional Description

The Real-time Timer can be used to count elapsed seconds. It is built around a 32-bit counter fed by Slow Clock divided by a programmable 16-bit value. The value can be programmed in the field RTPRES of the Real-time Mode Register (RTT\_MR).

Programming RTPRES at 0x00008000 corresponds to feeding the real-time counter with a 1 Hz signal (if the Slow Clock is 32.768 kHz). The 32-bit counter can count up to  $2^{32}$  seconds, corresponding to more than 136 years, then roll over to 0.

The real-time 32-bit counter can also be supplied by the RTC 1 Hz clock. This mode is interesting when the RTC 1Hz is calibrated (CORRECTION field of RTC\_MR register differs from 0) in order to guaranty the synchronism between RTC and RTT counters.

Setting the RTC 1HZ clock to 1 in RTT\_MR register allows to drive the 32-bit RTT counter with the RTC 1Hz clock. In this mode, RTPRES field has no effect on 32-bit counter but RTTINC is still triggered by RTPRES.

The Real-time Timer can also be used as a free-running timer with a lower time-base. The best accuracy is achieved by writing RTPRES to 3. Programming RTPRES to 1 or 2 is possible, but may result in losing status events because the status register is cleared two Slow Clock cycles after read. Thus if the RTT is configured to trigger an interrupt, the interrupt occurs during 2 Slow Clock cycles after reading RTT\_SR. To prevent several executions of the interrupt handler, the interrupt must be disabled in the interrupt handler and re-enabled when the status register is clear.

The Real-time Timer value (CRTV) can be read at any time in the register RTT\_VR (Real-time Value Register). As this value can be updated asynchronously from the Master Clock, it is advisable to read this register twice at the same value to improve accuracy of the returned value.

The current value of the counter is compared with the value written in the alarm register RTT\_AR (Real-time Alarm Register). If the counter value matches the alarm, the bit ALMS in RTT\_SR is set. The alarm register is set to its maximum value, corresponding to 0xFFFF\_FFFF, after a reset.

The alarm interrupt must be disabled (ALMIEN must be cleared in RTT\_MR register) when writing a new ALMV value in Real-time Alarm Register.

The bit RTTINC in RTT\_SR is set each time there is a prescaler roll-over, so each time the Real-time Timer counter is incremented if RTC1HZ=0 else if RTC1HZ=1 the RTTINC bit can be triggered according to RTPRES value, in a fully independent way from the 32-bit counter increment. This bit can be used to start a periodic interrupt, the period being one second when the RTPRES is programmed with 0x8000 and Slow Clock equal to 32.768 Hz.

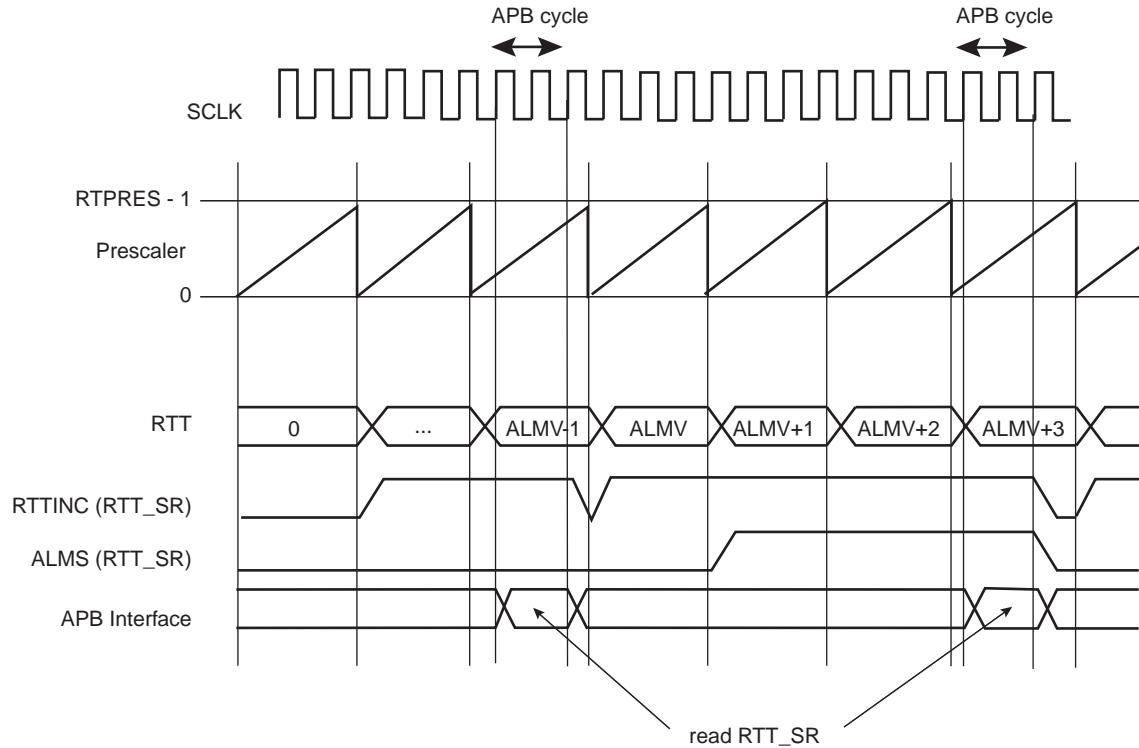
The RTTINCIEN field must be cleared prior to write a new RTPRES value in RTT\_MR register.

Reading the RTT\_SR status register resets the RTTINC and ALMS fields.

Writing the bit RTTRST in RTT\_MR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

When not used, the Real-time Timer can be disabled in order to suppress dynamic power consumption in this module. This can be achieved by setting the RTTDIS field to 1 in RTT\_MR register.

**Figure 16-2. RTT Counting**



## 16.5 Real-time Timer (RTT) User Interface

Table 16-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Mode Register	RTT_MR	Read-write	0x0000_8000
0x04	Alarm Register	RTT_AR	Read-write	0xFFFF_FFFF
0x08	Value Register	RTT_VR	Read-only	0x0000_0000
0x0C	Status Register	RTT_SR	Read-only	0x0000_0000

### 16.5.1 Real-time Timer Mode Register

**Name:** RTT\_MR

**Address:** 0x400E1830

**Access:** Read-write

31	30	29	28	27	26	25	24	
—	—	—	—	—	—	—	—	RTC1HZ
23	22	21	20	19	18	17	16	
—	—	—	RTTDIS	—	RTTRST	RTTINCIEN	ALMIEN	
15	14	13	12	11	10	9	8	
RTPRES								
7	6	5	4	3	2	1	0	
RTPRES								

- **RTPRES: Real-time Timer Prescaler Value**

Defines the number of SCLK periods required to increment the Real-time timer. RTPRES is defined as follows:

RTPRES = 0: The prescaler period is equal to  $2^{16} * \text{SCLK}$  period.

RTPRES ≠ 0: The prescaler period is equal to RTPRES \* SCLK period.

Note: The RTTINCIEN field must be cleared prior to write a new RTPRES value.

- **ALMIEN: Alarm Interrupt Enable**

0 = The bit ALMS in RTT\_SR has no effect on interrupt.

1 = The bit ALMS in RTT\_SR asserts interrupt.

- **RTTINCIEN: Real-time Timer Increment Interrupt Enable**

0 = The bit RTTINC in RTT\_SR has no effect on interrupt.

1 = The bit RTTINC in RTT\_SR asserts interrupt.

- **RTTRST: Real-time Timer Restart**

0 = No effect.

1 = Reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

- **RTTDIS: Real-time Timer Disable**

0 = The real-time timer is enabled.

1 = The real-time timer is disabled (no dynamic power consumption).

Note: RTTDIS is write only.

- **RTC1HZ: Real-Time Clock 1Hz Clock Selection**

0 = The RTT 32-bit counter is driven by the 16-bit prescaler roll-over events.

1 = The RTT 32-bit counter is driven by the RTC 1 Hz clock.

Note: RTC1HZ is write only.

### 16.5.2 Real-time Timer Alarm Register

**Name:** RTT\_AR  
**Address:** 0x400E1834  
**Access:** Read-write

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

- **ALMV: Alarm Value**

Defines the alarm value (ALMV+1) compared with the Real-time Timer.

**Note:** The alarm interrupt must be disabled (ALMIEN must be cleared in RTT\_MR register) when writing a new ALMV value.

### 16.5.3 Real-time Timer Value Register

**Name:** RTT\_VR  
**Address:** 0x400E1838  
**Access:** Read-only

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

- **CRTV: Current Real-time Value**

Returns the current value of the Real-time Timer.

#### 16.5.4 Real-time Timer Status Register

**Name:** RTT\_SR

**Address:** 0x400E183C

**Access:** Read-only

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
-	-	-	-	-	-	RTTINC	ALMS

- **ALMS: Real-time Alarm Status**

0 = The Real-time Alarm has not occurred since the last read of RTT\_SR.

1 = The Real-time Alarm occurred since the last read of RTT\_SR.

- **RTTINC: Real-time Timer Increment**

0 = The Real-time Timer has not been incremented since the last read of the RTT\_SR.

1 = The Real-time Timer has been incremented since the last read of the RTT\_SR.



## 17. Reinforced Safety Watchdog Timer (RSWDT)

### 17.1 Description

When two watchdog timers are implemented in a device, the second one, the Reinforced Safety Watchdog Timer (RSWDT) works in parallel with the Watchdog Timer (WDT) to reinforce safe watchdog operations.

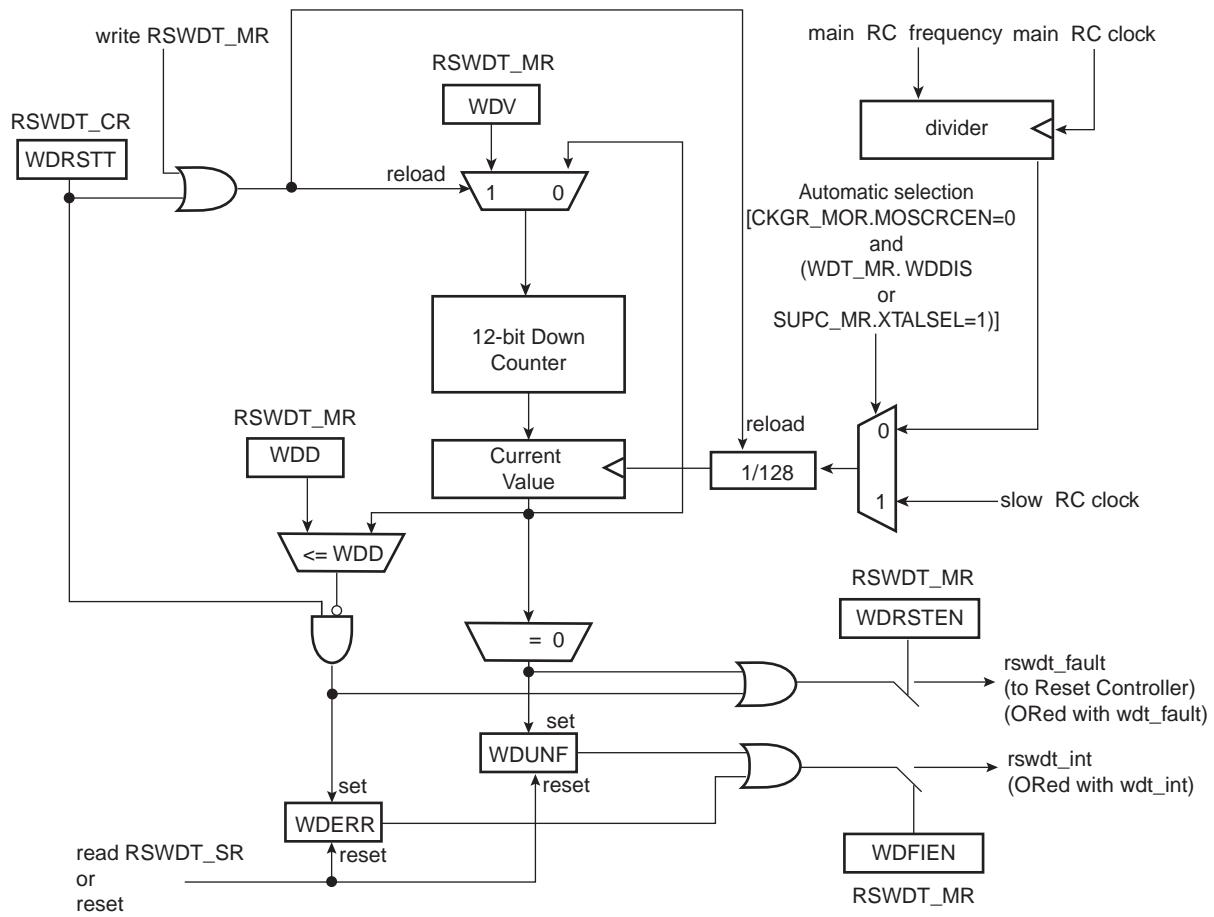
The Reinforced Safety Watchdog Timer (RSWDT) can be used to reinforce the safety level provided by the Watchdog Timer (WDT) in order to prevent system lock-up if the software becomes trapped in a deadlock. The RSWDT works in a fully operable mode, independent of the Watchdog Timer. Its clock source is automatically selected from either the slow RC oscillator clock or main RC oscillator divided clock to get an equivalent slow RC oscillator clock. If the Watchdog Timer clock source (for example the 32 kHz crystal oscillator) fails, the system lock-up is no longer monitored by the Watchdog Timer as the second watchdog timer, RSWDT, will perform the monitoring. Thus, there is no lack of safety irrespective of the external operating conditions. This Reinforced Safety Watchdog Timer shares the same features as the Watchdog Timer (i.e. a 12-bit down counter that allows a watchdog period of up to 16 seconds with slow clock at 32.768 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in debug mode or idle mode.

### 17.2 Embedded Characteristics

- System safety level reinforced by means of an independent second watchdog timer
- Automatically selected reliable independent clock source other than that of first watchdog timer
- 12-bit Key-protected programmable counter
- Provides reset or interrupt signals to the system
- Counter may be stopped while the processor is in debug state or in idle mode

## 17.3 Block Diagram

Figure 17-1. Reinforced Safety Watchdog Timer Block Diagram



## 17.4 Functional Description

The Reinforced Safety Watchdog Timer (RSWDT) can be used to prevent system lock-up if the software becomes trapped in a deadlock. It is supplied with VDDCORE. RSWDT is initialized with default values on processor reset, or power-on sequence and is disabled (it's default mode) under such conditions.

The Reinforced Safety Watchdog Timer works in a fully independent mode distinct from the Watchdog Timer (WDT). Its clock source is automatically selected from either slow RC oscillator clock or main RC oscillator divided clock to get an equivalent slow RC oscillator clock. If the Watchdog Timer (WDT), clock source (for example the 32 kHz crystal oscillator) fails, the system lock-up is no longer monitored by the WDT, but the second watchdog timer, the RSWDT will perform the monitoring. Therefore, continuous safety is assured regardless of the external operating conditions.

The selection of the Reinforced Safety Watchdog Timer clock source consists of a combination of the state of the main RC oscillator (field MOSCRCEN in CKGR\_MOR register), Watchdog Timer (field WDDIS of WDT\_MR register) and slow clock selection (field XTALSEL in the SUPC\_MR register). The Reinforced Safety Watchdog Timer is driven by the slow RC oscillator if the main RC oscillator is not already in use, and either the selected slow clock is the 32 kHz crystal oscillator, or the Watchdog Timer (WDT) is disabled. Accordingly, slow or main RC oscillators are automatically enabled.

The RSWDT is built around a 12-bit down counter, which is loaded with a slow clock value other than that of the slow clock in the Watchdog Timer, defined in the WDV field of the Mode Register (RSWDT\_MR). The Reinforced Safety Watchdog Timer uses the Slow Clock divided by 128 to establish the maximum watchdog period to be 16 seconds (with a typical Slow Clock of 32.768 kHz).

After a processor reset, the value of WDV is 0xFFFF, corresponding to the maximum value of the counter with the external reset generation enabled (WDRSTEN field at 1 after a backup reset). This means that a default watchdog is running at reset, i.e., at power-up.

The Mode Register (RSWDT\_MR) can be written only once. Only a processor reset resets it. Writing the RSWDT\_MR register reloads the timer with the newly programmed mode parameters.

In normal operation, the user reloads the watchdog at regular intervals before the timer underflow occurs, by writing the Control Register (RSWDT\_CR) with the bit WDRSTT to 1. The watchdog counter is then immediately reloaded from RSWDT\_MR and restarted, and the Slow Clock 128 divider is reset and restarted. The RSWDT\_CR register is write-protected. As a result, writing RSWDT\_CR without the correct hard-coded key has no effect. If an underflow does occur, the "wdt\_fault" signal to the reset controller is asserted if the bit WDRSTEN is set in the Mode Register (RSWDT\_MR). Moreover, the bit WDUNF is set in the Status Register (RSWDT\_SR).

To prevent a software deadlock that continuously triggers the RSWDT, the reload of the RSWDT must occur while the watchdog counter is within a window between 0 and WDD, WDD is defined in the Mode Register, RSWDT\_MR.

Any attempt to restart the watchdog while the watchdog counter is between WDV and WDD results in a watchdog error, even if the RSWDT is disabled. The WDERR bit is updated in the RSWDT\_SR and the "wdt\_fault" signal to the reset controller is asserted.

Note that this feature can be disabled by programming a WDD value greater than or equal to the WDV value. In such a configuration, restarting the Reinforced Safety Watchdog Timer is permitted in the whole range [0; WDV] and does not generate an error. This is the default configuration on reset (the WDD and WDV values are equal).

The status bits, WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the WDFIEN bit is set in the mode register. The signal "wdt\_fault" to the reset controller causes a Watchdog reset if the WDRSTEN bit is set as explained in the reset controller programmer's documentation. In that case, the processor and the watchdog timer are reset, and the WDERR and WDUNF flags are reset.

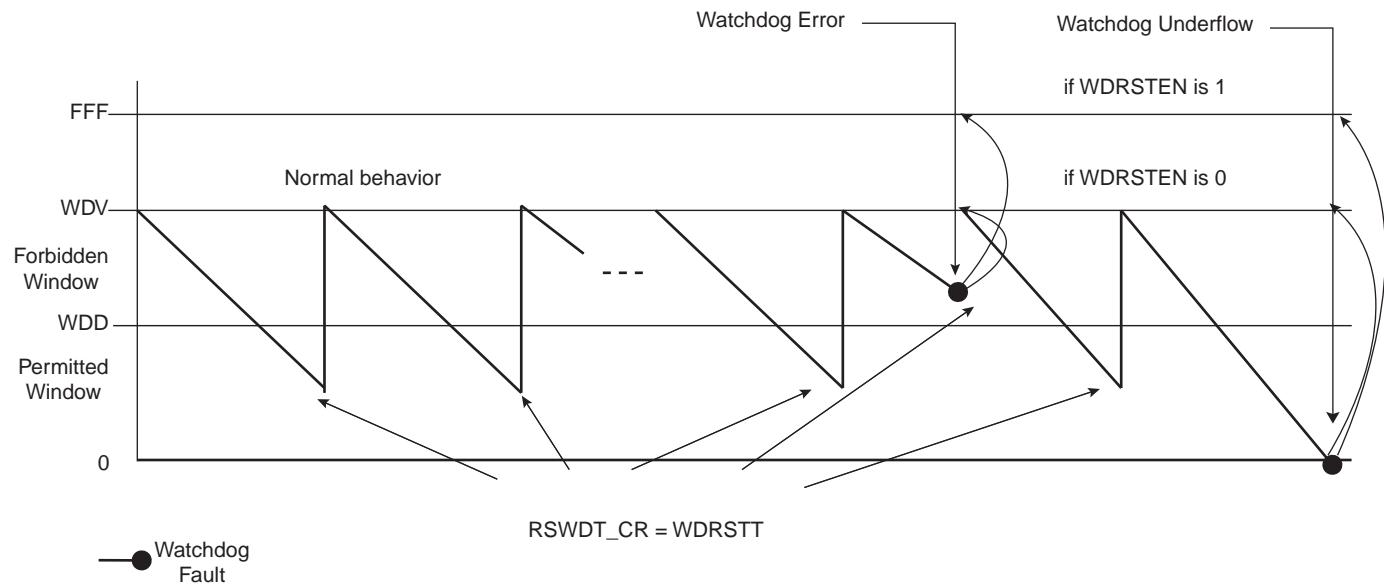
If a reset is generated, or if RSWDT\_SR is read, the status bits are reset, the interrupt is cleared, and the "wdt\_fault" signal to the reset controller is deasserted.

Writing the RSWDT\_MR reloads and restarts the down counter.

The RSWDT is disabled after any power-on sequence.

While the processor is in debug state or in idle mode, the counter may be stopped depending on the value programmed for the WDIDLEHLT and Wddbghlt bits in the RSWDT\_MR.

**Figure 17-2. Watchdog Behavior**



## 17.5 Reinforced Safety Watchdog Timer (RSWDT) User Interface

Table 17-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RSWDT_CR	Write-only	-
0x04	Mode Register	RSWDT_MR	Read-write Once	0x3FFF_AFFF
0x08	Status Register	RSWDT_SR	Read-only	0x0000_0000

### 17.5.1 Reinforced Safety Watchdog Timer Control Register

**Name:** RSWDT\_CR

**Address:** 0x400E1900

**Access:** Write-only

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

- **WDRSTT: Watchdog Restart**

0: No effect.

1: Restarts the watchdog.

- **KEY: Password**

Should be written at value 0xC4. Writing any other value in this field aborts the write operation.

## 17.5.2 Reinforced Safety Watchdog Timer Mode Register

**Name:** RSWDT\_MR

**Address:** 0x400E1904

**Access:** Read-write Once

31	30	29	28	27	26	25	24
		WDIDLEHLT	WDBGHLT			WDD	
23	22	21	20	19	18	17	16
				WDD			
15	14	13	12	11	10	9	8
WDDIS	WDRPROC	WDRSTEN	WDFIEN			WDV	
7	6	5	4	3	2	1	0
				WDV			

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 12-bit watchdog counter.

- **WDFIEN: Watchdog Fault Interrupt Enable**

0: A Watchdog fault (underflow or error) has no effect on interrupt.

1: A Watchdog fault (underflow or error) asserts interrupt.

- **WDRSTEN: Watchdog Reset Enable**

0: A Watchdog fault (underflow or error) has no effect on the resets.

1: A Watchdog fault (underflow or error) triggers a watchdog reset.

- **WDRPROC: Watchdog Reset Processor**

0: If WDRSTEN is 1, a watchdog fault (underflow or error) activates all resets.

1: If WDRSTEN is 1, a watchdog fault (underflow or error) activates the processor reset.

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the watchdog timer.

If the watchdog timer value is less than or equal to WDD, writing RSWDT\_CR with WDRSTT = 1 restarts the timer.

If the watchdog timer value is greater than WDD, writing RSWDT\_CR with WDRSTT = 1 causes a Watchdog error.

- **WDBGHLT: Watchdog Debug Halt**

0: The watchdog runs when the processor is in debug state.

1: The watchdog stops when the processor is in debug state.

- **WDIDLEHLT: Watchdog Idle Halt**

0: The watchdog runs when the system is in idle mode.

1: The watchdog stops when the system is in idle state.

- **WDDIS: Watchdog Disable**

0: Enables the watchdog timer.

1: Disables the watchdog timer.

### 17.5.3 Reinforced Safety Watchdog Timer Status Register

**Name:** RSWDT\_SR

**Address:** 0x400E1908

**Access:** Read-only

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
-	-	-	-	-	-	WDERR	WDUNF

- **WDUNF: Watchdog Underflow**

0: No watchdog underflow occurred since the last read of RSWDT\_SR.

1: At least one watchdog underflow occurred since the last read of RSWDT\_SR.

- **WDERR: Watchdog Error**

0: No watchdog error occurred since the last read of RSWDT\_SR.

1: At least one watchdog error occurred since the last read of RSWDT\_SR.

## 18. Real-time Clock (RTC)

### 18.1 Description

The Real-time Clock (RTC) peripheral is designed for very low power consumption.

It combines a complete time-of-day clock with alarm and a two-hundred-year Gregorian or Persian calendar, complemented by a programmable periodic interrupt. The alarm and calendar registers are accessed by a 32-bit data bus.

The time and calendar values are coded in binary-coded decimal (BCD) format. The time format can be 24-hour mode or 12-hour mode with an AM/PM indicator.

Updating time and calendar fields and configuring the alarm fields are performed by a parallel capture on the 32-bit data bus. An entry control is performed to avoid loading registers with incompatible BCD format data or with an incompatible date according to the current month/year/century.

A clock divider calibration circuitry enables to compensate crystal oscillator frequency inaccuracy.

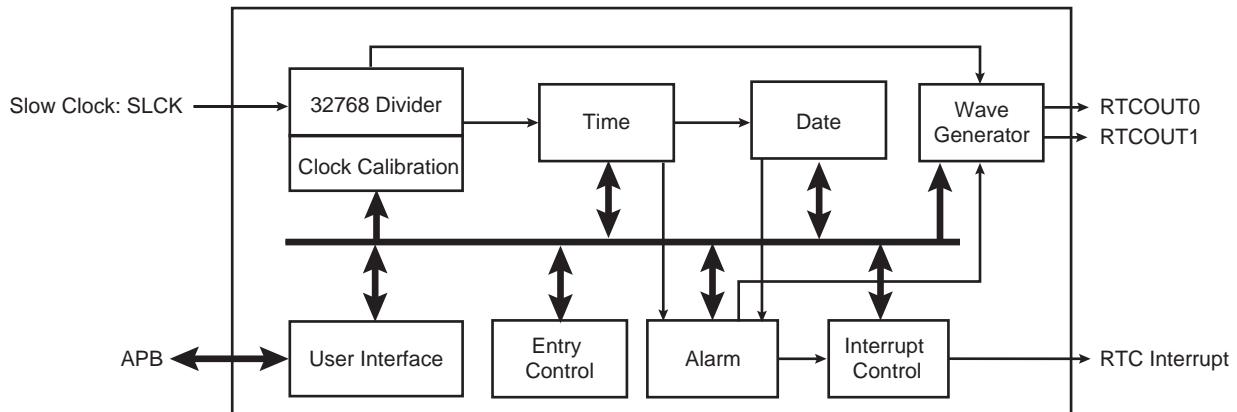
An RTC output can be programmed to generate several waveforms, including a prescaled clock derived from 32.768 kHz

### 18.2 Embedded Characteristics

- Ultra Low Power Consumption
- Full Asynchronous Design
- Gregorian Calendar up to 2099 or Persian Calendar
- Programmable Periodic Interrupt
- Safety/security features:
  - Valid Time and Date Programmation Check
  - On-The-Fly Time and Date Validity Check
- Crystal Oscillator Clock Calibration
- Waveform Generation

## 18.3 Block Diagram

Figure 18-1. RTC Block Diagram



## 18.4 Product Dependencies

### 18.4.1 Power Management

The Real-time Clock is continuously clocked at 32768 Hz. The Power Management Controller has no effect on RTC behavior.

### 18.4.2 Interrupt

RTC interrupt line is connected on one of the internal sources of the interrupt controller. RTC interrupt requires the interrupt controller to be programmed first.

## 18.5 Functional Description

The RTC provides a full binary-coded decimal (BCD) clock that includes century (19/20), year (with leap years), month, date, day, hours, minutes and seconds.

The valid year range is 1900 to 2099 in Gregorian mode, a two-hundred-year calendar(or 1300 to 1499 in Persian mode).

The RTC can operate in 24-hour mode or in 12-hour mode with an AM/PM indicator.

Corrections for leap years are included (all years divisible by 4 being leap years). This is correct up to the year 2099.

The RTC can generate configurable waveforms on RTCOUT0/1 outputs.

### 18.5.1 Reference Clock

The reference clock is Slow Clock (SLCK). It can be driven internally or by an external 32.768 kHz crystal.

During low power modes of the processor, the oscillator runs and power consumption is critical. The crystal selection has to take into account the current consumption for power saving and the frequency drift due to temperature effect on the circuit for time accuracy.

### 18.5.2 Timing

The RTC is updated in real time at one-second intervals in normal mode for the counters of seconds, at one-minute intervals for the counter of minutes and so on.

Due to the asynchronous operation of the RTC with respect to the rest of the chip, to be certain that the value read in the RTC registers (century, year, month, date, day, hours, minutes, seconds) are valid and stable, it is necessary to read

these registers twice. If the data is the same both times, then it is valid. Therefore, a minimum of two and a maximum of three accesses are required.

### 18.5.3 Alarm

The RTC has five programmable fields: month, date, hours, minutes and seconds.

Each of these fields can be enabled or disabled to match the alarm condition:

- If all the fields are enabled, an alarm flag is generated (the corresponding flag is asserted and an interrupt generated if enabled) at a given month, date, hour/minute/second.
- If only the "seconds" field is enabled, then an alarm is generated every minute.

Depending on the combination of fields enabled, a large number of possibilities are available to the user ranging from minutes to 365/366 days.

Hour, minute and second matching alarm (SECEN, MINEN, HOUREN) can be enabled independently of SEC, MIN, HOUR fields.

Note: To change one of the SEC, MIN, HOUR, DATE, MONTH fields, it is recommended to disable the field before changing the value and then re-enable it after the change has been made. This requires up to 3 accesses to the RTC\_TIMALR or RTC\_CALALR registers. The first access clears the enable corresponding to the field to change (SECEN,MINEN,HOUREN,DATEEN,MTHEN). If the field is already cleared, this access is not required. The second access performs the change of the value (SEC, MIN, HOUR, DATE, MONTH). The third access is required to re-enable the field by writing 1 in SECEN,MINEN,HOUREN,DATEEN,MTHEN fields.

### 18.5.4 Error Checking when Programming

Verification on user interface data is performed when accessing the century, year, month, date, day, hours, minutes, seconds and alarms. A check is performed on illegal BCD entries such as illegal date of the month with regard to the year and century configured.

If one of the time fields is not correct, the data is not loaded into the register/counter and a flag is set in the validity register. The user can not reset this flag. It is reset as soon as an acceptable value is programmed. This avoids any further side effects in the hardware. The same procedure is done for the alarm.

The following checks are performed:

1. Century (check if it is in range 19 - 20 or 13-14 in Persian mode)
2. Year (BCD entry check)
3. Date (check range 01 - 31)
4. Month (check if it is in BCD range 01 - 12, check validity regarding "date")
5. Day (check range 1 - 7)
6. Hour (BCD checks: in 24-hour mode, check range 00 - 23 and check that AM/PM flag is not set if RTC is set in 24-hour mode; in 12-hour mode check range 01 - 12)
7. Minute (check BCD and range 00 - 59)
8. Second (check BCD and range 00 - 59)

Note: If the 12-hour mode is selected by means of the RTC\_MR register, a 12-hour value can be programmed and the returned value on RTC\_TIMR will be the corresponding 24-hour value. The entry control checks the value of the AM/PM indicator (bit 22 of RTC\_TIMR register) to determine the range to be checked.

### 18.5.5 RTC Internal Free Running Counter Error Checking

To improve the reliability and security of the RTC, a permanent check is performed on the internal free running counters to report non-BCD or invalid date/time values.

An error is reported by TDERR bit in the status register (RTC\_SR) if an incorrect value has been detected. The flag can be cleared by programming the TDERRCLR in the RTC status clear control register (RTC\_SCCR).

Anyway the TDERR error flag will be set again if the source of the error has not been cleared before clearing the TDERR flag. The clearing of the source of such error can be done either by reprogramming a correct value on RTC\_CALR and/or RTC\_TIMR registers.

The RTC internal free running counters may automatically clear the source of TDERR due to their roll-over (i.e. every 10 seconds for SECONDS[3:0] bitfield in RTC\_TIMR register). In this case the TDERR is held high until a clear command is asserted by TDERRCLR bit in RTC\_SCCR register.

#### 18.5.6 Updating Time/Calendar

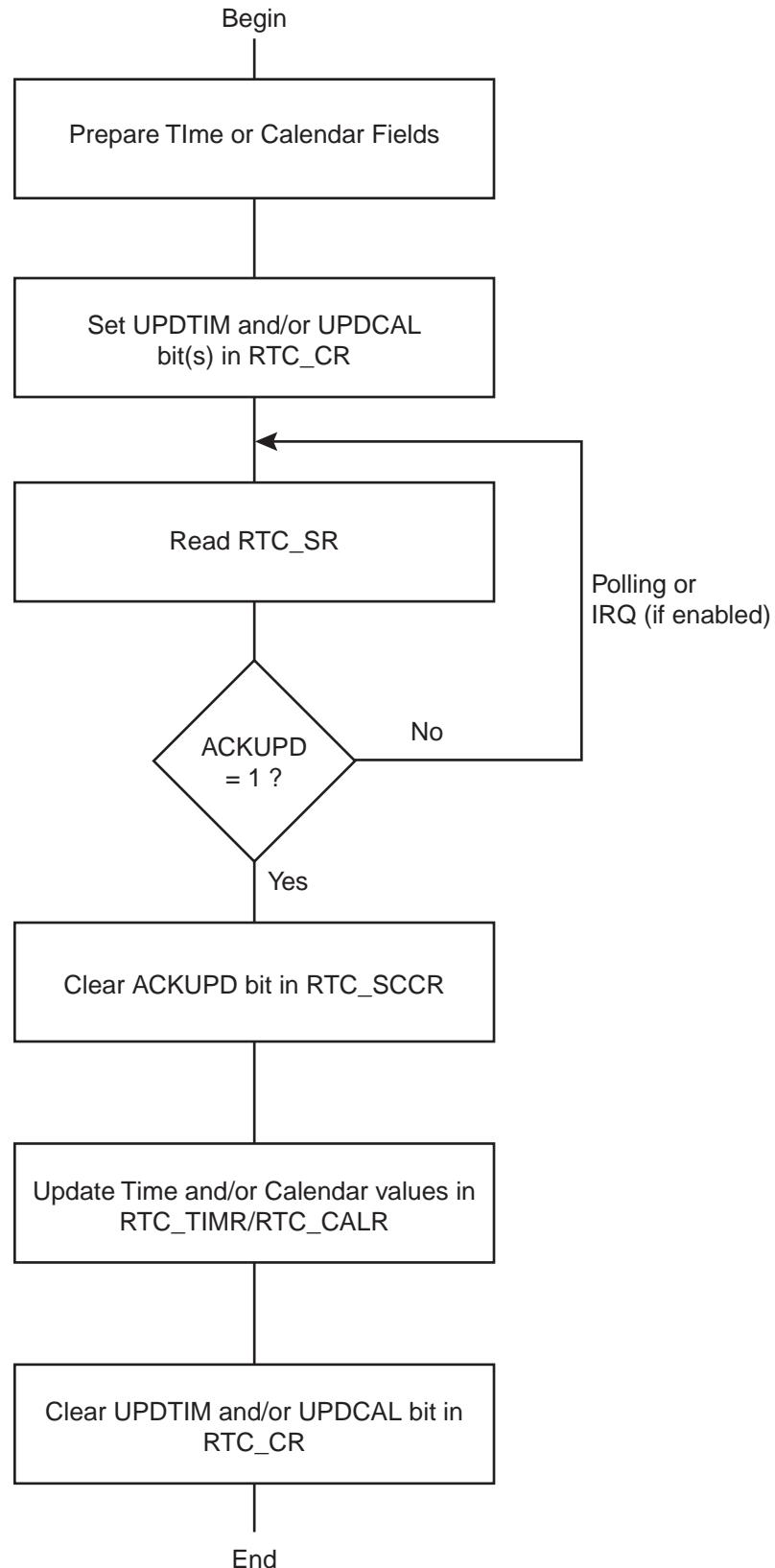
To update any of the time/calendar fields, the user must first stop the RTC by setting the corresponding field in the Control Register. Bit UPDTIM must be set to update time fields (hour, minute, second) and bit UPDCAL must be set to update calendar fields (century, year, month, date, day).

Then the user must poll or wait for the interrupt (if enabled) of bit ACKUPD in the Status Register. Once the bit reads 1, it is mandatory to clear this flag by writing the corresponding bit in RTC\_SCCR. The user can now write to the appropriate Time and Calendar register.

Once the update is finished, the user must reset (0) UPDTIM and/or UPDCAL in the Control

When entering programming mode of the calendar fields, the time fields remain enabled. When entering the programming mode of the time fields, both time and calendar fields are stopped. This is due to the location of the calendar logic circuitry (downstream for low-power considerations). It is highly recommended to prepare all the fields to be updated before entering programming mode. In successive update operations, the user must wait at least one second after resetting the UPDTIM/UPDCAL bit in the RTC\_CR (Control Register) before setting these bits again. This is done by waiting for the SEC flag in the Status Register before setting UPDTIM/UPDCAL bit. After resetting UPDTIM/UPDCAL, the SEC flag must also be cleared.

**Figure 18-2. Update Sequence**



### 18.5.7 RTC Accurate Clock Calibration

The crystal oscillator that drives the RTC may not be as accurate as expected mainly due to temperature variation. The RTC is equipped with circuitry able to correct slow clock crystal drift.

To compensate for possible temperature variations over time, this accurate clock calibration circuitry can be programmed on-the-fly and also programmed during application manufacturing, in order to correct the crystal frequency accuracy at room temperature (20-25°C). The typical clock drift range at room temperature is  $\pm 20$  ppm.

In the device operating temperature range, the 32.768 kHz crystal oscillator clock inaccuracy can be up to -200 ppm.

The RTC clock calibration circuitry allows positive or negative correction in a range of 1.5 ppm to 1950 ppm. After correction, the remaining crystal drift is as follows:

- Below 1 ppm, for an initial crystal drift between 1.5 ppm up to 90 ppm
- Below 2 ppm, for an initial crystal drift between 90 ppm up to 130 ppm
- Below 5 ppm, for an initial crystal drift between 130 ppm up to 200 ppm

The calibration circuitry acts by slightly modifying the 1 Hz clock period from time to time. When the period is modified, depending on the sign of the correction, the 1 Hz clock period increases or reduces by around 4 ms. The period interval between 2 correction events is programmable in order to cover the possible crystal oscillator clock variations.

The inaccuracy of a crystal oscillator at typical room temperature ( $\pm 20$  ppm at 20-25 degrees Celsius) can be compensated if a reference clock/signal is used to measure such inaccuracy. This kind of calibration operation can be set up during the final product manufacturing by means of measurement equipment embedding such a reference clock. The correction of value must be programmed into the RTC Mode Register (RTC\_MR), and this value is kept as long as the circuitry is powered (backup area). Removing the backup power supply cancels this calibration. This room temperature calibration can be further processed by means of the networking capability of the target application.

To ease the comparison of the inherent crystal accuracy with the reference clock/signal during manufacturing, an internal prescaled 32.768 kHz clock derivative signal can be assigned to drive RTC output. To accommodate the measure, several clock frequencies can be selected among 1 Hz, 32 Hz, 64 Hz, 512 Hz.

In any event, this adjustment does not take into account the temperature variation.

The frequency drift (up to -200 ppm) due to temperature variation can be compensated using a reference time if the application can access such a reference. If a reference time cannot be used, a temperature sensor can be placed close to the crystal oscillator in order to get the operating temperature of the crystal oscillator. Once obtained, the temperature may be converted using a lookup table (describing the accuracy/temperature curve of the crystal oscillator used) and RTC\_MR configured accordingly. The calibration can be performed on-the-fly. This adjustment method is not based on a measurement of the crystal frequency/drift and therefore can be improved by means of the networking capability of the target application.

If no crystal frequency adjustment has been done during manufacturing, it is still possible to do it. In the case where a reference time of the day can be obtained through LAN/WAN network, it is possible to calculate the drift of the application crystal oscillator by comparing the values read on RTC Time Register (RTC\_TIMR) and programming the HIGHPPM and CORRECTION bitfields on RTC\_MR according to the difference measured between the reference time and those of RTC\_TIMR.

### 18.5.8 Waveform Generation

Waveforms can be generated by the RTC in order to take advantage of the RTC inherent prescalers while the RTC is the only powered circuitry (low power mode of operation, backup mode) or in any active modes. Going into backup or low power operating modes does not affect the waveform generation outputs.

The RTC outputs (RTCOUT0 and RTCOUT1) have a source driver selected among 7 possibilities.

The first selection choice sticks the associated output at 0 (This is the reset value and it can be used at any time to disable the waveform generation).

Selection choices 1 to 4 respectively select 1 Hz, 32 Hz, 64 Hz and 512 Hz.

32 Hz or 64 Hz can drive, for example, a TN LCD backplane signal while 1 Hz can be used to drive a blinking character like ":" for basic time display (hour, minute) on TN LCDs.

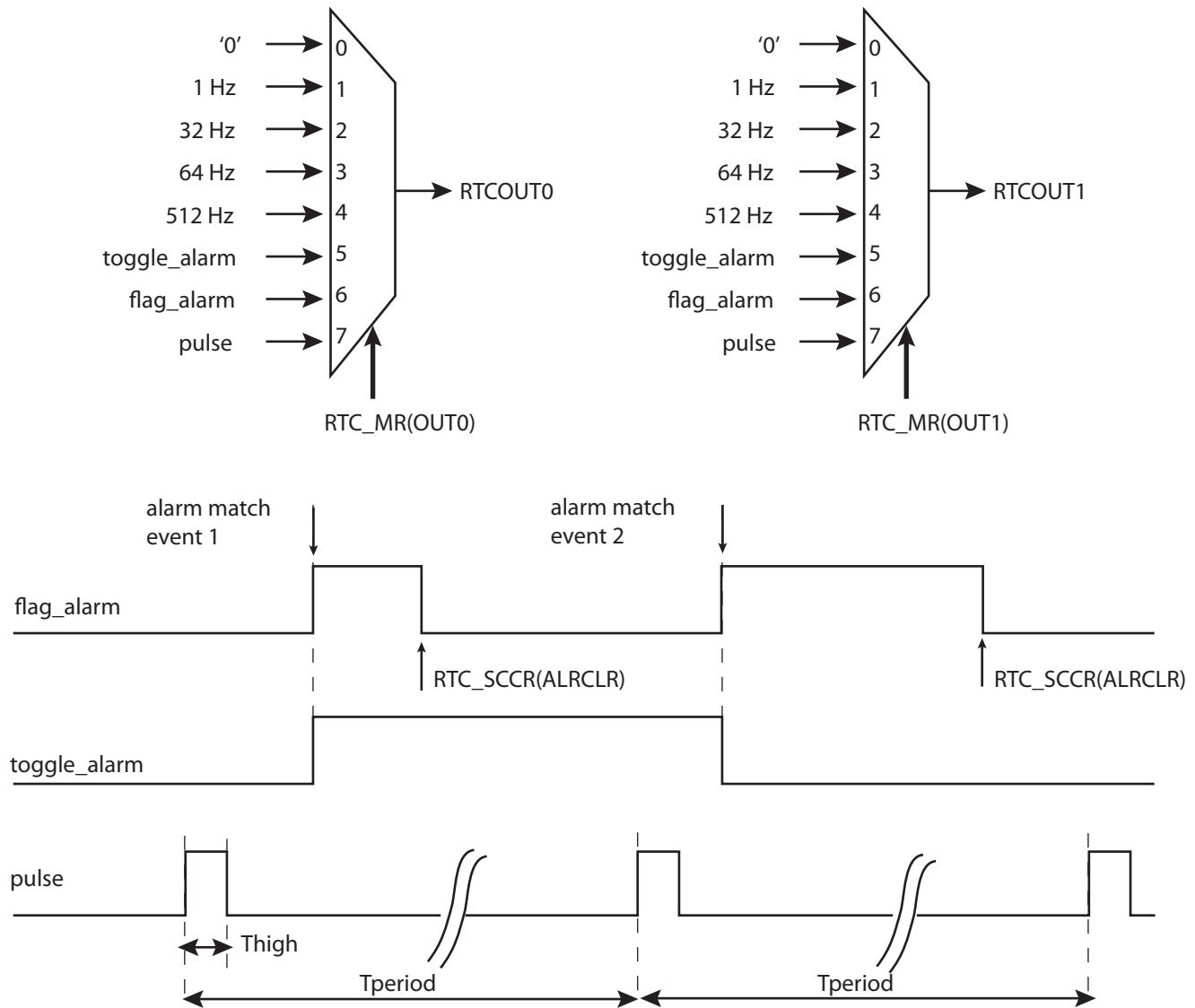
Selection choice 5 provides a toggling signal when the RTC alarm is reached.

Selection choice 6 provides a copy of the alarm flag, so the associated output is set high (logical 1) when an alarm occurs and immediately cleared when software clears the alarm interrupt source.

Selection choice 7 provides a 1 Hz periodic high pulse of 15  $\mu$ s duration that can be used to drive external devices for power consumption reduction or any other purpose.

PIO lines associated to RTC outputs are automatically selecting these waveforms as soon as RTC\_MR register corresponding fields OUT0 and OUT1 differ from 0.

**Figure 18-3. Waveform Generation**



## 18.6 Real-time Clock (RTC) User Interface

Table 18-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RTC_CR	Read-write	0x0
0x04	Mode Register	RTC_MR	Read-write	0x0
0x08	Time Register	RTC_TIMR	Read-write	0x0
0x0C	Calendar Register	RTC_CALR	Read-write	0x01a11020
0x10	Time Alarm Register	RTC_TIMALR	Read-write	0x0
0x14	Calendar Alarm Register	RTC_CALALR	Read-write	0x01010000
0x18	Status Register	RTC_SR	Read-only	0x0
0x1C	Status Clear Command Register	RTC_SCCR	Write-only	—
0x20	Interrupt Enable Register	RTC_IER	Write-only	—
0x24	Interrupt Disable Register	RTC_IDR	Write-only	—
0x28	Interrupt Mask Register	RTC_IMR	Read-only	0x0
0x2C	Valid Entry Register	RTC_VER	Read-only	0x0
0x30–0xC4	Reserved Register	—	—	—
0xC8–0xF8	Reserved Register	—	—	—
0xFC	Reserved Register	—	—	—

Note: If an offset is not listed in the table it must be considered as reserved.

### 18.6.1 RTC Control Register

**Name:** RTC\_CR

**Address:** 0x400E1860

**Access:** Read-write

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

- **UPDTIM: Update Request Time Register**

0 = No effect.

1 = Stops the RTC time counting.

Time counting consists of second, minute and hour counters. Time counters can be programmed once this bit is set and acknowledged by the bit ACKUPD of the Status Register.

- **UPDCAL: Update Request Calendar Register**

0 = No effect.

1 = Stops the RTC calendar counting.

Calendar counting consists of day, date, month, year and century counters. Calendar counters can be programmed once this bit is set.

- **TIMEVSEL: Time Event Selection**

The event that generates the flag TIMEV in RTC\_SR (Status Register) depends on the value of TIMEVSEL.

Value	Name	Description
0	MINUTE	Minute change
1	HOUR	Hour change
2	MIDNIGHT	Every day at midnight
3	NOON	Every day at noon

- **CALEVSEL: Calendar Event Selection**

The event that generates the flag CALEV in RTC\_SR depends on the value of CALEVSEL

Value	Name	Description
0	WEEK	Week change (every Monday at time 00:00:00)
1	MONTH	Month change (every 01 of each month at time 00:00:00)
2	YEAR	Year change (every January 1 at time 00:00:00)

## 18.6.2 RTC Mode Register

**Name:** RTC\_MR

**Address:** 0x400E1864

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	TPERIOD	–	–	THIGH	–	–
23	22	21	20	19	18	17	16
–	OUT1	–	–	–	OUT0	–	–
15	14	13	12	11	10	9	8
HIGHPPM	–	–	CORRECTION	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	NEGPPM	–	–	PERSIAN	HRMOD

- **HRMOD: 12-/24-hour Mode**

0 = 24-hour mode is selected.

1 = 12-hour mode is selected.

- **PERSIAN: PERSIAN Calendar**

0 = Gregorian Calendar.

1 = Persian Calendar.

- **NEGPPM: NEGative PPM Correction**

0 = positive correction (the divider will be slightly lower than 32768).

1 = negative correction (the divider will be slightly higher than 32768).

Refer to CORRECTION and HIGHPPM field descriptions.

- **CORRECTION: Slow Clock Correction**

0 = No correction

1..127 = The slow clock will be corrected according to the formula given below in HIGHPPM description.

- **HIGHPPM: HIGH PPM Correction**

0 = lower range ppm correction with accurate correction.

1 = higher range ppm correction with accurate correction.

If the absolute value of the correction to be applied is lower than 30ppm, it is recommended to clear HIGHPPM. HIGHPPM set to 1 is recommended for 30 ppm correction and above.

Formula:

If HIGHPPM = 0, then the clock frequency correction range is from 1.5 ppm up to 98 ppm. The RTC accuracy is less than 1 ppm for a range correction from 1.5 ppm up to 30 ppm..

The correction field must be programmed according to the required correction in ppm, the formula is as follows:

$$CORRECTION = \frac{3906}{20 \times ppm} - 1$$

The value obtained must be rounded to the nearest integer prior to being programmed into CORRECTION field.

If HIGHPPM = 1, then the clock frequency correction range is from 30.5 ppm up to 1950 ppm. The RTC accuracy is less than 1 ppm for a range correction from 30.5 ppm up to 90 ppm.

The correction field must be programmed according to the required correction in ppm, the formula is as follows:

$$CORRECTION = \frac{3906}{ppm} - 1$$

The value obtained must be rounded to the nearest integer prior to be programmed into CORRECTION field.

If NEGPPM is set to 1, the ppm correction is negative.

- **OUT0: RTCOUT0 OutputSource Selection**

Value	Name	Description
0	NO_WAVE	no waveform, stuck at '0'
1	FREQ1HZ	1 Hz square wave
2	FREQ32HZ	32 Hz square wave
3	FREQ64HZ	64 Hz square wave
4	FREQ512HZ	512 Hz square wave
5	ALARM_TOGGLE	output toggles when alarm flag rises
6	ALARM_FLAG	output is a copy of the alarm flag
7	PROG_PULSE	duty cycle programmable pulse

- **OUT1: RTCOUT1 Output Source Selection**

Value	Name	Description
0	NO_WAVE	no waveform, stuck at '0'
1	FREQ1HZ	1 Hz square wave
2	FREQ32HZ	32 Hz square wave
3	FREQ64HZ	64 Hz square wave
4	FREQ512HZ	512 Hz square wave
5	ALARM_TOGGLE	output toggles when alarm flag rises
6	ALARM_FLAG	output is a copy of the alarm flag
7	PROG_PULSE	duty cycle programmable pulse

- **THIGH: High Duration of the Output Pulse**

Value	Name	Description
0	H_31MS	31.2 ms
1	H_16MS	15.6 ms
2	H_4MS	3.91 5.7ms
3	H_976US	976 µs
4	H_488US	488 µs

Value	Name	Description
5	H_122US	122 µs
6	H_30US	30.5 µs
7	H_15US	15.2 µs

- **TPERIOD: Period of the Output Pulse**

Value	Name	Description
0	P_1S	1 second
1	P_500MS	500 ms
2	P_250MS	250 ms
3	P_125MS	125 ms

### 18.6.3 RTC Time Register

**Name:** RTC\_TIMR

**Address:** 0x400E1868

**Access:** Read-write

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

- **SEC: Current Second**

The range that can be set is 0 - 59 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **MIN: Current Minute**

The range that can be set is 0 - 59 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **HOUR: Current Hour**

The range that can be set is 1 - 12 (BCD) in 12-hour mode or 0 - 23 (BCD) in 24-hour mode.

- **AMPM: Ante Meridiem Post Meridiem Indicator**

This bit is the AM/PM indicator in 12-hour mode.

0 = AM.

1 = PM.

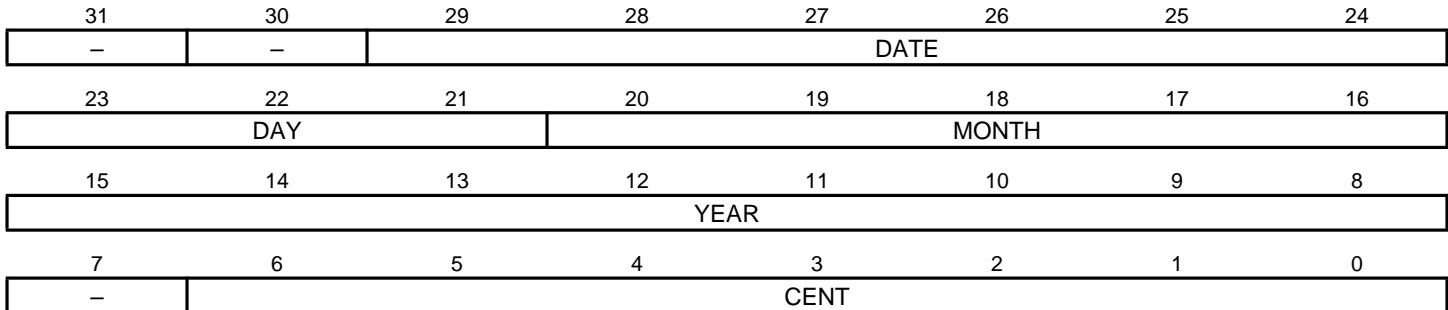
All non-significant bits read zero.

#### 18.6.4 RTC Calendar Register

**Name:** RTC\_CALR

**Address:** 0x400E186C

**Access:** Read-write



- **CENT: Current Century**

The range that can be set is 19 - 20 (gregorian) or 13-14 (persian) (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **YEAR: Current Year**

The range that can be set is 00 - 99 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **MONTH: Current Month**

The range that can be set is 01 - 12 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **DAY: Current Day in Current Week**

The range that can be set is 1 - 7 (BCD).

The coding of the number (which number represents which day) is user-defined as it has no effect on the date counter.

- **DATE: Current Day in Current Month**

The range that can be set is 01 - 31 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

All non-significant bits read zero.

### 18.6.5 RTC Time Alarm Register

**Name:** RTC\_TIMALR

**Address:** 0x400E1870

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
HOUREN	AMPM				HOUR		
15	14	13	12	11	10	9	8
MINEN				MIN			
7	6	5	4	3	2	1	0
SECEN				SEC			

**Note:** To change one of the SEC, MIN, HOUR fields, it is recommended to disable the field before changing the value and then re-enable it after the change has been made. This requires up to 3 accesses to the RTC\_TIMALR register. The first access clears the enable corresponding to the field to change (SECEN,MINEN,HOUREN). If the field is already cleared, this access is not required. The second access performs the change of the value (SEC, MIN, HOUR). The third access is required to re-enable the field by writing 1 in SECEN, MINEN, HOUREN fields.

- **SEC: Second Alarm**

This field is the alarm field corresponding to the BCD-coded second counter.

- **SECEN: Second Alarm Enable**

0 = The second-matching alarm is disabled.

1 = The second-matching alarm is enabled.

- **MIN: Minute Alarm**

This field is the alarm field corresponding to the BCD-coded minute counter.

- **MINEN: Minute Alarm Enable**

0 = The minute-matching alarm is disabled.

1 = The minute-matching alarm is enabled.

- **HOUR: Hour Alarm**

This field is the alarm field corresponding to the BCD-coded hour counter.

- **AMPM: AM/PM Indicator**

This field is the alarm field corresponding to the BCD-coded hour counter.

- **HOUREN: Hour Alarm Enable**

0 = The hour-matching alarm is disabled.

1 = The hour-matching alarm is enabled.

### 18.6.6 RTC Calendar Alarm Register

**Name:** RTC\_CALALR

**Address:** 0x400E1874

**Access:** Read-write

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

**Note:** To change one of the DATE, MONTH fields, it is recommended to disable the field before changing the value and then re-enable it after the change has been made. This requires up to 3 accesses to the RTC\_CALALR register. The first access clears the enable corresponding to the field to change (DATEEN,MTHEN). If the field is already cleared, this access is not required. The second access performs the change of the value (DATE,MONTH). The third access is required to re-enable the field by writing 1 in DATEEN, MTHEN fields.

- **MONTH: Month Alarm**

This field is the alarm field corresponding to the BCD-coded month counter.

- **MTHEN: Month Alarm Enable**

0 = The month-matching alarm is disabled.

1 = The month-matching alarm is enabled.

- **DATE: Date Alarm**

This field is the alarm field corresponding to the BCD-coded date counter.

- **DATEEN: Date Alarm Enable**

0 = The date-matching alarm is disabled.

1 = The date-matching alarm is enabled.

### 18.6.7 RTC Status Register

**Name:** RTC\_SR

**Address:** 0x400E1878

**Access:** Read-only

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
—	—	TDERR	CALEV	TIMEV	SEC	ALARM	ACKUPD

- **ACKUPD: Acknowledge for Update**

0 (FREERUN) = Time and calendar registers cannot be updated.

1 (UPDATE) = Time and calendar registers can be updated.

- **ALARM: Alarm Flag**

0 (NO\_ALARMEVENT) = No alarm matching condition occurred.

1 (ALARMEVENT) = An alarm matching condition has occurred.

- **SEC: Second Event**

0 (NO\_SECEVENT) = No second event has occurred since the last clear.

1 (SECEVENT) = At least one second event has occurred since the last clear.

- **TIMEV: Time Event**

0 (NO\_TIMEVENT) = No time event has occurred since the last clear.

1 (TIMEVENT) = At least one time event has occurred since the last clear.

The time event is selected in the TIMEVSEL field in RTC\_CR (Control Register) and can be any one of the following events: minute change, hour change, noon, midnight (day change).

- **CALEV: Calendar Event**

0 (NO\_CALEVENT) = No calendar event has occurred since the last clear.

1 (CALEVENT) = At least one calendar event has occurred since the last clear.

The calendar event is selected in the CALEVSEL field in RTC\_CR and can be any one of the following events: week change, month change and year change.

- **TDERR: Time and/or Date Free Running Error**

0 (CORRECT) = The internal free running counters are carrying valid values since the last read of RTC\_SR.

1 (ERR\_TIMEDATE) = The internal free running counters have been corrupted (invalid date or time, non-BCD values) since the last read and/or they are still invalid.

### 18.6.8 RTC Status Clear Command Register

**Name:** RTC\_SCCR

**Address:** 0x400E187C

**Access:** Write-only

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
-	-	TDERRCLR	CALCLR	TIMCLR	SECCLR	ALRCLR	ACKCLR

- **ACKCLR: Acknowledge Clear**

0 = No effect.

1 = Clears corresponding status flag in the Status Register (RTC\_SR).

- **ALRCLR: Alarm Clear**

0 = No effect.

1 = Clears corresponding status flag in the Status Register (RTC\_SR).

- **SECCLR: Second Clear**

0 = No effect.

1 = Clears corresponding status flag in the Status Register (RTC\_SR).

- **TIMCLR: Time Clear**

0 = No effect.

1 = Clears corresponding status flag in the Status Register (RTC\_SR).

- **CALCLR: Calendar Clear**

0 = No effect.

1 = Clears corresponding status flag in the Status Register (RTC\_SR).

- **TDERRCLR: Time and/or Date Free Running Error Clear**

0 = No effect.

1 = Clears corresponding status flag in the Status Register (RTC\_SR).

### 18.6.9 RTC Interrupt Enable Register

**Name:** RTC\_IER

**Address:** 0x400E1880

**Access:** Write-only

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
—	—	TDERREN	CALEN	TIMEN	SECEN	ALREN	ACKEN

- **ACKEN: Acknowledge Update Interrupt Enable**

0 = No effect.

1 = The acknowledge for update interrupt is enabled.

- **ALREN: Alarm Interrupt Enable**

0 = No effect.

1 = The alarm interrupt is enabled.

- **SECEN: Second Event Interrupt Enable**

0 = No effect.

1 = The second periodic interrupt is enabled.

- **TIMEN: Time Event Interrupt Enable**

0 = No effect.

1 = The selected time event interrupt is enabled.

- **CALEN: Calendar Event Interrupt Enable**

0 = No effect.

1 = The selected calendar event interrupt is enabled.

- **TDERREN: Time and/or Date Error Interrupt Enable**

0 = No effect.

1 = The time and date error interrupt is enabled.

### 18.6.10 RTC Interrupt Disable Register

**Name:** RTC\_IDR

**Address:** 0x400E1884

**Access:** Write-only

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
—	—	TDERRDIS	CALDIS	TIMDIS	SECDIS	ALRDIS	ACKDIS

- **ACKDIS: Acknowledge Update Interrupt Disable**

0 = No effect.

1 = The acknowledge for update interrupt is disabled.

- **ALRDIS: Alarm Interrupt Disable**

0 = No effect.

1 = The alarm interrupt is disabled.

- **SECDIS: Second Event Interrupt Disable**

0 = No effect.

1 = The second periodic interrupt is disabled.

- **TIMDIS: Time Event Interrupt Disable**

0 = No effect.

1 = The selected time event interrupt is disabled.

- **CALDIS: Calendar Event Interrupt Disable**

0 = No effect.

1 = The selected calendar event interrupt is disabled.

- **TDERRDIS: Time and/or Date Error Interrupt Disable**

0 = No effect.

1 = The time and date error interrupt is disabled.

### 18.6.11 RTC Interrupt Mask Register

**Name:** RTC\_IMR

**Address:** 0x400E1888

**Access:** Read-only

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
-	-	-	CAL	TIM	SEC	ALR	ACK

- **ACK: Acknowledge Update Interrupt Mask**

0 = The acknowledge for update interrupt is disabled.

1 = The acknowledge for update interrupt is enabled.

- **ALR: Alarm Interrupt Mask**

0 = The alarm interrupt is disabled.

1 = The alarm interrupt is enabled.

- **SEC: Second Event Interrupt Mask**

0 = The second periodic interrupt is disabled.

1 = The second periodic interrupt is enabled.

- **TIM: Time Event Interrupt Mask**

0 = The selected time event interrupt is disabled.

1 = The selected time event interrupt is enabled.

- **CAL: Calendar Event Interrupt Mask**

0 = The selected calendar event interrupt is disabled.

1 = The selected calendar event interrupt is enabled.

### 18.6.12 RTC Valid Entry Register

**Name:** RTC\_VER  
**Address:** 0x400E188C  
**Access:** Read-only

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
-	-	-	-	NVCALALR	NVTIMALR	NVCAL	NVTIM

- NVTIM: Non-valid Time**

0 = No invalid data has been detected in RTC\_TIMR (Time Register).

1 = RTC\_TIMR has contained invalid data since it was last programmed.

- NVCAL: Non-valid Calendar**

0 = No invalid data has been detected in RTC\_CALR (Calendar Register).

1 = RTC\_CALR has contained invalid data since it was last programmed.

- NVTIMALR: Non-valid Time Alarm**

0 = No invalid data has been detected in RTC\_TIMALR (Time Alarm Register).

1 = RTC\_TIMALR has contained invalid data since it was last programmed.

- NVCALALR: Non-valid Calendar Alarm**

0 = No invalid data has been detected in RTC\_CALALR (Calendar Alarm Register).

1 = RTC\_CALALR has contained invalid data since it was last programmed.

## 19. Watchdog Timer (WDT)

### 19.1 Description

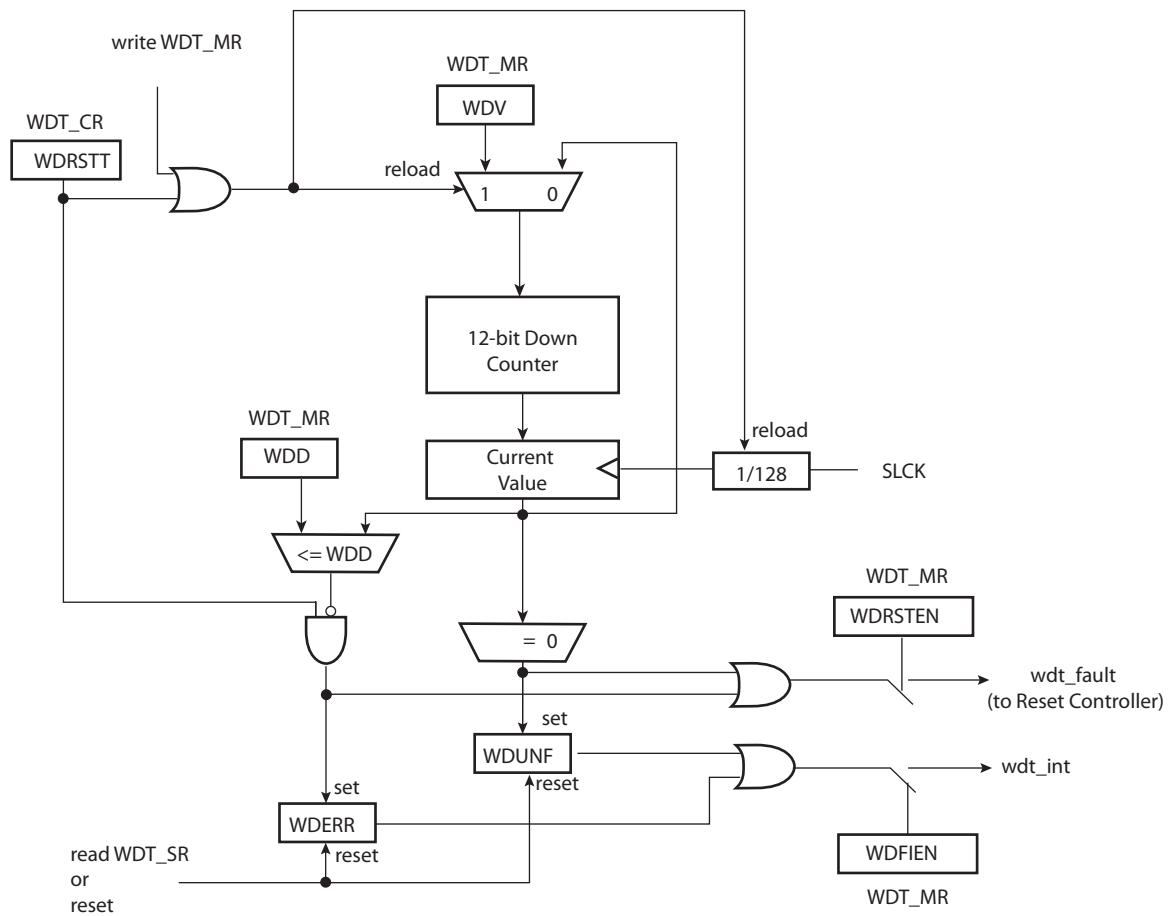
The Watchdog Timer (WDT) can be used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow clock around 32 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in debug mode or idle mode.

### 19.2 Embedded Characteristics

- 12-bit key-protected programmable counter
- Watchdog Clock is independent from Processor Clock
- Provides reset or interrupt signals to the system
- Counter may be stopped while the processor is in debug state or in idle mode

## 19.3 Block Diagram

Figure 19-1. Watchdog Timer Block Diagram



## 19.4 Functional Description

The Watchdog Timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. It is supplied with VDDCORE. It restarts with initial values on processor reset.

The Watchdog is built around a 12-bit down counter, which is loaded with the value defined in the field WDV of the Mode Register (WDT\_MR). The Watchdog Timer uses the Slow Clock divided by 128 to establish the maximum Watchdog period to be 16 seconds (with a typical Slow Clock of 32.768 kHz).

After a Processor Reset, the value of WDV is 0xFFFF, corresponding to the maximum value of the counter with the external reset generation enabled (field WDRSTEN at 1 after a Backup Reset). This means that a default Watchdog is running at reset, i.e., at power-up. The user must either disable it (by setting the WDDIS bit in WDT\_MR) if he does not expect to use it or must reprogram it to meet the maximum Watchdog period the application requires.

If the watchdog is restarted by writing into the WDT\_CR register, the WDT\_MR register must not be programmed during a period of time of 3 slow clock periods following the WDT\_CR write access. In any case, programming a new value in the WDT\_MR register automatically initiates a restart instruction.

The Watchdog Mode Register (WDT\_MR) can be written only once . Only a processor reset resets it. Writing the WDT\_MR register reloads the timer with the newly programmed mode parameters.

In normal operation, the user reloads the Watchdog at regular intervals before the timer underflow occurs, by writing the Control Register (WDT\_CR) with the bit WDRSTT to 1. The Watchdog counter is then immediately reloaded from WDT\_MR and restarted, and the Slow Clock 128 divider is reset and restarted. The WDT\_CR register is write-protected. As a result, writing WDT\_CR without the correct hard-coded key has no effect. If an underflow does occur, the “wdt\_fault” signal to the Reset Controller is asserted if the bit WDRSTEN is set in the Mode Register (WDT\_MR). Moreover, the bit WDUNF is set in the Watchdog Status Register (WDT\_SR).

To prevent a software deadlock that continuously triggers the Watchdog, the reload of the Watchdog must occur while the Watchdog counter is within a window between 0 and WDD, WDD is defined in the WatchDog Mode Register WDT\_MR.

Any attempt to restart the Watchdog while the Watchdog counter is between WDV and WDD results in a Watchdog error, even if the Watchdog is disabled. The bit WDERR is updated in the WDT\_SR and the “wdt\_fault” signal to the Reset Controller is asserted.

Note that this feature can be disabled by programming a WDD value greater than or equal to the WDV value. In such a configuration, restarting the Watchdog Timer is permitted in the whole range [0; WDV] and does not generate an error. This is the default configuration on reset (the WDD and WDV values are equal).

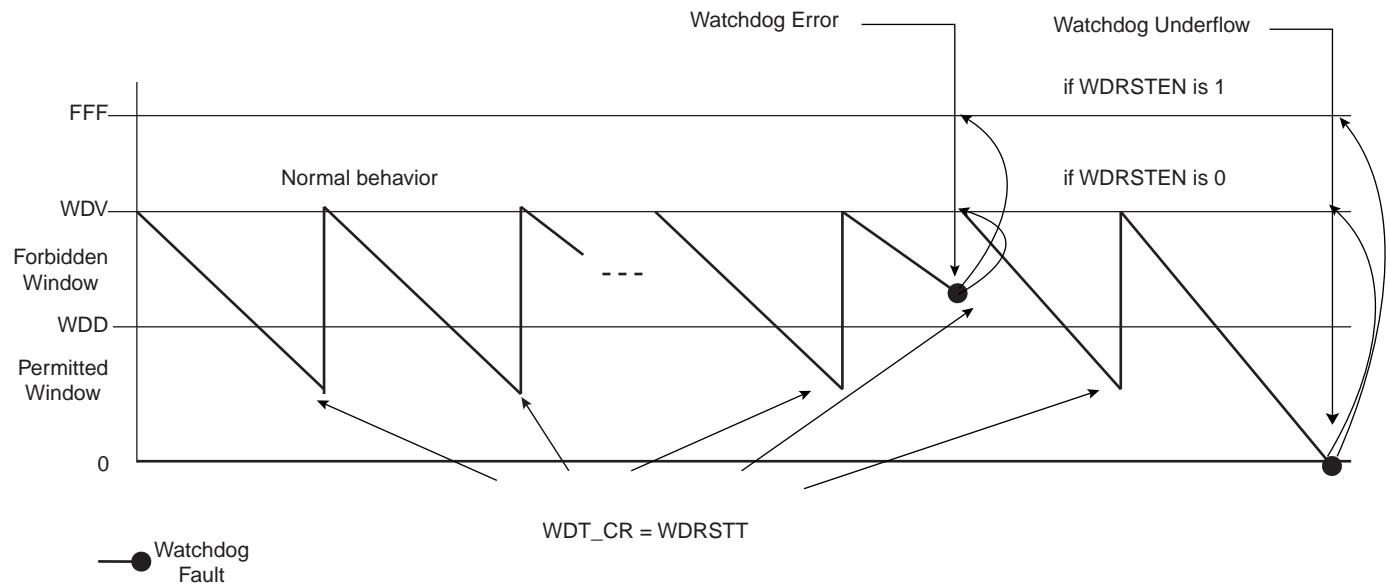
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDFIEN is set in the mode register. The signal “wdt\_fault” to the reset controller causes a Watchdog reset if the WDRSTEN bit is set as already explained in the reset controller programmer Datasheet. In that case, the processor and the Watchdog Timer are reset, and the WDERR and WDUNF flags are reset.

If a reset is generated or if WDT\_SR is read, the status bits are reset, the interrupt is cleared, and the “wdt\_fault” signal to the reset controller is deasserted.

Writing the WDT\_MR reloads and restarts the down counter.

While the processor is in debug state or in idle mode, the counter may be stopped depending on the value programmed for the bits WDIDLEHLT and WDDBGHLT in the WDT\_MR.

**Figure 19-2. Watchdog Behavior**



## 19.5 Watchdog Timer (WDT) User Interface

Table 19-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	WDT_CR	Write-only	-
0x04	Mode Register	WDT_MR	Read-write Once	0x3FFF_2FFF
0x08	Status Register	WDT_SR	Read-only	0x0000_0000

### 19.5.1 Watchdog Timer Control Register

**Name:** WDT\_CR

**Address:** 0x400E1850

**Access:** Write-only

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

- **WDRSTT: Watchdog Restart**

0: No effect.

1: Restarts the Watchdog if KEY is written to 0xA5.

- **KEY: Password.**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

### 19.5.2 Watchdog Timer Mode Register

**Name:** WDT\_MR

**Address:** 0x400E1854

**Access:** Read-write Once

31	30	29	28	27	26	25	24
		WDDLEHLT	WDBGHLT			WDD	
23	22	21	20	19	18	17	16
				WDD			
15	14	13	12	11	10	9	8
WDDIS	WDRPROC	WDRSTEN	WDFIEN			WDV	
7	6	5	4	3	2	1	0
				WDV			

Note: The first write access prevents any further modification of the value of this register, read accesses remain possible.

Note: The WDD and WDV values must not be modified within a period of time of 3 slow clock periods following a restart of the watchdog performed by means of a write access in the WDT\_CR register, else the watchdog may trigger an end of period earlier than expected.

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 12-bit Watchdog Counter.

- **WDFIEN: Watchdog Fault Interrupt Enable**

0: A Watchdog fault (underflow or error) has no effect on interrupt.

1: A Watchdog fault (underflow or error) asserts interrupt.

- **WDRSTEN: Watchdog Reset Enable**

0: A Watchdog fault (underflow or error) has no effect on the resets.

1: A Watchdog fault (underflow or error) triggers a Watchdog reset.

- **WDRPROC: Watchdog Reset Processor**

0: If WDRSTEN is 1, a Watchdog fault (underflow or error) activates all resets.

1: If WDRSTEN is 1, a Watchdog fault (underflow or error) activates the processor reset.

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the Watchdog Timer.

If the Watchdog Timer value is less than or equal to WDD, writing WDT\_CR with WDRSTT = 1 restarts the timer.

If the Watchdog Timer value is greater than WDD, writing WDT\_CR with WDRSTT = 1 causes a Watchdog error.

- **WDBGHLT: Watchdog Debug Halt**

0: The Watchdog runs when the processor is in debug state.

1: The Watchdog stops when the processor is in debug state.

- **WDIDLEHLT: Watchdog Idle Halt**

0: The Watchdog runs when the system is in idle mode.

1: The Watchdog stops when the system is in idle state.

- **WDDIS: Watchdog Disable**

0: Enables the Watchdog Timer.

1: Disables the Watchdog Timer.

### 19.5.3 Watchdog Timer Status Register

**Name:** WDT\_SR

**Address:** 0x400E1858

**Access** Read-only

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
–	–	–	–	–	–	WDERR	WDUNF

- **WDUNF: Watchdog Underflow**

0: No Watchdog underflow occurred since the last read of WDT\_SR.

1: At least one Watchdog underflow occurred since the last read of WDT\_SR.

- **WDERR: Watchdog Error**

0: No Watchdog error occurred since the last read of WDT\_SR.

1: At least one Watchdog error occurred since the last read of WDT\_SR.



## 20. Supply Controller (SUPC)

### 20.1 Description

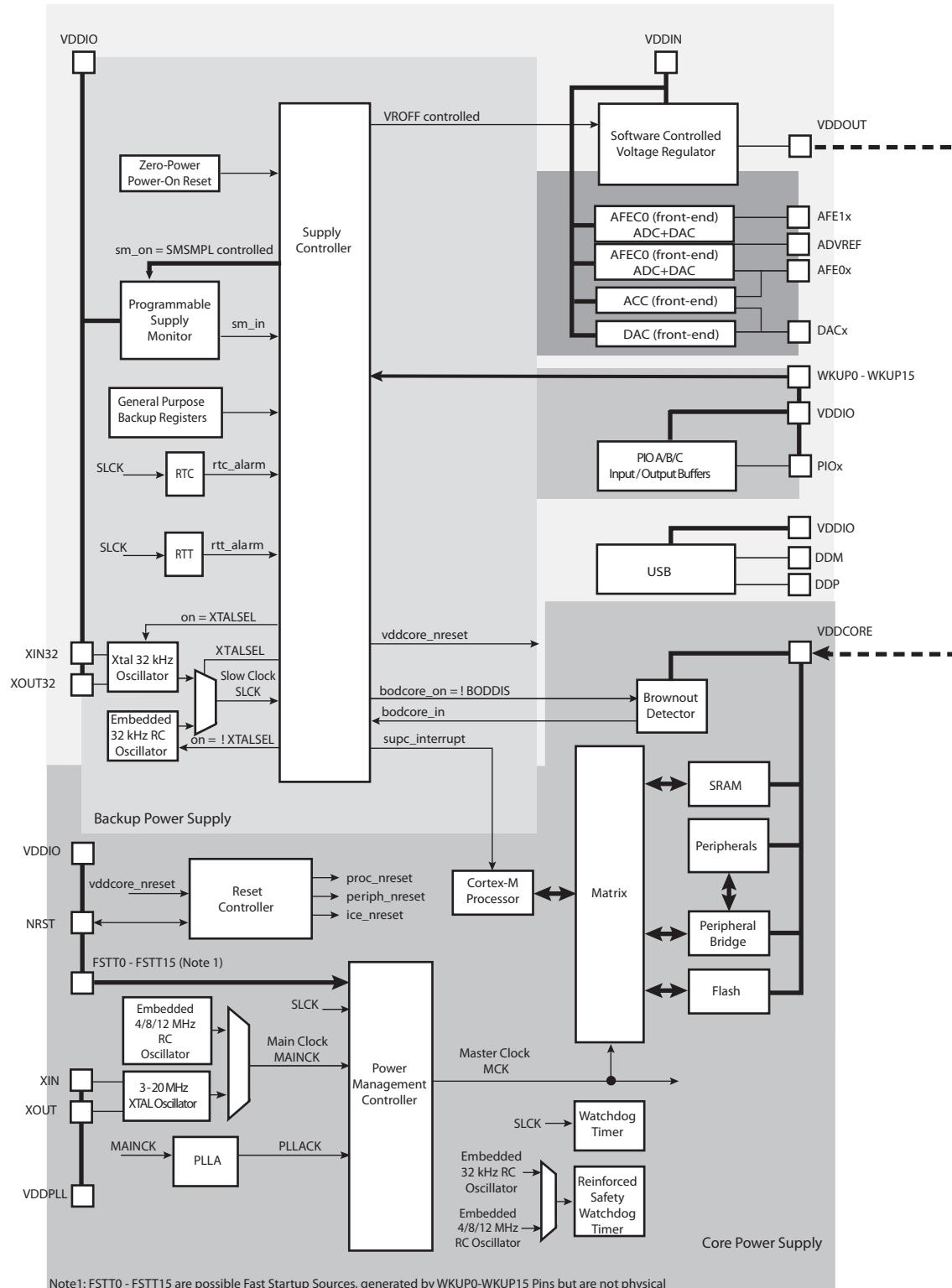
The Supply Controller (SUPC) controls the supply voltages of the system and manages the Backup Low Power Mode. In this mode, the current consumption is reduced to a few microamps for Backup power retention. Exit from this mode is possible on multiple wake-up sources. The SUPC also generates the Slow Clock by selecting either the Low Power RC oscillator or the Low Power Crystal oscillator.

### 20.2 Embedded Characteristics

- Manages the Core Power Supply VDDCORE and the Backup Low Power Mode by Controlling the Embedded Voltage Regulator
- A Supply Monitor Detection on VDDIO or a Brownout Detection on VDDCORE can Trigger a Core Reset
- Generates the Slow Clock SLCK, by Selecting Either the 22-42 kHz Low Power RC Oscillator or the 32 kHz Low Power Crystal Oscillator
- Supports Multiple Wake-up Sources, for Exit from Backup Low Power Mode
  - Force Wake-up Pin, with Programmable Debouncing
  - 16 Wake-up Inputs (including Tamper inputs), with Programmable Debouncing
  - Real Time Clock Alarm
  - Real Time Timer Alarm
  - Supply Monitor Detection on VDDIO, with Programmable Scan Period and Voltage Threshold

## 20.3 Block Diagram

Figure 20-1. Supply Controller Block Diagram



## 20.4 Supply Controller Functional Description

### 20.4.1 Supply Controller Overview

The device can be divided into two power supply areas:

- The Backup VDDIO Power Supply: including the Supply Controller, a part of the Reset Controller, the Slow Clock switch, the General Purpose Backup Registers, the Supply Monitor and the Clock which includes the Real Time Timer and the Real Time Clock
- The Core Power Supply: including the other part of the Reset Controller, the Brownout Detector, the Processor, the SRAM memory, the FLASH memory and the Peripherals

The Supply Controller (SUPC) controls the supply voltage of the core power supply. The SUPC intervenes when the VDDIO power supply rises (when the system is starting) or when the Backup Low Power Mode is entered.

The SUPC also integrates the Slow Clock generator which is based on a 32 kHz crystal oscillator and an embedded 32 kHz RC oscillator. The Slow Clock defaults to the RC oscillator, but the software can enable the crystal oscillator and select it as the Slow Clock source.

The Supply Controller and the VDDIO power supply have a reset circuitry based on a zero-power power-on reset cell. The zero-power power-on reset allows the SUPC to start properly as soon as the VDDIO voltage becomes valid.

At start-up of the system, once the backup voltage VDDIO is valid and the embedded 32 kHz RC oscillator is stabilized, the SUPC starts up the core by sequentially enabling the internal Voltage Regulator, waiting that the core voltage VDDCORE is valid, then releasing the reset signal of the core “vddcore\_nreset” signal.

Once the system has started, the user can program a supply monitor and/or a brownout detector. If the supply monitor detects a voltage on VDDIO that is too low, the SUPC can assert the reset signal of the core “vddcore\_nreset” signal until VDDIO is valid. Likewise, if the brownout detector detects a core voltage VDDCORE that is too low, the SUPC can assert the reset signal “vddcore\_nreset” until VDDCORE is valid.

When the Backup Low Power Mode is entered, the SUPC sequentially asserts the reset signal of the core power supply “vddcore\_nreset” and disables the voltage regulator, in order to supply only the VDDIO power supply. In this mode the current consumption is reduced to a few microamps for Backup part retention. Exit from this mode is possible on multiple wake-up sources including an event on FWUP pin or WKUP pins, or a Clock alarm. To exit this mode, the SUPC operates in the same way as system start-up.

#### 20.4.2 Slow Clock Generator

The Supply Controller embeds a slow clock generator that is supplied with the VDDIO power supply. As soon as the VDDIO is supplied, both the crystal oscillator and the embedded RC oscillator are powered up, but only the embedded RC oscillator is enabled. This allows the slow clock to be valid in a short time (about 100 µs).

The user can select the crystal oscillator to be the source of the slow clock, as it provides a more accurate frequency. The command is made by writing the Supply Controller Control Register (SUPC\_CR) with the XTALSEL bit at 1. This results in a sequence which first configures the PIO lines multiplexed with XIN32 and XOUT32 to be driven by the oscillator, then enables the crystal oscillator, then counts a number of slow RC oscillator clock periods to cover the start-up time of the crystal oscillator (refer to electrical characteristics for details of 32KHz crystal oscillator start-up time), then switches the slow clock on the output of the crystal oscillator and then disables the RC oscillator to save power. The switching time may vary according to the slow RC oscillator clock frequency range. The switch of the slow clock source is glitch free. The OSCSEL bit of the Supply Controller Status Register (SUPC\_SR) allows knowing when the switch sequence is done.

Coming back on the RC oscillator is only possible by shutting down the VDDIO power supply.

If the user does not need the crystal oscillator, the XIN32 and XOUT32 pins should be left unconnected.

The user can also set the crystal oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin are given in the product electrical characteristics section. In order to set the bypass mode, the OSCBYPASS bit of the Supply Controller Mode Register (SUPC\_MR) needs to be set at 1.

#### 20.4.3 Core Voltage Regulator Control/Backup Low Power Mode

The Supply Controller can be used to control the embedded voltage regulator.

The voltage regulator automatically adapts its quiescent current depending on the required load current. Please refer to the electrical characteristics section.

The programmer can switch off the voltage regulator, and thus put the device in Backup mode, by writing the Supply Controller Control Register (SUPC\_CR) with the VROFF bit at 1.

This asserts the vddcore\_nreset signal after the write resynchronization time which lasts, in the worse case, two slow clock cycles. Once the vddcore\_nreset signal is asserted, the processor and the peripherals are stopped one slow clock cycle before the core power supply shuts off.

When the user does not use the internal voltage regulator and wants to supply VDDCORE by an external supply, it is possible to disable the voltage regulator. This is done through ONREG bit in SUPC\_MR.

#### 20.4.4 Supply Monitor

The Supply Controller embeds a supply monitor which is located in the VDDIO Power Supply and which monitors VDDIO power supply.

The supply monitor can be used to prevent the processor from falling into an unpredictable state if the Main power supply drops below a certain level.

The threshold of the supply monitor is programmable. It can be selected from 1.9V to 3.4V by steps of 100 mV. This threshold is programmed in the SMTH field of the Supply Controller Supply Monitor Mode Register (SUPC\_SMMR).

The supply monitor can also be enabled during one slow clock period on every one of either 32, 256 or 2048 slow clock periods, according to the choice of the user. This can be configured by programming the SMSMPL field in SUPC\_SMMR.

Enabling the supply monitor for such reduced times allows to divide the typical supply monitor power consumption respectively by factors of 32, 256 or 2048, if the user does not need a continuous monitoring of the VDDIO power supply.

A supply monitor detection can either generate a reset of the core power supply or a wake-up of the core power supply. Generating a core reset when a supply monitor detection occurs is enabled by writing the SMRSTEN bit to 1 in SUPC\_SMMR.

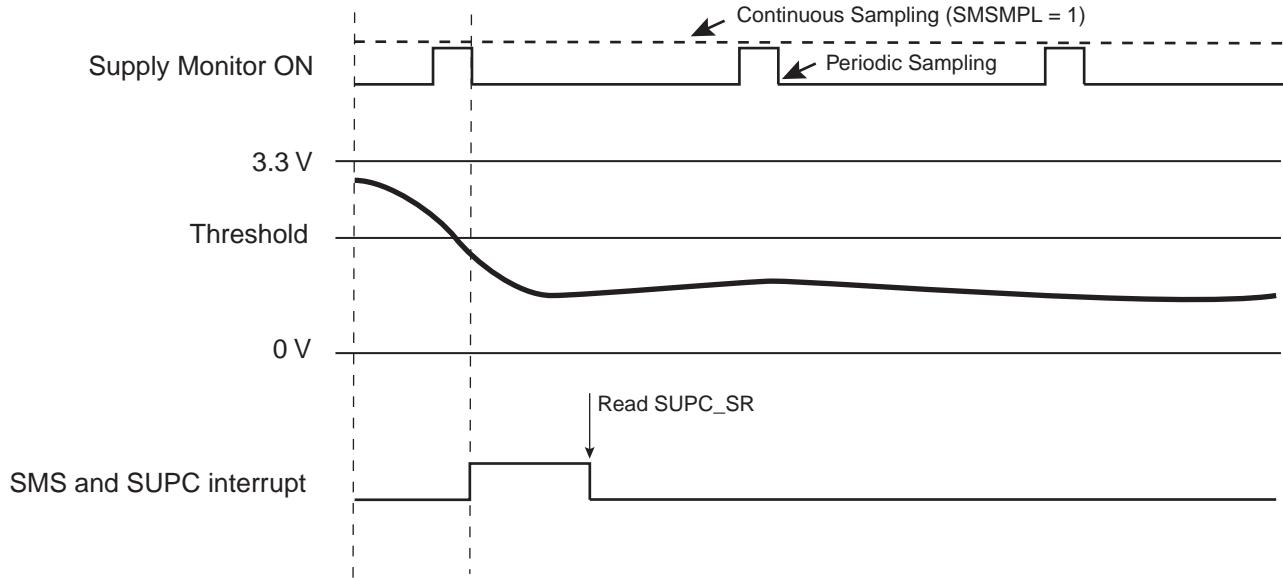
Waking up the core power supply when a supply monitor detection occurs can be enabled by programming the SMEN bit to 1 in the Supply Controller Wake-up Mode Register (SUPC\_WUMR).

The Supply Controller provides two status bits in the Supply Controller Status Register for the supply monitor which allows to determine whether the last wake-up was due to the supply monitor:

- The SMOS bit provides real time information, which is updated at each measurement cycle or updated at each Slow Clock cycle, if the measurement is continuous.
- The SMS bit provides saved information and shows a supply monitor detection has occurred since the last read of SUPC\_SR.

The SMS bit can generate an interrupt if the SMIEN bit is set to 1 in the Supply Controller Supply Monitor Mode Register (SUPC\_SMMR).

**Figure 20-2. Supply Monitor Status Bit and Associated Interrupt**



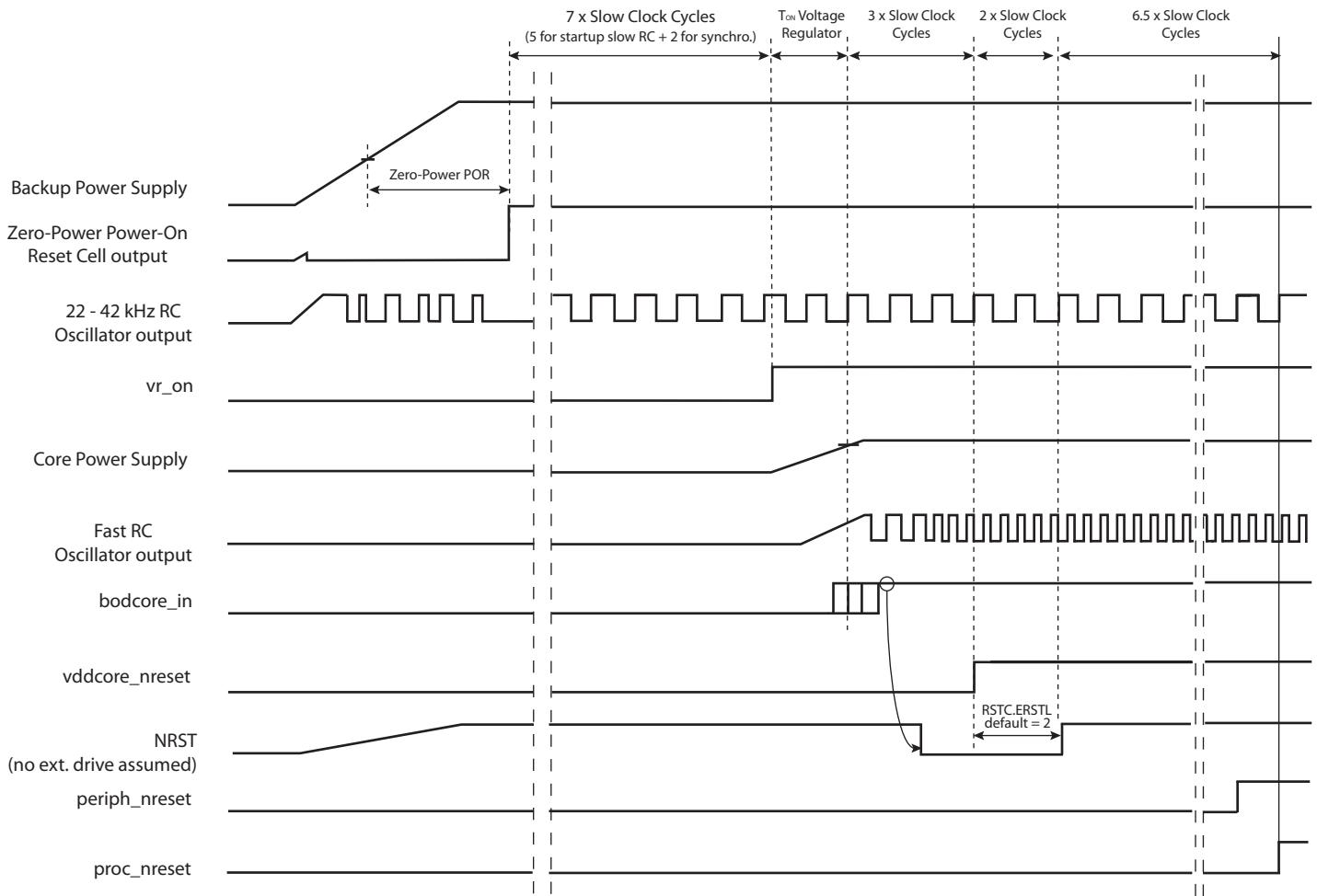
## 20.4.5 Backup Power Supply Reset

### 20.4.5.1 Raising the Backup Power Supply

As soon as the backup voltage VDDIO rises, the RC oscillator is powered up and the zero-power power-on reset cell maintains its output low as long as VDDIO has not reached its target voltage. During this time, the Supply Controller is entirely reset. When the VDDIO voltage becomes valid and zero-power power-on reset signal is released, a counter is started for 5 slow clock cycles. This is the time it takes for the 32 kHz RC oscillator to stabilize.

After this time, the voltage regulator is enabled. The core power supply rises and the brownout detector provides the bodcore\_in signal as soon as the core voltage VDDCORE is valid. This results in releasing the vddcore\_nreset signal to the Reset Controller after the bodcore\_in signal has been confirmed as being valid for at least one slow clock cycle.

**Figure 20-3. Raising the VDDIO Power Supply**



#### 20.4.6 Core Reset

The Supply Controller manages the vddcore\_nreset signal to the Reset Controller, as described previously in [Section 20.4.5 "Backup Power Supply Reset"](#). The vddcore\_nreset signal is normally asserted before shutting down the core power supply and released as soon as the core power supply is correctly regulated.

There are two additional sources which can be programmed to activate vddcore\_nreset:

- a supply monitor detection
- a brownout detection

##### 20.4.6.1 Supply Monitor Reset

The supply monitor is capable of generating a reset of the system. This can be enabled by setting the SMRSTEN bit in the Supply Controller Supply Monitor Mode Register (SUPC\_SMMR).

If SMRSTEN is set and if a supply monitor detection occurs, the vddcore\_nreset signal is immediately activated for a minimum of 1 slow clock cycle.

##### 20.4.6.2 Brownout Detector Reset

The brownout detector provides the bodcore\_in signal to the SUPC which indicates that the voltage regulation is operating as programmed. If this signal is lost for longer than 1 slow clock period while the voltage regulator is enabled,

the Supply Controller can assert vddcore\_nreset. This feature is enabled by writing the bit, BODRSTEN (Brownout Detector Reset Enable) to 1 in the Supply Controller Mode Register (SUPC\_MR).

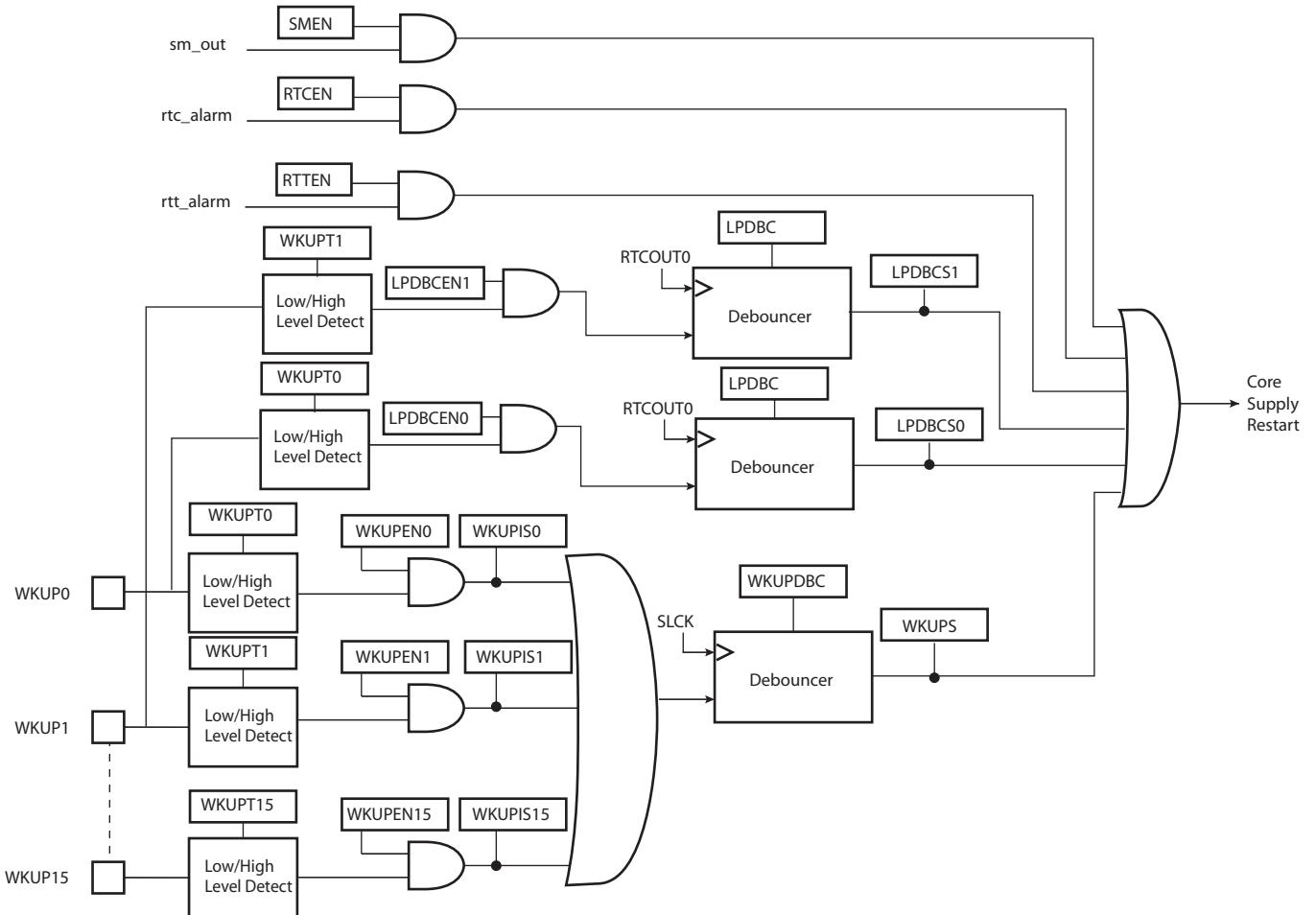
If BODRSTEN is set and the voltage regulation is lost (output voltage of the regulator too low), the vddcore\_nreset signal is asserted for a minimum of 1 slow clock cycle and then released if bodcore\_in has been reactivated. The BODRSTS bit is set in the Supply Controller Status Register (SUPC\_SR) so that the user can know the source of the last reset.

Until bodcore\_in is deactivated, the vddcore\_nreset signal remains active.

#### 20.4.7 Wake-up Sources

The wake-up events allow the device to exit backup mode. When a wake-up event is detected, the Supply Controller performs a sequence which automatically reenables the core power supply.

**Figure 20-4. Wake-up Sources**



##### 20.4.7.1 Force Wake-up

The FWUP pin is enabled as a wake-up source by writing the FWUPEN bit to 1 in the Supply Controller Wake-up Mode Register (SUPC\_WUMR). Then, the FWUPDBC field in the same register selects the debouncing period, which can be selected between 3, 32, 512, 4,096 or 32,768 slow clock cycles. This corresponds respectively to about 100 µs, about 1 ms, about 16 ms, about 128 ms and about 1 second (for a typical slow clock frequency of 32 kHz). Programming FWUPDBC to 0x0 selects an immediate wake-up, i.e., the FWUP must be low during a minimum of one slow clock period to wake-up the core power supply.

If the FWUP pin is asserted for a time longer than the debouncing period, a wake-up of the core power supply is started and the FWUP bit in the Supply Controller Status Register (SUPC\_SR) is set and remains high until the register is read.

#### 20.4.7.2 Wake-up Inputs

The wake-up inputs, WKUP0 to WKUP15, can be programmed to perform a wake-up of the core power supply. Each input can be enabled by writing to 1 the corresponding bit, WKUPEN0 to WKUPEN 15, in the Wake-up Inputs Register (SUPC\_WUIR). The wake-up level can be selected with the corresponding polarity bit, WKUPPL0 to WKUPPL15, also located in SUPC\_WUIR.

All the resulting signals are wired-ORed to trigger a debounce counter, which can be programmed with the WKUPDBC field in the Supply Controller Wake-up Mode Register (SUPC\_WUMR). The WKUPDBC field can select a debouncing period of 3, 32, 512, 4,096 or 32,768 slow clock cycles. This corresponds respectively to about 100 µs, about 1 ms, about 16 ms, about 128 ms and about 1 second (for a typical slow clock frequency of 32 kHz). Programming WKUPDBC to 0x0 selects an immediate wake-up, i.e., an enabled WKUP pin must be active according to its polarity during a minimum of one slow clock period to wake up the core power supply.

If an enabled WKUP pin is asserted for a time longer than the debouncing period, a wake-up of the core power supply is started and the signals, WKUP0 to WKUP15 as shown in [Figure 20-4](#), are latched in the Supply Controller Status Register (SUPC\_SR). This allows the user to identify the source of the wake-up, however, if a new wake-up condition occurs, the primary information is lost. No new wake-up can be detected since the primary wake-up condition has disappeared.

#### 20.4.7.3 Low-power Debouncer Inputs

It is possible to generate a waveform (RTCOUT0) in all modes (including backup mode). It can be useful to control an external sensor and/or tampering function without waking up the processor. Please refer to the RTC section for waveform generation.

Two separate debouncers are embedded for WKUP0 and WKUP1 inputs.

The WKUP0 and/or WKUP1 inputs can be programmed to perform a wake-up of the core power supply with a debouncing done by RTCOUT0.

These inputs can be also used when VDDCORE is powered to get tamper detection function with a low power debounce function.

This can be enabled by setting LPDBC0 bit and/or LPDBC1 bit in SUPC\_WUMR.

In this mode of operation, WKUP0 and/or WKUP1 must not be configured to also act as debouncing source for the WKUPDBC counter (WKUPEN0 and/or WKUPEN1 must be cleared in SUPC\_WUIR). Refer to [Section 20.4.7.1 "Force Wake-up"](#).

This mode of operation requires the RTC Output (RTCOUT0) to be configured to generate a duty cycle programmable pulse (i.e. OUT0 = 0x7 in RTC\_MR) in order to create the sampling points of both debouncers. The sampling point is the falling edge of the RTCOUT0 waveform.

[Figure 20-5](#) shows an example of an application where two tamper switches are used. RTCOUT0 powers the external pull-up used by the tampers.

Figure 20-5. Low Power Debouncer (Push-to-Make switch, pull-up resistors)

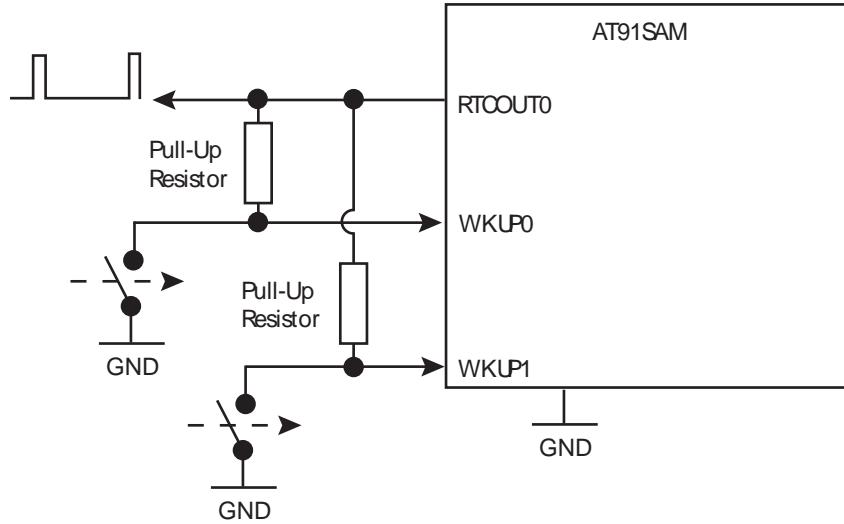
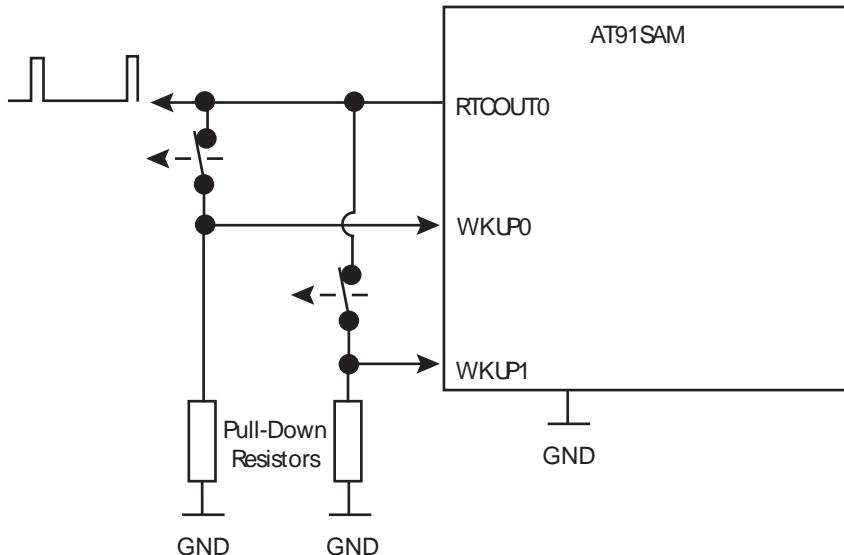


Figure 20-6. Low Power Debouncer (Push-to-Break switch, pull-down resistors)



The debouncing parameters can be adjusted and are shared (except the wake-up input polarity) by both debouncers. The number of successive identical samples to wake up the core can be configured from 2 up to 8 in the LPDBC field of SUPC\_WUMR. The period of time between 2 samples can be configured by programming the TPERIOD field in the RTC\_MR register.

Power parameters can be adjusted by modifying the period of time in the THIGH field in RTC\_MR.

The wake-up polarity of the inputs can be independently configured by writing WKUPT0 and WKUPT1 fields in SUPC\_WUMR.

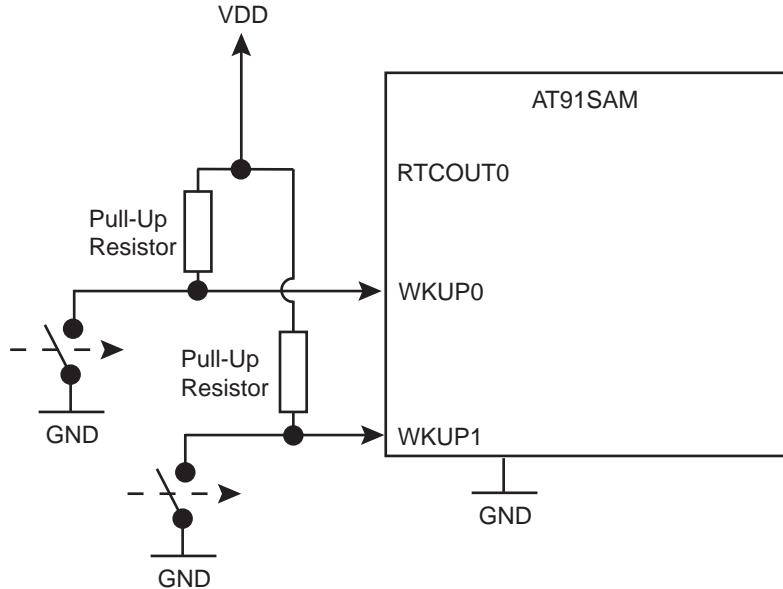
In order to determine which wake-up pin triggers the core wake-up or simply which debouncer triggers an event (even if there is no wake-up, so when VDDCORE is powered on), a status flag is associated for each low power debouncer. These 2 flags can be read in the SUPC\_SR.

A debounce event can perform an immediate clear (0 delay) on first half the general purpose backup registers (GPBR). The LPDBCCCLR bit must be set to 1 in SUPC\_MR.

Please note that it is not mandatory to use the RTCOUT pins when using the WKUP0/WKUP1 pins as tampering inputs (TMP0/TMP1) in backup mode or any other modes. Using RTCOUT0 pins provides a “sampling mode” to further reduce the power consumption in low power modes. However the RTC must be configured in the same manner as RTCOUT0 is used in order to create a sampling point for the debouncer logic.

[Figure 20-7](#) shows how to use WKUP0/WKUP1 without RTCOUT pins.

**Figure 20-7. Using WKUP0/WKUP1 without RTCOUT Pins**



#### 20.4.7.4 Low-power Tamper Detection Inputs

WKUP0 and WKUP1 can be used as tamper detect inputs.

In Backup Mode they can be used also to wake up the core.

If a tamper is detected, it performs an immediate clear (0 delay) on first half the general purpose backup registers (GPBR).

Refer to [“Wake-up Sources” on page 341](#) for more details.

#### 20.4.7.5 Clock Alarms

The RTC and the RTT alarms can generate a wake-up of the core power supply. This can be enabled by writing respectively, the bits RTCEN and RTTEN to 1 in the Supply Controller Wake-up Mode Register (SUPC\_WUMR).

The Supply Controller does not provide any status as the information is available in the User Interface of either the Real Time Timer or the Real Time Clock.

#### 20.4.7.6 Supply Monitor Detection

The supply monitor can generate a wake-up of the core power supply. See [Section 20.4.4 “Supply Monitor”](#).

## 20.5 Supply Controller (SUPC) User Interface

The User Interface of the Supply Controller is part of the System Controller User Interface.

### 20.5.1 System Controller (SYSC) User Interface

Table 20-1. System Controller Registers

Offset	System Controller Peripheral	Name
0x00-0x0c	Reset Controller	RSTC
0x10-0x2C	Supply Controller	SUPC
0x30-0x3C	Real Time Timer	RTT
0x50-0x5C	Watchdog Timer	WDT
0x60-0x8C	Real Time Clock	RTC
0x90-0xDC	General Purpose Backup Register	GPBR
0xE0	Reserved	
0xE4	Write Protect Mode Register	SYSC_WPMR
0xE8-0xF8	Reserved	
0x100-0x10C	Reinforced Safety Watchdog Timer	RSWDT

### 20.5.2 Supply Controller (SUPC) User Interface

Table 20-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Supply Controller Control Register	SUPC_CR	Write-only	N/A
0x04	Supply Controller Supply Monitor Mode Register	SUPC_SMMR	Read-write	0x0000_0000
0x08	Supply Controller Mode Register	SUPC_MR	Read-write	0x0000_5A00
0x0C	Supply Controller Wake-up Mode Register	SUPC_WUMR	Read-write	0x0000_0000
0x10	Supply Controller Wake-up Inputs Register	SUPC_WUIR	Read-write	0x0000_0000
0x14	Supply Controller Status Register	SUPC_SR	Read-only	0x0000_0000
0x18	Reserved			

### 20.5.3 Supply Controller Control Register

**Name:** SUPC\_CR

**Address:** 0x400E1810

**Access:** Write-only

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

- **VROFF: Voltage Regulator Off**

0 (NO\_EFFECT) = no effect.

1 (STOP\_VREG) = if KEY is correct, asserts the vddcore\_nreset and stops the voltage regulator.

- **XTALSEL: Crystal Oscillator Select**

0 (NO\_EFFECT) = no effect.

1 (CRYSTAL\_SEL) = if KEY is correct, switches the slow clock on the crystal oscillator output.

- **KEY: Password**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

#### 20.5.4 Supply Controller Supply Monitor Mode Register

**Name:** SUPC\_SMMR

**Address:** 0x400E1814

**Access:** Read-write

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

- **SMTH: Supply Monitor Threshold**

Allows to select the threshold voltage of the supply monitor. Refer to electrical characteristics for voltage values.

- **SMSMPL: Supply Monitor Sampling Period**

Value	Name	Description
0x0	SMD	Supply Monitor disabled
0x1	CSM	Continuous Supply Monitor
0x2	32SLCK	Supply Monitor enabled one SLCK period every 32 SLCK periods
0x3	256SLCK	Supply Monitor enabled one SLCK period every 256 SLCK periods
0x4	2048SLCK	Supply Monitor enabled one SLCK period every 2,048 SLCK periods

- **SMRSTEN: Supply Monitor Reset Enable**

0 (NOT\_ENABLE) = the core reset signal “vddcore\_nreset” is not affected when a supply monitor detection occurs.

1 (ENABLE) = the core reset signal, vddcore\_nreset is asserted when a supply monitor detection occurs.

- **SMIEN: Supply Monitor Interrupt Enable**

0 (NOT\_ENABLE) = the SUPC interrupt signal is not affected when a supply monitor detection occurs.

1 (ENABLE) = the SUPC interrupt signal is asserted when a supply monitor detection occurs.

### 20.5.5 Supply Controller Mode Register

**Name:** SUPC\_MR

**Address:** 0x400E1818

**Access:** Read-write

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

- **BODRSTEN: Brownout Detector Reset Enable**

0 (NOT\_ENABLE) = the core reset signal “vddcore\_nreset” is not affected when a brownout detection occurs.

1 (ENABLE) = the core reset signal, vddcore\_nreset is asserted when a brownout detection occurs.

- **BODDIS: Brownout Detector Disable**

0 (ENABLE) = the core brownout detector is enabled.

1 (DISABLE) = the core brownout detector is disabled.

- **ONREG: Voltage Regulator enable**

0 (ONREG\_UNUSED) = Internal voltage regulator is not used (external power supply is used)

1 (ONREG\_USED) = internal voltage regulator is used

- **OSCBYPASS: Oscillator Bypass**

0 (NO\_EFFECT) = no effect. Clock selection depends on XTALSEL value.

1 (BYPASS) = the 32-KHz XTAL oscillator is selected and is put in bypass mode.

- **KEY: Password Key**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

## 20.5.6 Supply Controller Wake-up Mode Register

**Name:** SUPC\_WUMR

**Address:** 0x400E181C

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	LPDBC	
15	14	13	12	11	10	9	8
—	WKUPDBC		—	—	FWUPDBC		
7	6	5	4	3	2	1	0
LPDBCCCLR	LPDBCEN1	LPDBCENO	—	RTCEN	RTTEN	SMEN	FWUPEN

- **FWUPEN: Force Wake-up Enable**

0 (NOT\_ENABLE) = the Force Wake-up pin has no wake-up effect.

1 (ENABLE) = the Force Wake-up pin low forces the wake-up of the core power supply.

- **SMEN: Supply Monitor Wake-up Enable**

0 (NOT\_ENABLE) = the supply monitor detection has no wake-up effect.

1 (ENABLE) = the supply monitor detection forces the wake-up of the core power supply.

- **RTTEN: Real Time Timer Wake-up Enable**

0 (NOT\_ENABLE) = the RTT alarm signal has no wake-up effect.

1 (ENABLE) = the RTT alarm signal forces the wake-up of the core power supply.

- **RTCEN: Real Time Clock Wake-up Enable**

0 (NOT\_ENABLE) = the RTC alarm signal has no wake-up effect.

1 (ENABLE) = the RTC alarm signal forces the wake-up of the core power supply.

- **LPDBCEN0: Low power Debouncer ENable WKUP0**

0 (NOT\_ENABLE) = the WKUP0 input pin is not connected with low power debouncer.

1 (ENABLE) = the WKUP0 input pin is connected with low power debouncer and can force a core wake-up.

- **LPDBCEN1: Low power Debouncer ENable WKUP1**

0 (NOT\_ENABLE) = the WKUP1 input pin is not connected with low power debouncer.

1 (ENABLE) = the WKUP1 input pin is connected with low power debouncer and can force a core wake-up.

- **LPDBCCCLR: Low power Debouncer Clear**

0 (NOT\_ENABLE) = a low power debounce event does not create an immediate clear on first half GPBR registers.

1 (ENABLE) = a low power debounce event on WKUP0 or WKUP1 generates an immediate clear on first half GPBR registers.

- **FWUPDBC: Force Wake-up Debouncer Period**

Value	Name	Description
0	IMMEDIATE	Immediate, no debouncing, detected active at least on one Slow Clock edge.
1	3_SCLK	FWUP shall be low for at least 3 SLCK periods
2	32_SCLK	FWUP shall be low for at least 32 SLCK periods
3	512_SCLK	FWUP shall be low for at least 512 SLCK periods
4	4096_SCLK	FWUP shall be low for at least 4,096 SLCK periods
5	32768_SCLK	FWUP shall be low for at least 32,768 SLCK periods
6	Reserved	Reserved
7	Reserved	Reserved

- **WKUPDBC: Wake-up Inputs Debouncer Period**

Value	Name	Description
0	IMMEDIATE	Immediate, no debouncing, detected active at least on one Slow Clock edge.
1	3_SCLK	WKUPx shall be in its active state for at least 3 SLCK periods
2	32_SCLK	WKUPx shall be in its active state for at least 32 SLCK periods
3	512_SCLK	WKUPx shall be in its active state for at least 512 SLCK periods
4	4096_SCLK	WKUPx shall be in its active state for at least 4,096 SLCK periods
5	32768_SCLK	WKUPx shall be in its active state for at least 32,768 SLCK periods
6	Reserved	Reserved
7	Reserved	Reserved

- **LPDBC: Low Power DeBounCer Period**

Value	Name	Description
0	DISABLE	Disable the low power debouncer.
1	2_RTCOUT0	WKUP0/1 in its active state for at least 2 RTCOUT0 periods
2	3_RTCOUT0	WKUP0/1 in its active state for at least 3 RTCOUT0 periods
3	4_RTCOUT0	WKUP0/1 in its active state for at least 4 RTCOUT0 periods
4	5_RTCOUT0	WKUP0/1 in its active state for at least 5 RTCOUT0 periods
5	6_RTCOUT0	WKUP0/1 in its active state for at least 6 RTCOUT0 periods
6	7_RTCOUT0	WKUP0/1 in its active state for at least 7 RTCOUT0 periods
7	8_RTCOUT0	WKUP0/1 in its active state for at least 8 RTCOUT0 periods

### 20.5.7 System Controller Wake-up Inputs Register

**Name:** SUPC\_WUIR

**Address:** 0x400E1820

**Access:** Read-write

31	30	29	28	27	26	25	24
WKUPT15	WKUPT14	WKUPT13	WKUPT12	WKUPT11	WKUPT10	WKUPT9	WKUPT8
23	22	21	20	19	18	17	16
WKUPT7	WKUPT6	WKUPT5	WKUPT4	WKUPT3	WKUPT2	WKUPT1	WKUPT0
15	14	13	12	11	10	9	8
WKUPEN15	WKUPEN14	WKUPEN13	WKUPEN12	WKUPEN11	WKUPEN10	WKUPEN9	WKUPEN8
7	6	5	4	3	2	1	0
WKUPEN7	WKUPEN6	WKUPEN5	WKUPEN4	WKUPEN3	WKUPEN2	WKUPEN1	WKUPEN0

- **WKUPEN0 - WKUPEN15: Wake-up Input Enable 0 to 15**

0 (DISABLE) = the corresponding wake-up input has no wake-up effect.

1 (ENABLE) = the corresponding wake-up input forces the wake-up of the core power supply.

- **WKUPT0 - WKUPT15: Wake-up Input Type 0 to 15**

0 (LOW) = a low level for a period defined by WKUPDBC on the corresponding wake-up input forces the wake-up of the core power supply.

1 (HIGH) = a high level for a period defined by WKUPDBC on the corresponding wake-up input forces the wake-up of the core power supply.

## 20.5.8 Supply Controller Status Register

**Name:** SUPC\_SR

**Address:** 0x400E1824

**Access:** Read-only

31	30	29	28	27	26	25	24
WKUPIS15	WKUPIS14	WKUPIS13	WKUPIS12	WKUPIS11	WKUPIS10	WKUPIS9	WKUPIS8
23	22	21	20	19	18	17	16
WKUPIS7	WKUPIS6	WKUPIS5	WKUPIS4	WKUPIS3	WKUPIS2	WKUPIS1	WKUPIS0
15	14	13	12	11	10	9	8
—	LPDBCS1	LPDBCS0	FWUPIS	—	—	—	—
7	6	5	4	3	2	1	0
OSCSEL	SMOS	SMS	SMRSTS	BODRSTS	SMWS	WKUPS	FWUPS

**Note:** Because of the asynchronism between the Slow Clock (SCLK) and the System Clock (MCK), the status register flag reset is taken into account only 2 slow clock cycles after the read of the SUPC\_SR.

- **FWUPS: FWUP Wake-up Status**

0 (NO) = no wake-up due to the assertion of the FWUP pin has occurred since the last read of SUPC\_SR.

1 (PRESENT) = at least one wake-up due to the assertion of the FWUP pin has occurred since the last read of SUPC\_SR.

- **WKUPS: WKUP Wake-up Status**

0 (NO) = no wake-up due to the assertion of the WKUP pins has occurred since the last read of SUPC\_SR.

1 (PRESENT) = at least one wake-up due to the assertion of the WKUP pins has occurred since the last read of SUPC\_SR.

- **SMWS: Supply Monitor Detection Wake-up Status**

0 (NO) = no wake-up due to a supply monitor detection has occurred since the last read of SUPC\_SR.

1 (PRESENT) = at least one wake-up due to a supply monitor detection has occurred since the last read of SUPC\_SR.

- **BODRSTS: Brownout Detector Reset Status**

0 (NO) = no core brownout rising edge event has been detected since the last read of the SUPC\_SR.

1 (PRESENT) = at least one brownout output rising edge event has been detected since the last read of the SUPC\_SR.

When the voltage remains below the defined threshold, there is no rising edge event at the output of the brownout detection cell. The rising edge event occurs only when there is a voltage transition below the threshold.

- **SMRSTS: Supply Monitor Reset Status**

0 (NO) = no supply monitor detection has generated a core reset since the last read of the SUPC\_SR.

1 (PRESENT) = at least one supply monitor detection has generated a core reset since the last read of the SUPC\_SR.

- **SMS: Supply Monitor Status**

0 (NO) = no supply monitor detection since the last read of SUPC\_SR.

1 (PRESENT) = at least one supply monitor detection since the last read of SUPC\_SR.

- **SMOS: Supply Monitor Output Status**

0 (HIGH) = the supply monitor detected VDDIO higher than its threshold at its last measurement.

1 (LOW) = the supply monitor detected VDDIO lower than its threshold at its last measurement.

- **OSCSEL: 32-kHz Oscillator Selection Status**

0 (RC) = the slow clock, SLCK is generated by the embedded 32-kHz RC oscillator.

1 (CRYST) = the slow clock, SLCK is generated by the 32-kHz crystal oscillator.

- **FWUPIS: FWUP Input Status**

0 (LOW) = FWUP input is tied low.

1 (HIGH) = FWUP input is tied high.

- **LPDBCS0: Low Power Debouncer Wake-up Status on WKUP0**

0 (NO) = no wake-up due to the assertion of the WKUP0 pin has occurred since the last read of SUPC\_SR.

1 (PRESENT) = at least one wake-up due to the assertion of the WKUP0 pin has occurred since the last read of SUPC\_SR.

- **LPDBCS1: Low Power Debouncer Wake-up Status on WKUP1**

0 (NO) = no wake-up due to the assertion of the WKUP1 pin has occurred since the last read of SUPC\_SR.

1 (PRESENT) = at least one wake-up due to the assertion of the WKUP1 pin has occurred since the last read of SUPC\_SR.

- **WKUPIS0-WKUPIS15: WKUP Input Status 0 to 15**

0 (DIS) = the corresponding wake-up input is disabled, or was inactive at the time the debouncer triggered a wake-up event.

1 (EN) = the corresponding wake-up input was active at the time the debouncer triggered a wake-up event.

### 20.5.9 System Controller Write Protect Mode Register

**Name:** SYSC\_WPMR

**Access:** Read-write

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

- **WPEN:**

0: The Write Protection is disabled.

1: The Write Protection is enabled.

List of the write-protected registers:

RSTC Mode Register

RTT Mode Register

RTT Alarm Register

RTC Control Register

RTC Mode Register

RTC Time Alarm Register

RTC Calendar Alarm Register

General Purpose Backup Registers

SUPC Control Register

SUPC Supply Monitor Mode Register

SUPC Mode Register

SUPC Wake-up Mode Register

SUPC Wake-up Input Mode Register

- **WPKEY:**

Value	Name	Description
0x525443	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

## 21. General Purpose Backup Registers (GPBR)

### 21.1 Description

The System Controller embeds 20 General Purpose Backup Registers (GPBR).

It is possible to generate an immediate clear of the content of general purpose backup registers 0 to 9 (first half), if a low power debounce event is detected on a wakeup pin, WKUP0 or WKUP1. The content of the other general-purpose backup registers (second half) remains unchanged.

To enter this mode of operation, the supply controller module must be programmed accordingly. In supply controller SUPC\_WUMR register, LPDBCCLR, LPDBCEN0 and/or LPDBCEN1 bit must be configured to 1 and LPDBC must be other than 0.

If a tamper event has been detected, it is not possible to write into general purpose backup registers while the LPDBCS0 or LPDBCS1 flags are not cleared in supply controller status register SUPC\_SR.

### 21.2 Embedded Characteristics

- Twenty 32-bit General Purpose Backup Registers

## 21.3 General Purpose Backup Register (GPBR) User Interface

Table 21-1. Register Mapping

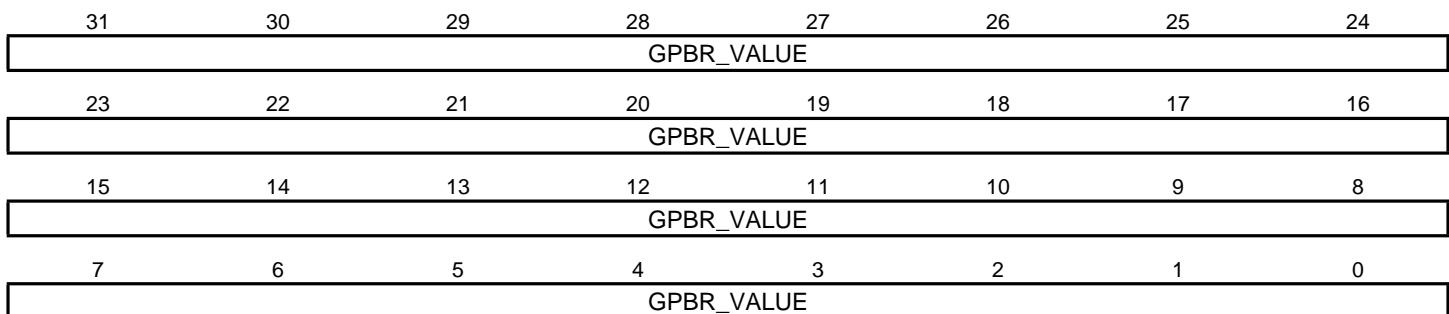
Offset	Register	Name	Access	Reset
0x0	General Purpose Backup Register 0	SYS_GPBR0	Read-write	–
...	...	...	...	...
0x64	General Purpose Backup Register 19	SYS_GPBR19	Read-write	–

### 21.3.1 General Purpose Backup Register x

**Name:** SYS\_GPBRx

**Address:** 0x400E1890

**Access:** Read-write



- **GPBR\_VALUE: Value of GPBR x**

If a tamper event has been detected, it is not possible to write GPBR\_VALUE while the LPDBCS0 or LPDBCS1 flags are not cleared in supply controller status register SUPC\_SR.

## 22. Embedded Flash Controller (EFC)

### 22.1 Description

The Enhanced Embedded Flash Controller (EEFC) ensures the interface of the Flash block with the 32-bit internal bus. Its 128-bit or 64-bit wide memory interface increases performance. It also manages the programming, erasing, locking and unlocking sequences of the Flash using a full set of commands. One of the commands returns the embedded Flash descriptor definition that informs the system about the Flash organization, thus making the software generic.

### 22.2 Embedded Characteristics

- Interface of the Flash Block with the 32-bit Internal Bus
- Increases Performance in Thumb2 Mode with 128-bit or -64 bit Wide Memory Interface up to 120 MHz
- Code loops optimization
- 128 Lock Bits, Each Protecting a Lock Region
- GPNVMx General-purpose GPNVM Bits
- One-by-one Lock Bit Programming
- Commands Protected by a Keyword
- Erases the Entire Flash
- Erases by Plane
- Erase by Sector
- Erase by Pages
- Possibility of Erasing before Programming
- Locking and Unlocking Operations
- Possibility to read the Calibration Bits

### 22.3 Product Dependencies

#### 22.3.1 Power Management

The Enhanced Embedded Flash Controller (EEFC) is continuously clocked. The Power Management Controller has no effect on its behavior.

#### 22.3.2 Interrupt Sources

The Enhanced Embedded Flash Controller (EEFC) interrupt line is connected to the Nested Vectored Interrupt Controller (NVIC). Using the Enhanced Embedded Flash Controller (EEFC) interrupt requires the NVIC to be programmed first. The EEFC interrupt is generated only on FRDY bit rising.

Table 22-1. Peripheral IDs

Instance	ID
EFC	6

## 22.4 Functional Description

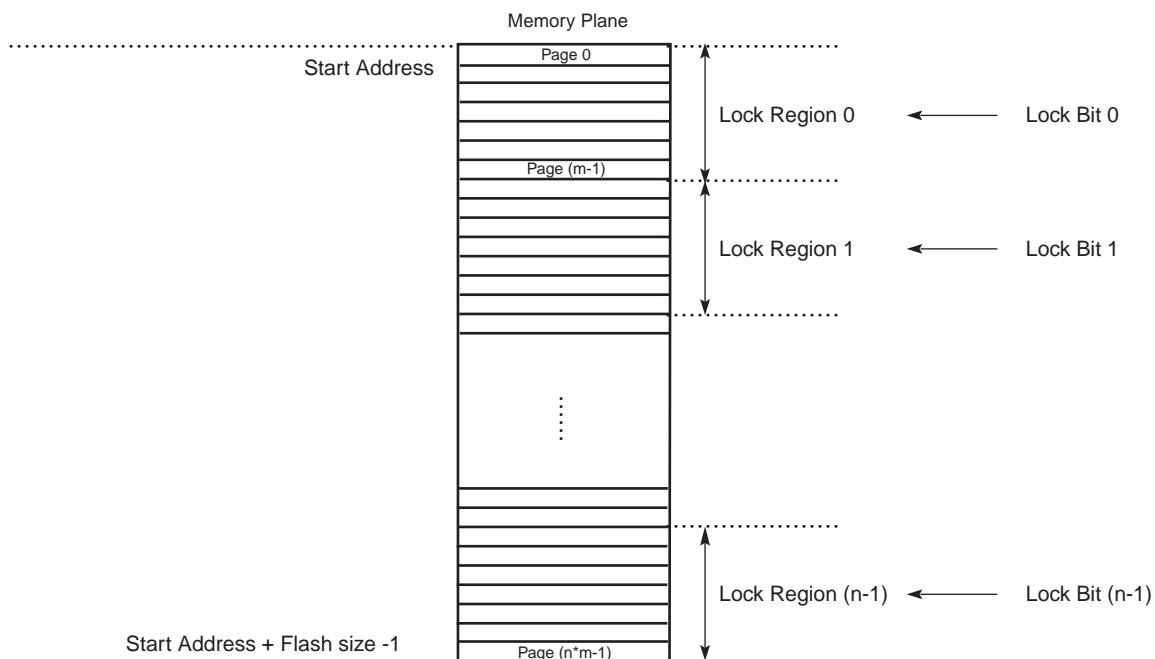
### 22.4.1 Embedded Flash Organization

The embedded Flash interfaces directly with the 32-bit internal bus. The embedded Flash is composed of:

- One memory plane organized in several pages of the same size.
- Two 128-bit or 64-bit read buffers used for code read optimization.
- One 128-bit or 64-bit read buffer used for data read optimization.
- One write buffer that manages page programming. The write buffer size is equal to the page size. This buffer is write-only and accessible all along the 1 MByte address space, so that each word can be written to its final address.
- Several lock bits used to protect write/erase operation on several pages (lock region). A lock bit is associated with a lock region composed of several pages in the memory plane.
- Several bits that may be set and cleared through the Enhanced Embedded Flash Controller (EEFC) interface, called General Purpose Non Volatile Memory bits (GPNVM bits).

The embedded Flash size, the page size, the lock regions organization and GPNVM bits definition are specific to the product. The Enhanced Embedded Flash Controller (EEFC) returns a descriptor of the Flash controlled after a get descriptor command issued by the application (see ["Getting Embedded Flash Descriptor" on page 363](#)).

**Figure 22-1. Embedded Flash Organization**



## 22.4.2 Read Operations

An optimized controller manages embedded Flash reads, thus increasing performance when the processor is running in Thumb2 mode by means of the 128- or 64-bit wide memory interface.

The Flash memory is accessible through 8-, 16- and 32-bit reads.

As the Flash block size is smaller than the address space reserved for the internal memory area, the embedded Flash wraps around the address space and appears to be repeated within it.

The read operations can be performed with or without wait states. Wait states must be programmed in the field FWS (Flash Read Wait State) in the Flash Mode Register (EEFC\_FMR). Defining FWS to be 0 enables the single-cycle access of the embedded Flash. Refer to the Electrical Characteristics for more details.

### 22.4.2.1 128-bit or 64-bit Access Mode

By default the read accesses of the Flash are performed through a 128-bit wide memory interface. It enables better system performance especially when 2 or 3 wait state needed.

For systems requiring only 1 wait state, or to privilege current consumption rather than performance, the user can select a 64-bit wide memory access via the FAM bit in the Flash Mode Register (EEFC\_FMR)

Please refer to the electrical characteristics section of the product datasheet for more details.

### 22.4.2.2 Code Read Optimization

This feature is enabled if the EEFC\_FMR register bit SCOD is cleared.

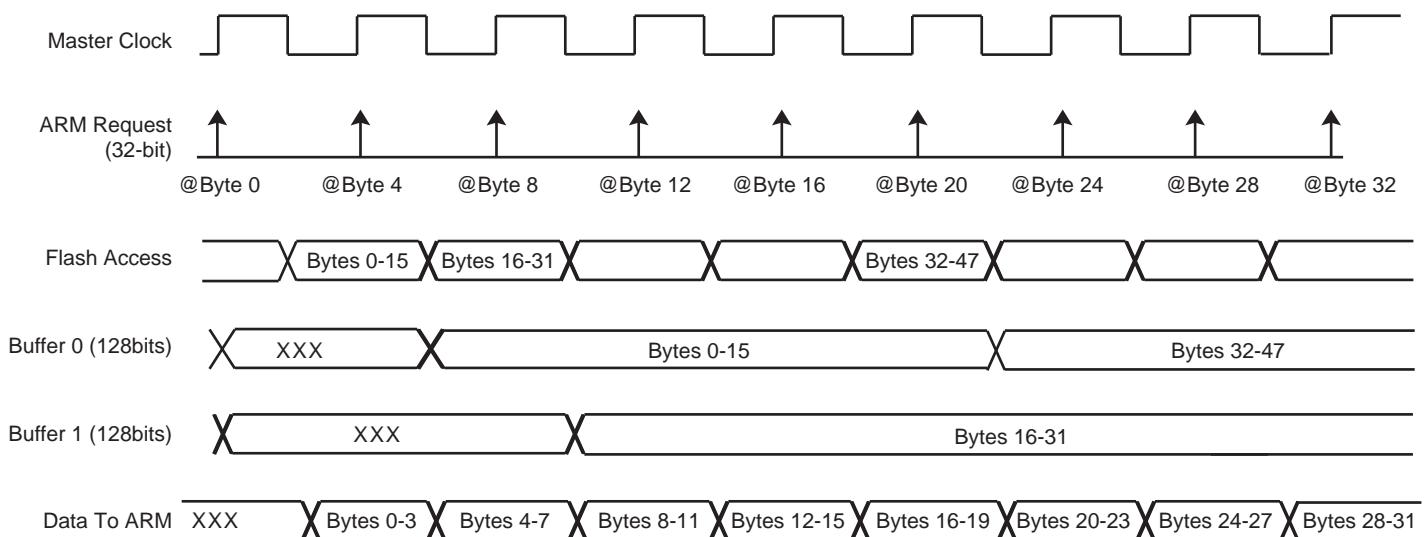
A system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize sequential Code Fetch.

Note: Immediate consecutive code read accesses are not mandatory to benefit from this optimization.

The sequential code read optimization is enabled by default. If the bit SCOD in Flash Mode Register (EEFC\_FMR) is set to 1, these buffers are disabled and the sequential code read is not optimized anymore.

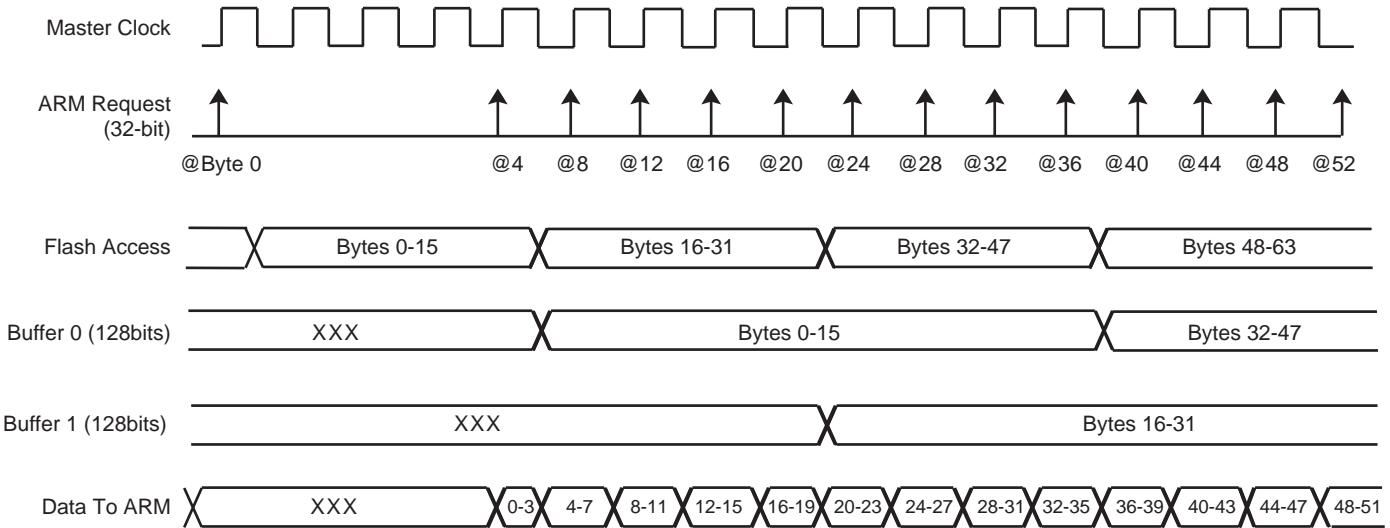
Another system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize loop code fetch (see “[Code Loops Optimization](#)” on page 360).

**Figure 22-2. Code Read Optimization for FWS = 0**



Note: When FWS is equal to 0, all the accesses are performed in a single-cycle access.

**Figure 22-3. Code Read Optimization for FWS = 3**



Note: When FWS is included between 1 and 3, in case of sequential reads, the first access takes (FWS+1) cycles, the other ones only 1 cycle.

#### 22.4.2.3 Code Loops Optimization

The Code Loops optimization is enabled when the CLOE bit of the EEFC\_FMR register is set at 1.

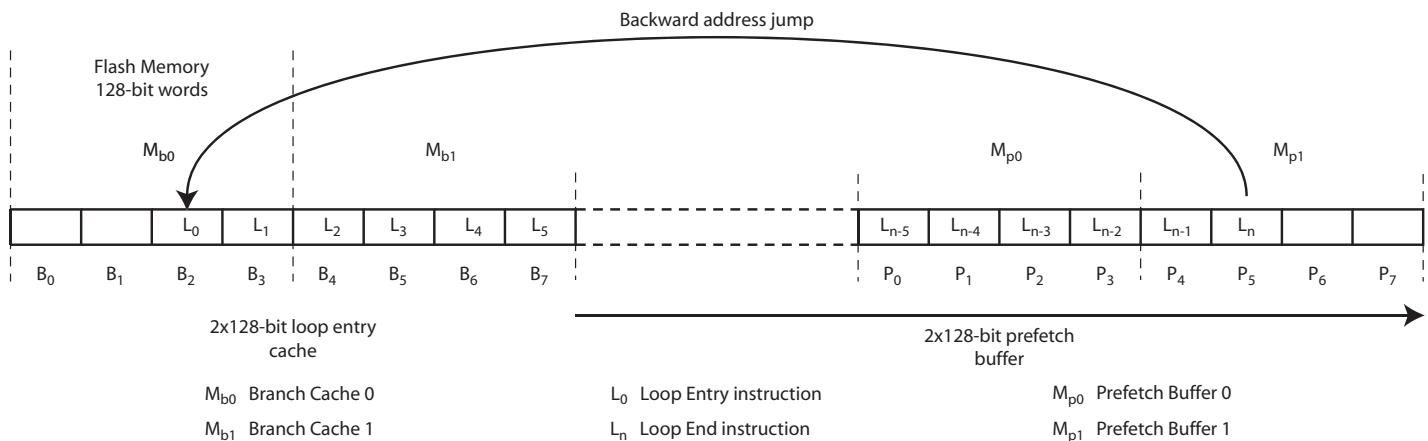
When a backward jump is inserted in the code, the pipeline of the sequential optimization is broken, and it becomes inefficient. In this case the loop code read optimization takes over from the sequential code read optimization to avoid insertion of wait states. The loop code read optimization is enabled by default. If in Flash Mode Register (EEFC\_FMR), the bit CLOE is reset to 0 or the bit SCOD is set to 1, these buffers are disabled and the loop code read is not optimized anymore.

When this feature is enabled, if inner loop body instructions  $L_0$  to  $L_n$  lay from the 128-bit flash memory cell  $M_{b0}$  to the memory cell  $M_{p1}$ , after recognition of a first backward branch, the two first flash memory cells  $M_{b0}$  and  $M_{b1}$  targeted by this branch are cached for fast access from the processor at the next loop iterations.

Afterwards, combining the sequential prefetch (described in [Section 22.4.2.2 "Code Read Optimization"](#)) through the loop body with the fast read access to the loop entry cache, the whole loop can be iterated with no wait-state.

[Figure 22-4](#) below illustrates the Code Loops optimization.

**Figure 22-4. Code Loops Optimization**

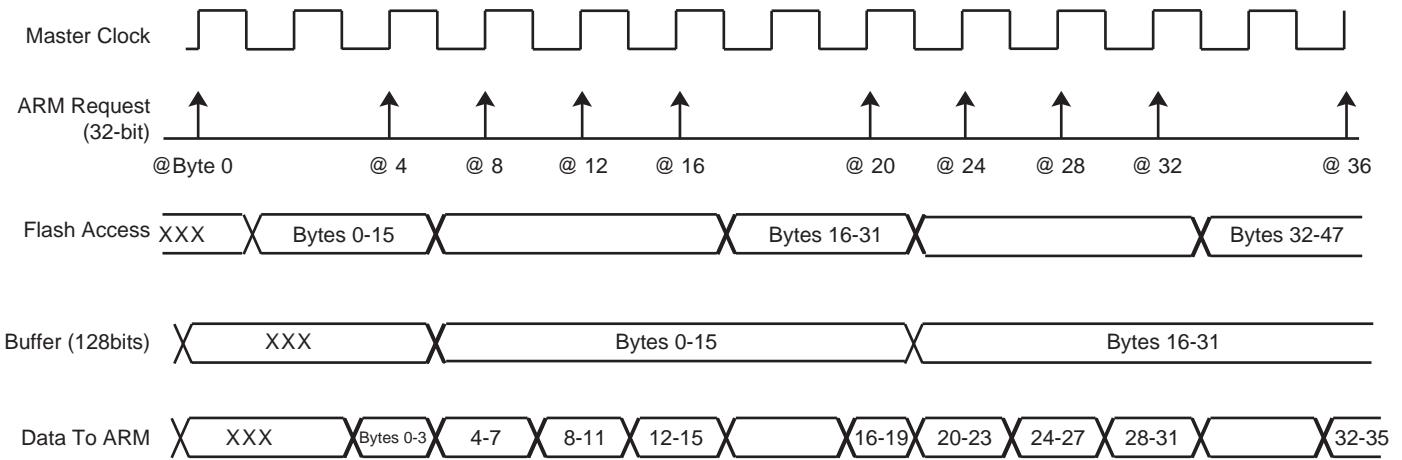


#### 22.4.2.4 Data Read Optimization

The organization of the Flash in 128 bits (or 64 bits) is associated with two 128-bit (or 64-bit) prefetch buffers and one 128-bit (or 64-bit) data read buffer, thus providing maximum system performance. This buffer is added in order to store the requested data plus all the data contained in the 128-bit (64-bit) aligned data. This speeds up sequential data reads if, for example, FWS is equal to 1 (see [Figure 22-5](#)). The data read optimization is enabled by default. If the bit SCOD in Flash Mode Register (EEFC\_FMR) is set to 1, this buffer is disabled and the data read is not optimized anymore.

Note: No consecutive data read accesses are mandatory to benefit from this optimization.

**Figure 22-5. Data Read Optimization for FWS = 1**



#### 22.4.3 Flash Commands

The Enhanced Embedded Flash Controller (EEFC) offers a set of commands such as programming the memory Flash, locking and unlocking lock regions, consecutive programming and locking and full Flash erasing, etc.

**Table 22-2. Set of Commands**

Command	Value	Mnemonic
Get Flash Descriptor	0x00	GETD
Write page	0x01	WP
Write page and lock	0x02	WPL
Erase page and write page	0x03	EWP
Erase page and write page then lock	0x04	EWPL
Erase all	0x05	EA
Erase Pages	0x07	EPA
Set Lock Bit	0x08	SLB
Clear Lock Bit	0x09	CLB
Get Lock Bit	0x0A	GLB
Set GPNVM Bit	0x0B	SGPB
Clear GPNVM Bit	0x0C	CGPB
Get GPNVM Bit	0x0D	GGPB
Start Read Unique Identifier	0x0E	STUI

**Table 22-2. Set of Commands**

Command	Value	Mnemonic
Stop Read Unique Identifier	0x0F	SPUI
Get CALIB Bit	0x10	GCALB
Erase Sector	0x11	ES
Write User Signature	0x12	WUS
Erase User Signature	0x13	EUS
Start Read User Signature	0x14	STUS
Stop Read User Signature	0x15	SPUS

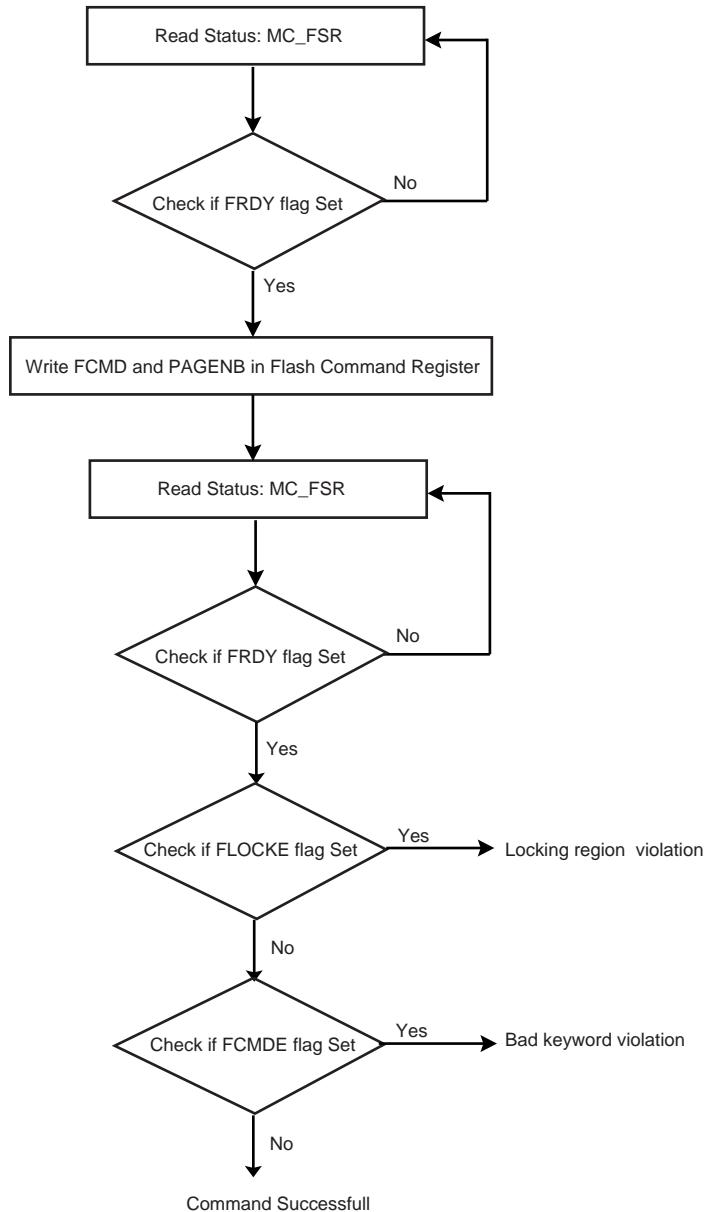
In order to perform one of these commands, the Flash Command Register (EEFC\_FCR) has to be written with the correct command using the FCMD field. As soon as the EEFC\_FCR register is written, the FRDY flag and the FVALUE field in the EEFC\_FRR register are automatically cleared. Once the current command is achieved, then the FRDY flag is automatically set. If an interrupt has been enabled by setting the FRDY bit in EEFC\_FMR, the corresponding interrupt line of the NVIC is activated. (Note that this is true for all commands except for the STUI Command. The FRDY flag is not set when the STUI command is achieved.)

All the commands are protected by the same keyword, which has to be written in the 8 highest bits of the EEFC\_FCR register.

Writing EEFC\_FCR with data that does not contain the correct key and/or with an invalid command has no effect on the whole memory plane, but the FCMDE flag is set in the EEFC\_FSR register. This flag is automatically cleared by a read access to the EEFC\_FSR register.

When the current command writes or erases a page in a locked region, the command has no effect on the whole memory plane, but the FLOCKE flag is set in the EEFC\_FSR register. This flag is automatically cleared by a read access to the EEFC\_FSR register.

**Figure 22-6. Command State Chart**



#### 22.4.3.1 Getting Embedded Flash Descriptor

This command allows the system to learn about the Flash organization. The system can take full advantage of this information. For instance, a device could be replaced by one with more Flash capacity, and so the software is able to adapt itself to the new configuration.

To get the embedded Flash descriptor, the application writes the GETD command in the EEFC\_FCR register. The first word of the descriptor can be read by the software application in the EEFC\_FRR register as soon as the FRDY flag in the EEFC\_FSR register rises. The next reads of the EEFC\_FRR register provide the following word of the descriptor. If extra

read operations to the EEFC\_FRR register are done after the last word of the descriptor has been returned, then the EEFC\_FRR register value is 0 until the next valid command.

**Table 22-3. Flash Descriptor Definition**

Symbol	Word Index	Description
FL_ID	0	Flash Interface Description
FL_SIZE	1	Flash size in bytes
FL_PAGE_SIZE	2	Page size in bytes
FL_NB_PLANE	3	Number of planes. (only for SAM3SD8).
FL_PLANE[0]	4	Number of bytes in the first plane.
...		
FL_PLANE[FL_NB_PLANE-1]	4 + FL_NB_PLANE - 1	Number of bytes in the last plane.
FL_NB_LOCK	4 + FL_NB_PLANE	Number of lock bits. A bit is associated with a lock region. A lock bit is used to prevent write or erase operations in the lock region.
FL_LOCK[0]	4 + FL_NB_PLANE + 1	Number of bytes in the first lock region.
...		

#### 22.4.3.2 Write Commands

Several commands can be used to program the Flash.

Flash technology requires that an erase be done before programming. The full memory plane can be erased at the same time, or several pages can be erased at the same time (refer to [Figure 22-7, "Example of Partial Page Programming"](#), and the paragraph below the figure.). Also, a page erase can be automatically done before a page write using EWP or EWPL commands.

After programming, the page (the whole lock region) can be locked to prevent miscellaneous write or erase sequences. The lock bit can be automatically set after page programming using WPL or EWPL commands.

Data to be written are stored in an internal latch buffer. The size of the latch buffer corresponds to the page size. The latch buffer wraps around within the internal memory area address space and is repeated as many times as the number of pages within this address space.

Note: Writing of 8-bit and 16-bit data is not allowed and may lead to unpredictable data corruption.

Write operations are performed in a number of wait states equal to the number of wait states for read operations.

Data are written to the latch buffer before the programming command is written to the Flash Command Register EEFC\_FCR. The sequence is as follows:

- Write the full page, at any page address, within the internal memory area address space.
- Programming starts as soon as the page number and the programming command are written to the Flash Command Register. The FRDY bit in the Flash Programming Status Register (EEFC\_FSR) is automatically cleared.
- When programming is completed, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt has been enabled by setting the bit FRDY in EEFC\_FMR, the corresponding interrupt line of the NVIC is activated.

Two errors can be detected in the EEFC\_FSR register after a programming sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Lock Error: the page to be programmed belongs to a locked region. A command must be previously run to unlock the corresponding region.
- Flash Error: at the end of the programming, the WriteVerify test of the Flash memory has failed.

By using the WP command, a page can be programmed in several steps if it has been erased before (see [Figure 22-7](#) below). This mode is called Partial Programming. Optimization of the programming period is automatically performed by the Flash Controller (see [Section 22.4.3.3 “Optimized Partial Programming” on page 365](#)).

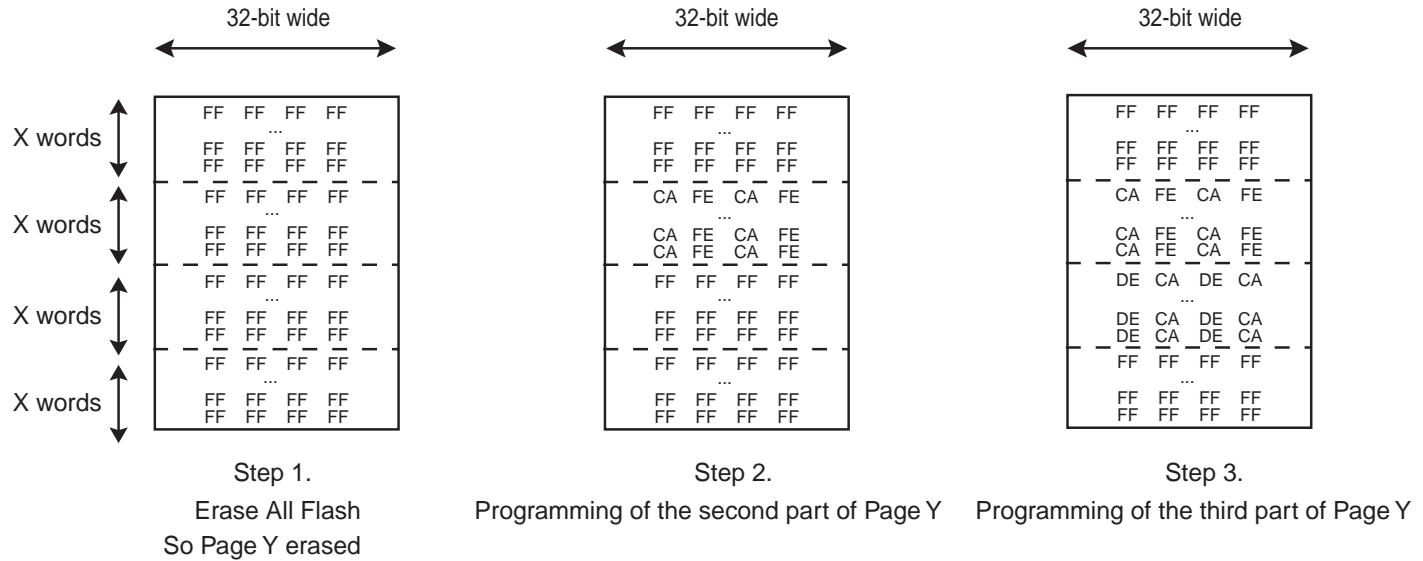
The Partial Programming mode works only with 32-bit (or higher) boundaries. It cannot be used with boundaries lower than 32 bits (8 or 16-bit for example). To write a single byte or a 16-bit halfword, the remaining byte of the 32-bit word must be filled with 0xFF, then the 32-bit word must be written to Flash buffer.

**Note:** If several 32-bit words need to be programmed, they must be written in ascending order to Flash buffer before executing the write page command. If a write page command is executed after writing each single 32-bit word, the write order of the word sequence does not matter.

After any power-on sequence, the Flash memory internal latch buffer is not initialized. Thus the latch buffer must be initialized by writing the part-select to be programmed with user data and the remaining of the buffer must be written with logical 1.

This action is not required for the next partial programming sequence because the latch buffer is automatically cleared after programming the page.

**Figure 22-7. Example of Partial Page Programming**



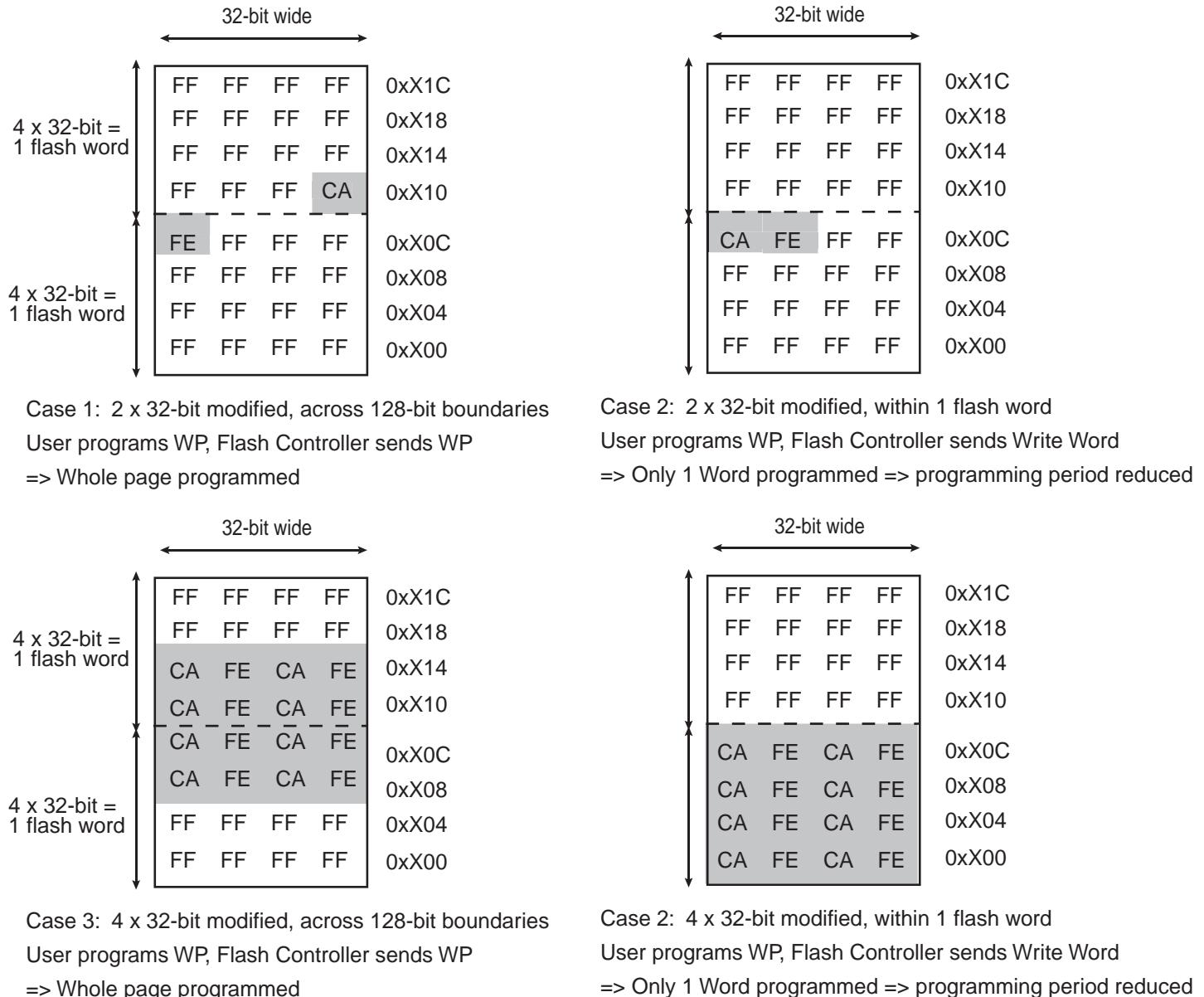
#### 22.4.3.3 Optimized Partial Programming

Depending on the flash buffer write access conditions between 2 commands, the controller may automatically optimize the programming period by passing a “write-word” command or a “write-page” command to the Flash memory array.

In any case the write-page command (WP) must be programmed in the EEFC\_FCR (see [Section 22.5.2 “EEFC Flash Command Register” on page 373](#)).

The selection of the command passed to the Flash memory array is defined by the Flash controller according to the location of newly programmed 32-bit words within a 128-bit Flash word (See [Figure 22-8 on page 366](#)).

**Figure 22-8. Optimized Partial Programming**



#### 22.4.3.4 Erase Commands

Erase commands are allowed only on unlocked regions. Depending on the Flash memory, several commands can be used to erase the Flash:

- Erase all memory (EA): all memory is erased. The processor must not fetch code from the Flash memory.
- Erase pages (EPA): 4, 8, 16 or 32 pages are erased in the memory plane. The first page to be erased is specified in the FARG[15:2] field of the MC\_FCR register. The first page number must be modulo 4, 8, 16 or 32 according to the number of pages to erase at the same time. The processor must not fetch code from the Flash memory.
- Erase Sector (ES): A full memory sector is erased. Sector size depends on the Flash memory. FARG must be set with a page number that is in the sector to be erased. The processor must not fetch code from the Flash memory.

The erase sequence is:

- Erase starts as soon as one of the erase commands and the FARG field are written in the Flash Command Register.

- For the EPA command, the 2 lowest bits of the FARG field define the number of pages to be erased (FARG[1:0]):

**Table 22-4. FARG Field for EPA command:**

FARG[1:0]	Number of pages to be erased with EPA command
0	4 pages
1	8 pages
2	16 pages
3	32 pages

- When the programming completes, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC\_FMR, the interrupt line of the NVIC is activated.

Two errors can be detected in the EEFC\_FSR register after a programming sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Lock Error: at least one page to be erased belongs to a locked region. The erase command has been refused, no page has been erased. A command must be run previously to unlock the corresponding region.
- Flash Error: at the end of the programming, the EraseVerify test of the Flash memory has failed.

#### 22.4.3.5 Lock Bit Protection

Lock bits are associated with several pages in the embedded Flash memory plane. This defines lock regions in the embedded Flash memory plane. They prevent writing/erasing protected pages.

The lock sequence is:

- The Set Lock command (SLB) and a page number to be protected are written in the Flash Command Register.
- When the locking completes, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC\_FMR, the interrupt line of the NVIC is activated.
- The result of the SLB command can be checked running a GLB (Get Lock Bit) command.

Note: The value of the FARG argument passed together with SLB command must not exceed the higher lock bit index available in the product.

One error can be detected in the EEFC\_FSR register after a programming sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Flash Error: at the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

It is possible to clear lock bits previously set. Then the locked region can be erased or programmed. The unlock sequence is:

- The Clear Lock command (CLB) and a page number to be unprotected are written in the Flash Command Register.
- When the unlock completes, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC\_FMR, the interrupt line of the NVIC is activated.

Note: The value of the FARG argument passed together with CLB command must not exceed the higher lock bit index available in the product.

One error can be detected in the EEFC\_FSR register after a programming sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Flash Error: at the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

The status of lock bits can be returned by the Enhanced Embedded Flash Controller (EEFC). The Get Lock Bit status sequence is:

- The Get Lock Bit command (GLB) is written in the Flash Command Register, FARG field is meaningless.

- Lock bits can be read by the software application in the EEFC\_FRR register. The first word read corresponds to the 32 first lock bits, next reads providing the next 32 lock bits as long as it is meaningful. Extra reads to the EEFC\_FRR register return 0.

For example, if the third bit of the first word read in the EEFC\_FRR is set, then the third lock region is locked.

One error can be detected in the EEFC\_FSR register after a programming sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Flash Error: at the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

Note: Access to the Flash in read is permitted when a set, clear or get lock bit command is performed.

#### 22.4.3.6 GPNVM Bit

GPNVM bits do not interfere with the embedded Flash memory plane. Refer to specific product details for information on GPNVM bit action.

The set GPNVM bit sequence is:

- Start the Set GPNVM Bit command (SGPB) by writing the Flash Command Register with the SGPB command and the number of the GPNVM bit to be set.
- When the GPNVM bit is set, the bit FRDY in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt was enabled by setting the FRDY bit in EEFC\_FMR, the interrupt line of the NVIC is activated.
- The result of the SGPB command can be checked by running a GGPB (Get GPNVM Bit) command.

Note: The value of the FARG argument passed together with SGPB command must not exceed the higher GPNVM index available in the product. Flash Data Content is not altered if FARG exceeds the limit. Command Error is detected only if FARG is greater than 8.

One error can be detected in the EEFC\_FSR register after a programming sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Flash Error: at the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

It is possible to clear GPNVM bits previously set. The clear GPNVM bit sequence is:

- Start the Clear GPNVM Bit command (CGPB) by writing the Flash Command Register with CGPB and the number of the GPNVM bit to be cleared.
- When the clear completes, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC\_FMR, the interrupt line of the NVIC is activated.

Note: The value of the FARG argument passed together with CGPB command must not exceed the higher GPNVM index available in the product. Flash Data Content is not altered if FARG exceeds the limit. Command Error is detected only if FARG is greater than 8.

One error can be detected in the EEFC\_FSR register after a programming sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Flash Error: at the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

The status of GPNVM bits can be returned by the Enhanced Embedded Flash Controller (EEFC). The sequence is:

- Start the Get GPNVM bit command by writing the Flash Command Register with GGPB. The FARG field is meaningless.
- GPNVM bits can be read by the software application in the EEFC\_FRR register. The first word read corresponds to the 32 first GPNVM bits, following reads provide the next 32 GPNVM bits as long as it is meaningful. Extra reads to the EEFC\_FRR register return 0.

For example, if the third bit of the first word read in the EEFC\_FRR is set, then the third GPNVM bit is active.

One error can be detected in the EEFC\_FSR register after a programming sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.

Note: Access to the Flash in read is permitted when a set, clear or get GPNVM bit command is performed.

#### 22.4.3.7 Calibration Bit

Calibration bits do not interfere with the embedded Flash memory plane.

It is impossible to modify the calibration bits.

The status of calibration bits can be returned by the Enhanced Embedded Flash Controller (EEFC). The sequence is:

- Issue the Get CALIB Bit command by writing the Flash Command Register with GCALB (see [Table 22-2](#)). The FARG field is meaningless.
- Calibration bits can be read by the software application in the EEFC\_FRR register. The first word read corresponds to the 32 first calibration bits, following reads provide the next 32 calibration bits as long as it is meaningful. Extra reads to the EEFC\_FRR register return 0.

The 4/8/12 MHz Fast RC oscillator is calibrated in production. This calibration can be read through the Get CALIB Bit command. The table below shows the bit implementation for each frequency:

**Table 22-5. Calibration Bit Indexes**

RC Calibration Frequency	EEFC_FRR Bits
8 MHz output	[28 - 22]
12 MHz output	[38 - 32]

The RC calibration for the 4 MHz is set to '1000000'.

#### 22.4.3.8 Security Bit Protection

When the security is enabled, access to the Flash, either through the JTAG/SWD interface or through the Fast Flash Programming Interface, is forbidden. This ensures the confidentiality of the code programmed in the Flash.

The security bit is GPNVM0.

Disabling the security bit can only be achieved by asserting the ERASE pin at 1, and after a full Flash erase is performed. When the security bit is deactivated, all accesses to the Flash are permitted.

#### 22.4.3.9 Unique Identifier

Each part is programmed with a 128-bit Unique Identifier. It can be used to generate keys for example. To read the Unique Identifier the sequence is:

- Send the Start Read unique Identifier command (STUI) by writing the Flash Command Register with the STUI command.
- When the Unique Identifier is ready to be read, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) falls.
- The Unique Identifier is located in the first 128 bits of the Flash memory mapping, thus, at the address 0x00400000-0x004003FF.
- To stop the Unique Identifier mode, the user needs to send the Stop Read unique Identifier command (SPUI) by writing the Flash Command Register with the SPUI command.
- When the Stop read Unique Identifier command (SPUI) has been performed, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt was enabled by setting the FRDY bit in EEFC\_FMR, the interrupt line of the NVIC is activated.

Note that during the sequence, the software can not run out of Flash (or the second plane in case of dual plane).

#### 22.4.3.10 User Signature

Each part contains a User Signature of 512-bytes. It can be used by the user for storage. Read, write and erase of this area is allowed.

To read the User Signature, the sequence is as follows:

- Send the Start Read User Signature command (STUS) by writing the Flash Command Register with the STUS command.
- When the User Signature is ready to be read, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) falls.
- The User Signature is located in the first 512 bytes of the Flash memory mapping, thus, at the address 0x00400000-0x004001FF.
- To stop the User Signature mode, the user needs to send the Stop Read User Signature command (SPUS) by writing the Flash Command Register with the SPUS command.
- When the Stop Read User Signature command (SPUI) has been performed, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt was enabled by setting the FRDY bit in EEFC\_FMR, the interrupt line of the NVIC is activated.

Note that during the sequence, the software can not run out of Flash (or the second plane, in case of dual plane).

One error can be detected in the EEFC\_FSR register after this sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.

To write the User Signature, the sequence is:

- Write the full page, at any page address, within the internal memory area address space.
- Send the Write User Signature command (WUS) by writing the Flash Command Register with the WUS command.
- When programming is completed, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC\_FMR, the corresponding interrupt line of the NVIC is activated.

Two errors can be detected in the EEFC\_FSR register after this sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Flash Error: at the end of the programming, the WriteVerify test of the Flash memory has failed.

To erase the User Signature, the sequence is:

- Send the Erase User Signature command (EUS) by writing the Flash Command Register with the EUS command.
- When programming is completed, the FRDY bit in the Flash Programming Status Register (EEFC\_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC\_FMR, the corresponding interrupt line of the NVIC is activated.

Two errors can be detected in the EEFC\_FSR register after this sequence:

- Command Error: a bad keyword has been written in the EEFC\_FCR register.
- Flash Error: at the end of the programming, the EraseVerify test of the Flash memory has failed.

## 22.5 Enhanced Embedded Flash Controller (EEFC) User Interface

The User Interface of the Enhanced Embedded Flash Controller (EEFC) is integrated within the System Controller with base address 0x400E0A00.

**Table 22-6. Register Mapping**

Offset	Register	Name	Access	Reset State
0x00	EEFC Flash Mode Register	EEFC_FMR	Read-write	0x0400_0000
0x04	EEFC Flash Command Register	EEFC_FCR	Write-only	–
0x08	EEFC Flash Status Register	EEFC_FSR	Read-only	0x00000001
0x0C	EEFC Flash Result Register	EEFC_FRR	Read-only	0x0
0x10	Reserved	–	–	–

### 22.5.1 EEFC Flash Mode Register

**Name:** EEFC\_FMR

**Address:** 0x400E0A00

**Access:** Read-write

**Offset:** 0x00

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

- **FRDY: Ready Interrupt Enable**

0: Flash Ready does not generate an interrupt.

1: Flash Ready (to accept a new command) generates an interrupt.

- **FWS: Flash Wait State**

This field defines the number of wait states for read and write operations:

$$\text{Number of cycles for Read/Write operations} = \text{FWS} + 1$$

- **SCOD: Sequential Code Optimization Disable**

0: The sequential code optimization is enabled.

1: The sequential code optimization is disabled.

No Flash read should be done during change of this register.

- **FAM: Flash Access Mode**

0: 128-bit access in read Mode only, to enhance access speed.

1: 64-bit access in read Mode only, to enhance power consumption.

No Flash read should be done during change of this register.

- **CLOE: Code Loops Optimization Enable**

0: The opcode loops optimization is disabled.

1: The opcode loops optimization is enabled.

No Flash read should be done during change of this register.

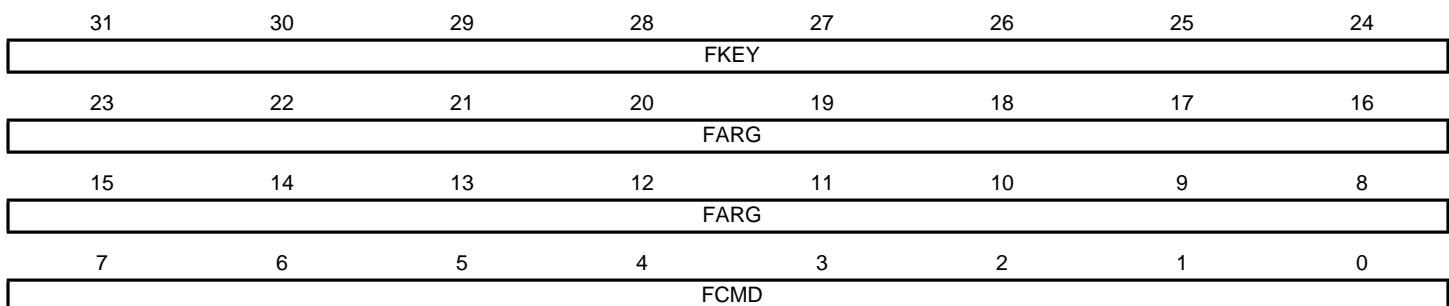
## 22.5.2 EEFC Flash Command Register

**Name:** EEFC\_FCR

**Address:** 0x400E0A04

**Access:** Write-only

**Offset:** 0x04



- **FCMD: Flash Command**

Value	Name	Description
0x00	GETD	Get Flash Descriptor
0x01	WP	Write page
0x02	WPL	Write page and lock
0x03	EWP	Erase page and write page
0x04	EWPL	Erase page and write page then lock
0x05	EA	Erase all
0x07	EPA	Erase Pages
0x08	SLB	Set Lock Bit
0x09	CLB	Clear Lock Bit
0x0A	GLB	Get Lock Bit
0x0B	SGPB	Set GPNVM Bit
0x0C	CGPB	Clear GPNVM Bit
0x0D	GGPB	Get GPNVM Bit
0x0E	STUI	Start Read Unique Identifier
0x0F	SPUI	Stop Read Unique Identifier
0x10	GCALB	Get CALIB Bit
0x11	ES	Erase Sector
0x12	WUS	Write User Signature
0x13	EUS	Erase User Signature
0x14	STUS	Start Read User Signature
0x15	SPUS	Stop Read User Signature

- **FARG: Flash Command Argument**

Erase all command	Field is meaningless.
Erase sector command	FARG must be set with a page number that is in the sector to be erased.
Erase pages command	<p>FARG[1:0] defines the number of pages to be erased.  The page number from which the erase will start is defined as follows:  FARG[1:0]=0, start page = 4*FARG[15:2]  FARG[1:0]=1, start page = 8*FARG[15:3], FARG[2] undefined  FARG[1:0]=2, start page = 16*FARG[15:4], FARG[3:2] undefined  FARG[1:0]=3, start page = 32*FARG[15:5], FARG[4:2] undefined  Note: undefined bit must be written to 0.  Refer to <a href="#">Table 22-4 on page 367</a></p>
Programming command	FARG defines the page number to be programmed.
Lock command	FARG defines the page number to be locked.
GPNVM command	FARG defines the GPNVM number.

- **FKEY: Flash Writing Protection Key**

Value	Name	Description
0x5A	PASSWD	The 0x5A value enables the command defined by the bits of the register. If the field is written with a different value, the write is not performed and no action is started.

### 22.5.3 EEFC Flash Status Register

**Name:** EEFC\_FSR

**Address:** 0x400E0A08

**Access:** Read-only

**Offset:** 0x08

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
-	-	-	-	FLERR	FLOCKE	FCMDE	FRDY

- **FRDY: Flash Ready Status**

0: The Enhanced Embedded Flash Controller (EEFC) is busy.

1: The Enhanced Embedded Flash Controller (EEFC) is ready to start a new command.

When it is set, this flag triggers an interrupt if the FRDY flag is set in the EEFC\_FMR register.

This flag is automatically cleared when the Enhanced Embedded Flash Controller (EEFC) is busy.

- **FCMDE: Flash Command Error Status**

0: No invalid commands and no bad keywords were written in the Flash Mode Register EEFC\_FMR.

1: An invalid command and/or a bad keyword was/were written in the Flash Mode Register EEFC\_FMR.

This flag is automatically cleared when EEFC\_FSR is read or EEFC\_FCR is written.

- **FLOCKE: Flash Lock Error Status**

0: No programming/erase of at least one locked region has happened since the last read of EEFC\_FSR.

1: Programming/erase of at least one locked region has happened since the last read of EEFC\_FSR.

This flag is automatically cleared when EEFC\_FSR is read or EEFC\_FCR is written.

- **FLERR: Flash Error Status**

0: No Flash Memory error occurred at the end of programming (EraseVerify or WriteVerify test has passed).

1: A Flash Memory error occurred at the end of programming (EraseVerify or WriteVerify test has failed).

#### 22.5.4 EEFC Flash Result Register

**Name:** EEFC\_FRR

**Address:** 0x400E0A0C

**Access:** Read-only

**Offset:** 0x0C

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

- **FVALUE: Flash Result Value**

The result of a Flash command is returned in this register. If the size of the result is greater than 32 bits, then the next resulting value is accessible at the next register read.

## 23. Fast Flash Programming Interface (FFPI)

### 23.1 Description

The Fast Flash Programming Interface provides parallel high-volume programming using a standard gang programmer. The parallel interface is fully handshaked and the device is considered to be a standard EEPROM. Additionally, the parallel protocol offers an optimized access to all the embedded Flash functionalities.

Although the Fast Flash Programming Mode is a dedicated mode for high volume programming, this mode is not designed for in-situ programming.

### 23.2 Embedded Characteristics

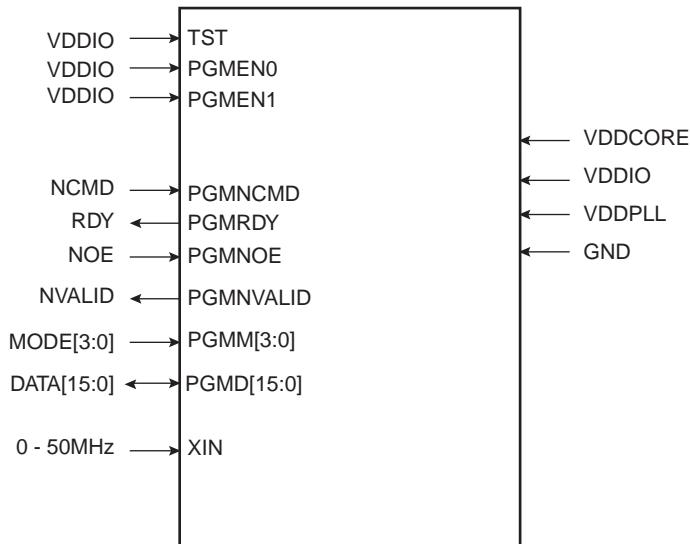
- Programming Mode for High-volume Flash Programming Using Gang Programmer
  - Offers Read and Write Access to the Flash Memory Plane
  - Enables Control of Lock Bits and General-purpose NVM Bits
  - Enables Security Bit Activation
  - Disabled once Security Bit is Set
- Offers Two Interfaces
  - Parallel Fast Flash Programming Interface
    - Provides a 16-bit Parallel Interface to Program the Embedded Flash
    - Full Handshake Protocol

## 23.3 Parallel Fast Flash Programming

### 23.3.1 Device Configuration

In Fast Flash Programming Mode, the device is in a specific test mode. Only a certain set of pins is significant. The rest of the PIOs are used as inputs with a pull-up. The crystal oscillator is in bypass mode. Other pins must be left unconnected.

**Figure 23-1. SAM4ExE Parallel Programming Interface**



**Table 23-1. Signal Description List**

Signal Name	Function	Type	Active Level	Comments
<b>Power</b>				
VDDIO	I/O Lines Power Supply	Power		
VDDCORE	Core Power Supply	Power		
VDDPLL	PLL Power Supply	Power		
GND	Ground	Ground		
<b>Clocks</b>				
XIN	Main Clock Input. This input can be tied to GND. In this case, the device is clocked by the internal RC oscillator.	Input		32 kHz to 50 MHz
<b>Test</b>				
TST	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN0	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN1	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN2	Test Mode Select	Input	Low	Must be connected to GND
<b>PIO</b>				
PGMNCMD	Valid command available	Input	Low	Pulled-up input at reset
PGMRDY	0: Device is busy 1: Device is ready for a new command	Output	High	Pulled-up input at reset
PGMNOE	Output Enable (active high)	Input	Low	Pulled-up input at reset
PGMNVALID	0: DATA[15:0] is in input mode 1: DATA[15:0] is in output mode	Output	Low	Pulled-up input at reset
PGMM[3:0]	Specifies DATA type (see <a href="#">Table 23-2</a> )	Input		Pulled-up input at reset
PGMD[15:0]	Bi-directional data bus	Input/Output		Pulled-up input at reset

### 23.3.2 Signal Names

Depending on the MODE settings, DATA is latched in different internal registers.

**Table 23-2. Mode Coding**

MODE[3:0]	Symbol	Data
0000	CMDE	Command Register
0001	ADDR0	Address Register LSBs
0010	ADDR1	
0101	DATA	Data Register
Default	IDLE	No register

When MODE is equal to CMDE, then a new command (strobed on DATA[15:0] signals) is stored in the command register.

**Table 23-3. Command Bit Coding**

DATA[15:0]	Symbol	Command Executed
0x0011	READ	Read Flash
0x0012	WP	Write Page Flash
0x0022	WPL	Write Page and Lock Flash
0x0032	EWP	Erase Page and Write Page
0x0042	EWPL	Erase Page and Write Page then Lock
0x0013	EA	Erase All
0x0014	SLB	Set Lock Bit
0x0024	CLB	Clear Lock Bit
0x0015	GLB	Get Lock Bit
0x0034	SGPB	Set General Purpose NVM bit
0x0044	CGPB	Clear General Purpose NVM bit
0x0025	GGPB	Get General Purpose NVM bit
0x0054	SSE	Set Security Bit
0x0035	GSE	Get Security Bit
0x001F	WRAM	Write Memory
0x001E	GVE	Get Version

### 23.3.3 Entering Programming Mode

The following algorithm puts the device in Parallel Programming Mode:

- Apply the supplies as described in [Table 23-1](#).
- Apply XIN clock within  $T_{POR\_RESET}$  if an external clock is available.
- Wait for  $T_{POR\_RESET}$
- Start a read or write handshaking.

Note: After reset, the device is clocked by the internal RC oscillator. Before clearing RDY signal, if an external clock ( $>32$  kHz) is connected to XIN, then the device switches on the external clock. Else, XIN input is not considered. A higher frequency on XIN speeds up the programmer handshake.

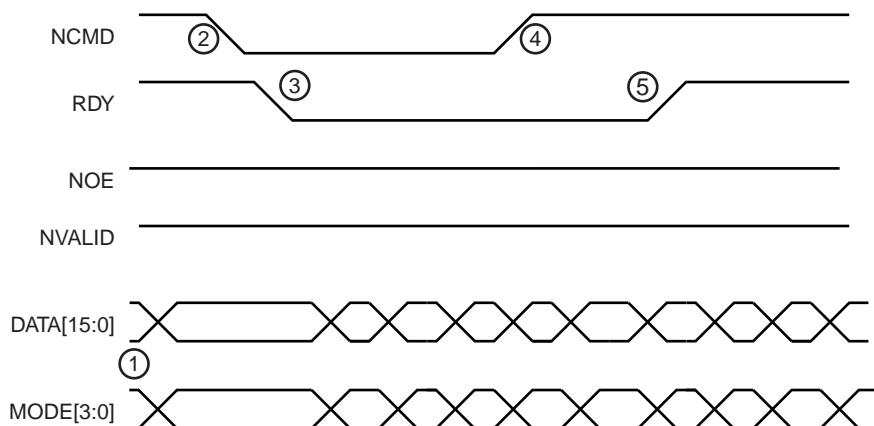
### 23.3.4 Programmer Handshaking

An handshake is defined for read and write operations. When the device is ready to start a new operation (RDY signal set), the programmer starts the handshake by clearing the NCMD signal. The handshaking is achieved once NCMD signal is high and RDY is high.

#### 23.3.4.1 Write Handshaking

For details on the write handshaking sequence, refer [Figure 23-2](#) and [Table 23-4](#).

**Figure 23-2. Parallel Programming Timing, Write Sequence**



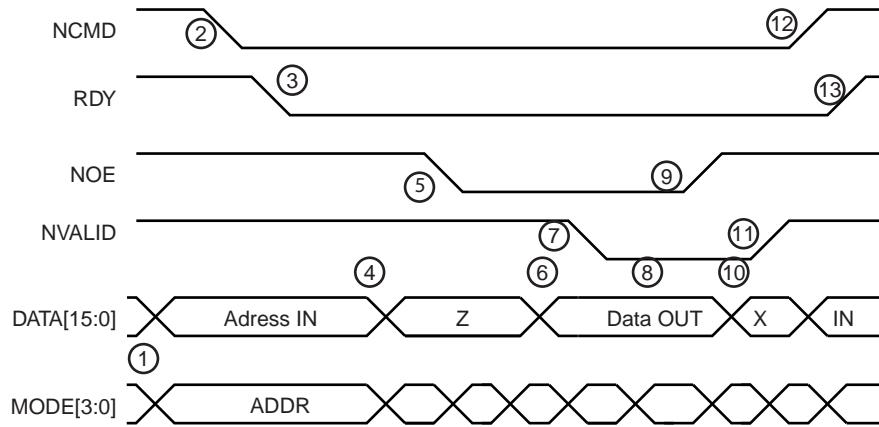
**Table 23-4. Write Handshake**

Step	Programmer Action	Device Action	Data I/O
1	Sets MODE and DATA signals	Waits for NCMD low	Input
2	Clears NCMD signal	Latches MODE and DATA	Input
3	Waits for RDY low	Clears RDY signal	Input
4	Releases MODE and DATA signals	Executes command and polls NCMD high	Input
5	Sets NCMD signal	Executes command and polls NCMD high	Input
6	Waits for RDY high	Sets RDY	Input

### 23.3.4.2 Read Handshaking

For details on the read handshaking sequence, refer to [Figure 23-3](#) and [Table 23-5](#).

**Figure 23-3. Parallel Programming Timing, Read Sequence**



**Table 23-5. Read Handshake**

Step	Programmer Action	Device Action	DATA I/O
1	Sets MODE and DATA signals	Waits for NCMD low	Input
2	Clears NCMD signal	Latch MODE and DATA	Input
3	Waits for RDY low	Clears RDY signal	Input
4	Sets DATA signal in tristate	Waits for NOE Low	Input
5	Clears NOE signal		Tristate
6	Waits for NVALID low	Sets DATA bus in output mode and outputs the flash contents.	Output
7		Clears NVALID signal	Output
8	Reads value on DATA Bus	Waits for NOE high	Output
9	Sets NOE signal		Output
10	Waits for NVALID high	Sets DATA bus in input mode	X
11	Sets DATA in output mode	Sets NVALID signal	Input
12	Sets NCMD signal	Waits for NCMD high	Input
13	Waits for RDY high	Sets RDY signal	Input

### 23.3.5 Device Operations

Several commands on the Flash memory are available. These commands are summarized in [Table 23-3 on page 380](#). Each command is driven by the programmer through the parallel interface running several read/write handshaking sequences.

When a new command is executed, the previous one is automatically achieved. Thus, chaining a read command after a write automatically flushes the load buffer in the Flash.

#### 23.3.5.1 Flash Read Command

This command is used to read the contents of the Flash memory. The read command can start at any valid address in the memory plane and is optimized for consecutive reads. Read handshaking can be chained; an internal address buffer is automatically increased.

**Table 23-6. Read Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	READ
2	Write handshaking	ADDR0	Memory Address LSB
3	Write handshaking	ADDR1	Memory Address
4	Read handshaking	DATA	*Memory Address++
5	Read handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	Memory Address LSB
n+1	Write handshaking	ADDR1	Memory Address
n+2	Read handshaking	DATA	*Memory Address++
n+3	Read handshaking	DATA	*Memory Address++
...	...	...	...

#### 23.3.5.2 Flash Write Command

This command is used to write the Flash contents.

The Flash memory plane is organized into several pages. Data to be written are stored in a load buffer that corresponds to a Flash memory page. The load buffer is automatically flushed to the Flash:

- Before access to any page other than the current one
- When a new command is validated (MODE = CMDE)

The **Write Page** command (**WP**) is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

**Table 23-8. Write Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	WP or WPL or EWP or EWPL
2	Write handshaking	ADDR0	Memory Address LSB
3	Write handshaking	ADDR1	Memory Address
4	Write handshaking	DATA	*Memory Address++
5	Write handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	Memory Address LSB

**Table 23-8. Write Command (Continued)**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
n+1	Write handshaking	ADDR1	Memory Address
n+2	Write handshaking	DATA	*Memory Address++
n+3	Write handshaking	DATA	*Memory Address++
...	...	...	...

The Flash command **Write Page and Lock (WPL)** is equivalent to the Flash Write Command. However, the lock bit is automatically set at the end of the Flash write operation. As a lock region is composed of several pages, the programmer writes to the first pages of the lock region using Flash write commands and writes to the last page of the lock region using a Flash write and lock command.

The Flash command **Erase Page and Write (EWP)** is equivalent to the Flash Write Command. However, before programming the load buffer, the page is erased.

The Flash command **Erase Page and Write the Lock (EWPL)** combines EWP and WPL commands.

### 23.3.5.3 Flash Full Erase Command

This command is used to erase the Flash memory planes.

All lock regions must be unlocked before the Full Erase command by using the CLB command. Otherwise, the erase command is aborted and no page is erased.

**Table 23-9. Full Erase Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	EA
2	Write handshaking	DATA	0

### 23.3.5.4 Flash Lock Commands

Lock bits can be set using WPL or EWPL commands. They can also be set by using the **Set Lock** command (**SLB**). With this command, several lock bits can be activated. A Bit Mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first lock bit is activated.

In the same way, the **Clear Lock** command (**CLB**) is used to clear lock bits.

**Table 23-10. Set and Clear Lock Bit Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	SLB or CLB
2	Write handshaking	DATA	Bit Mask

Lock bits can be read using **Get Lock Bit** command (**GLB**). The  $n^{\text{th}}$  lock bit is active when the bit  $n$  of the bit mask is set..

**Table 23-11. Get Lock Bit Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	GLB
2	Read handshaking	DATA	Lock Bit Mask Status 0 = Lock bit is cleared 1 = Lock bit is set

### 23.3.5.5 Flash General-purpose NVM Commands

General-purpose NVM bits (GP NVM bits) can be set using the **Set GPNVM** command (**SGPB**). This command also activates GP NVM bits. A bit mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first GP NVM bit is activated.

In the same way, the **Clear GPNVM** command (**CGPB**) is used to clear general-purpose NVM bits. The general-purpose NVM bit is deactivated when the corresponding bit in the pattern value is set to 1.

**Table 23-12. Set/Clear GP NVM Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	SGPB or CGPB
2	Write handshaking	DATA	GP NVM bit pattern value

General-purpose NVM bits can be read using the **Get GPNVM Bit** command (**GGPB**). The  $n^{\text{th}}$  GP NVM bit is active when bit  $n$  of the bit mask is set..

**Table 23-13. Get GP NVM Bit Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	GGPB
2	Read handshaking	DATA	GP NVM Bit Mask Status 0 = GP NVM bit is cleared 1 = GP NVM bit is set

### 23.3.5.6 Flash Security Bit Command

A security bit can be set using the **Set Security Bit** command (SSE). Once the security bit is active, the Fast Flash programming is disabled. No other command can be run. An event on the Erase pin can erase the security bit once the contents of the Flash have been erased.

**Table 23-14. Set Security Bit Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	SSE
2	Write handshaking	DATA	0

Once the security bit is set, it is not possible to access FFPI. The only way to erase the security bit is to erase the Flash.

In order to erase the Flash, the user must perform the following:

- Power-off the chip
- Power-on the chip with TST = 0
- Assert Erase during a period of more than 220 ms
- Power-off the chip

Then it is possible to return to FFPI mode and check that Flash is erased.

### 23.3.5.7 Memory Write Command

This command is used to perform a write access to any memory location.

The **Memory Write** command (**WRAM**) is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

**Table 23-15. Write Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	WRAM
2	Write handshaking	ADDR0	Memory Address LSB
3	Write handshaking	ADDR1	Memory Address
4	Write handshaking	DATA	*Memory Address++
5	Write handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	Memory Address LSB
n+1	Write handshaking	ADDR1	Memory Address
n+2	Write handshaking	DATA	*Memory Address++
n+3	Write handshaking	DATA	*Memory Address++
...	...	...	...

#### 23.3.5.8 Get Version Command

The **Get Version** (GVE) command retrieves the version of the FFPI interface.

**Table 23-16. Get Version Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	GVE
2	Read handshaking	DATA	Version

## 24. Cortex M Cache Controller (CMCC)

### 24.1 Description

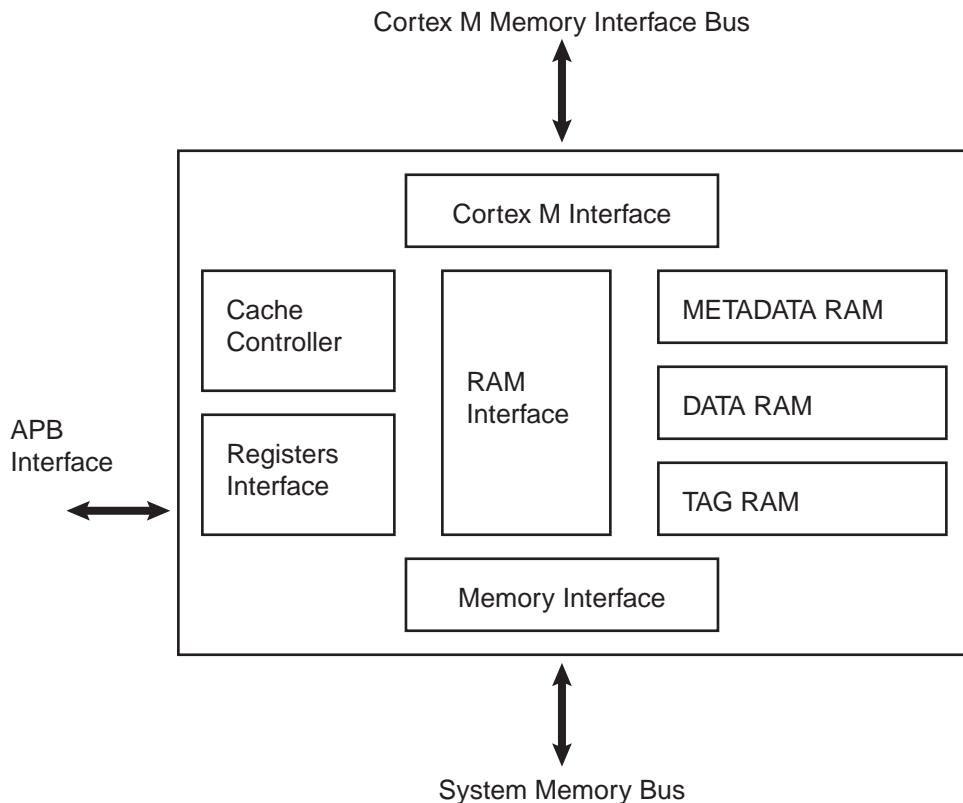
The Cortex M Cache Controller (CMCC) is a 4-Way set associative unified cache controller. It integrates a controller, a tag directory, data memory, metadata memory and a configuration interface.

### 24.2 Embedded Characteristics

- Physically addressed and physically tagged
- L1 data cache set to 2 Kbytes
- L1 cache line size set to 16 Bytes
- L1 cache integrates 32 bus master interface
- Unified Direct mapped cache architecture
- Unified 4-Way set associative cache architecture
- Write through cache operations, read allocate
- Round Robin victim selection policy
- Event Monitoring, with one programmable 32-bit counter
- Configuration registers accessible through Cortex M Private Peripheral Bus
- Cache Interface includes cache maintenance operations registers

## 24.3 Block Diagram

Figure 24-1. Block Diagram



## 24.4 Functional Description

### 24.4.1 Cache Operation

On reset, the cache controller data entries are all invalidated and the cache is disabled. The cache is transparent to processor operations. The cache controller is activated with its configuration registers. The configuration interface is memory mapped in the private peripheral bus.

Use the following sequence to enable the cache controller.

1. Verify that the cache controller is disabled, reading the value of the CSTS (cache status) field of the CMCC\_SR register.
2. Enable the cache controller, writing 1 to the CEN (cache enable) field of the CMCC\_CTRL register.

### 24.4.2 Cache Maintenance

If the contents seen by the cache has changed, the user needs to invalidate the cache entries. It can be done line by line or for all cache entries.

#### 24.4.2.1 Cache Invalidate by Line Operation

When an invalidate by line command is issued the cache controller resets the valid bit information of the decoded cache line. As the line is no longer valid the replacement counter points to that line.

Use the following sequence to invalidate one line of cache.

1. Disable the cache controller, writing 0 to the CEN field of the CMCC\_CTRL register.
2. Check CSTS field of the CMCC\_SR to verify that the cache is successfully disabled.

3. Perform an invalidate by line writing the bit set {index, way} in the CMCC\_MAINT1 register.
4. Enable the cache controller, writing 1 to the CEN field of the CMCC\_CTRL register.

#### 24.4.2.2 Cache Invalidate All Operation

To invalidate all cache entries:

Write 1 to the INVALL field of the CMCC\_MAINT0 register.

#### 24.4.3 Cache Performance Monitoring

The Cortex M cache controller includes a programmable 32-bit monitor counter. The monitor can be configured to count the number of clock cycles, the number of data hits or the number of instruction hits.

Use the following sequence to activate the counter

1. Configure the monitor counter, writing the MODE field of the CMCC\_CFG register.
2. Enable the counter, writing one to the MENABLE field of the CMCC\_MEN register.
3. If required, reset the counter, writing one to the SWRST field of the CMCC\_MCTRL register.
4. Check the value of the monitor counter, reading EVENT\_CNT field of the CMCC\_SR.

## 24.5 Cortex M Cache Controller (CMCC) User Interface

Table 24-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Cache Type Register	CMCC_TYPE	Read-only	–
0x04	Cache Configuration Register	CMCC_CFG	Read-write	0x00000000
0x08	Cache Control Register	CMCC_CTRL	Write-only	0x00000000
0x0C	Cache Status Register	CMCC_SR	Read-only	0x00000001
0x10 - 0x1C	Reserved	–	–	–
0x20	Cache Maintenance Register 0	CMCC_MAINT0	Write-only	–
0x24	Cache Maintenance Register 1	CMCC_MAINT1	Write-only	–
0x28	Cache Monitor Configuration Register	CMCC_MCFG	Read-write	0x00000000
0x2C	Cache Monitor Enable Register	CMCC_MEN	Read-write	0x00000000
0x30	Cache Monitor Control Register	CMCC_MCTRL	Write-only	–
0x34	Cache Monitor Status Register	CMCC_MSR	Read-only	0x00000000
0x38 - 0xFC	Reserved	–	–	–

#### 24.5.1 Cache Controller Type Register

**Name:** CMCC\_TYPE

**Address:** 0x400C4000

**Access:** Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—			CLSIZE		CSIZE		
7	6	5	4	3	2	1	0
LCKDOWN	WAYNUM	RRP	LRUP	RANDP	GCLK	AP	

- **AP: Access Port Access Allowed**

0: Access Port Access is disabled.

1: Access Port Access is enabled.

- **GCLK: Dynamic Clock Gating Supported**

0: Cache controller does not support clock gating.

1: Cache controller uses dynamic clock gating.

- **RANDP: Random Selection Policy Supported**

0: Random victim selection is not supported.

1: Random victim selection is supported.

- **LRUP: Least Recently Used Policy Supported**

0: Least Recently Used Policy is not supported.

1: Least Recently Used Policy is supported.

- **RRP: Random Selection Policy Supported**

0: Random Selection Policy is not supported.

1: Random Selection Policy is supported.

- **WAYNUM: Number of Way**

Value	Name	Description
0	DMAPPED	Direct Mapped Cache
1	ARCH2WAY	2-WAY set associative
2	ARCH4WAY	4-WAY set associative
3	ARCH8WAY	8-WAY set associative

- **LCKDOWN: Lock Down Supported**

0: Lock Down is not supported.

1: Lock Down is supported.

- **CSIZE: Cache Size**

Value	Name	Description
0	CSIZE_1KB	Cache Size 1 Kbytes
1	CSIZE_2KB	Cache Size 2 Kbytes
2	CSIZE_4KB	Cache Size 4 Kbytes
3	CSIZE_8KB	Cache Size 8 Kbytes

- **CLSIZE: Cache Size**

Value	Name	Description
0	CLSIZE_1KB	4 Bytes
1	CLSIZE_2KB	8 Bytes
2	CLSIZE_4KB	16 Bytes
3	CLSIZE_8KB	32 Bytes

#### 24.5.2 Cache Controller Configuration Register

**Name:** CMCC\_CFG

**Address:** 0x400C4004

**Access:** Read-write

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
-	-	-	-	-	-	-	GCLKDIS

- **GCLKDIS: Disable Clock Gating**

0: Clock gating is activated.

1: Clock gating is disabled.

### 24.5.3 Cache Controller Control Register

**Name:** CMCC\_CTRL

**Address:** 0x400C4008

**Access:** Write-only

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
-	-	-	-	-	-	-	CEN

- **CEN: Cache Controller Enable**

0: When set to 0, this field disables the cache controller.

1: When set to 1, this field enables the cache controller.

#### 24.5.4 Cache Controller Status Register

**Name:** CMCC\_SR

**Address:** 0x400C400C

**Access:** Read-only

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
-	-	-	-	-	-	-	CSTS

- **CSTS: Cache Controller Status**

0: When read as 0, this field indicates that the cache controller is disabled.

1: When read as 1, this field indicates that the cache controller is enabled.

#### 24.5.5 Cache Controller Maintenance Register 0

**Name:** CMCC\_MAINT0

**Address:** 0x400C4020

**Access:** Write-only

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
-	-	-	-	-	-	-	INVALL

- **INVALL: Cache Controller Invalidate All**

0: No effect.

1: When set to 1, this field invalidates all cache entries.

#### 24.5.6 Cache Controller Maintenance Register 1

**Name:** CMCC\_MAINT1

**Address:** 0x400C4024

**Access:** Write-only

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

- **INDEX: Invalidate Index**

This field indicates the cache line that is being invalidated.

The size of the INDEX field depends on the cache size:

- for 2 Kbytes: 5 bits
- for 4 Kbytes: 6 bits
- for 8 Kbytes: 7 bits, and so on

- **WAY: Invalidate Way**

Value	Name	Description
0	WAY0	Way 0 is selection for index invalidation
1	WAY1	Way 1 is selection for index invalidation
2	WAY2	Way 2 is selection for index invalidation
3	WAY3	Way 3 is selection for index invalidation

#### 24.5.7 Cache Controller Monitor Configuration Register

**Name:** CMCC\_MCFG

**Address:** 0x400C4028

**Access:** Write-only

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
–	–	–	–	–	–	–	MODE

- **MODE:** Cache Controller Monitor Counter Mode

Value	Name	Description
0	CYCLE_COUNT	Cycle counter
1	IHIT_COUNT	Instruction hit counter
2	DHIT_COUNT	Data hit counter

#### 24.5.8 Cache Controller Monitor Enable Register

**Name:** CMCC\_MEN

**Address:** 0x400C402C

**Access:** Write-only

**Reset:** 0x00002000

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
-	-	-	-	-	-	-	MENABLE

- **MENABLE: Cache Controller Monitor Enable**

0: When set to 0, the monitor counter is disabled.

1: When set to 1, the monitor counter is activated.

#### 24.5.9 Cache Controller Monitor Control Register

**Name:** CMCC\_MCTRL

**Address:** 0x400C4030

**Access:** Write-only

**Reset:** 0x00002000

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
-	-	-	-	-	-	-	SWRST

- **SWRST: Monitor**

0: No effect.

1: When set to 1, this field resets the event counter register.

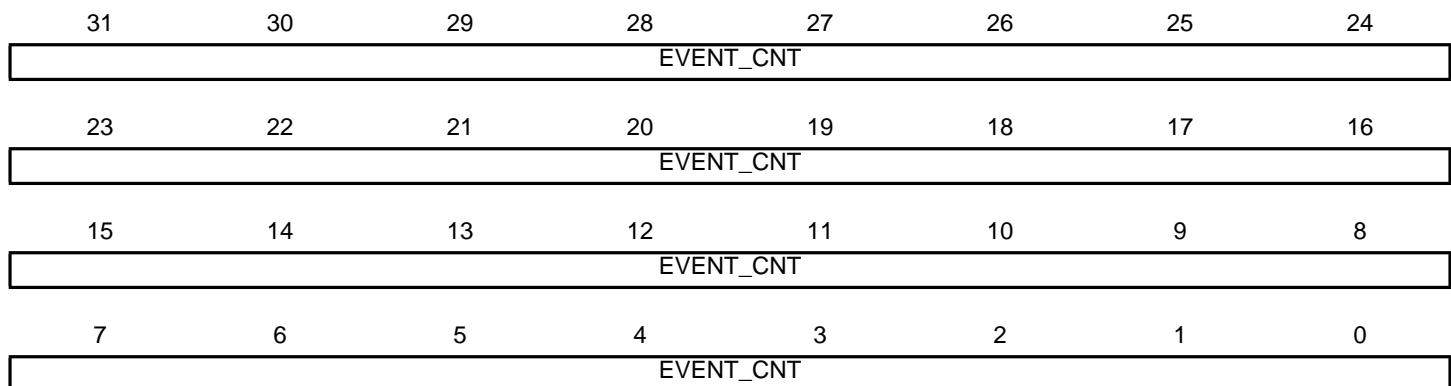
#### 24.5.10 Cache Controller Monitor Status Register

**Name:** CMCC\_MSR

**Address:** 0x400C4034

**Access:** Read-only

**Reset:** 0x00002000



- **EVENT\_CNT:** Monitor Event Counter



## 25. SAM-BA Boot Program for SAM4E Microcontrollers

### 25.1 Description

The SAM-BA® Boot Program integrates an array of programs permitting download and/or upload into the different memories of the product.

### 25.2 Embedded Characteristics

- Default Boot Program Stored in SAM4E Series Products
- Interface with SAM-BA® Graphic User Interface
- SAM-BA Boot
  - Supports Several Communication Media
    - Serial Communication on UART0
    - USB Device Port Communication up to 1M Byte/s
  - USB Requirements
    - External Crystal or Clock with the frequency of:
      - 11,289 MHz
      - 12,000 MHz
      - 16,000 MHz
      - 18,432 MHz

### 25.3 Hardware and Software Constraints

- SAM-BA Boot uses the first 2048 bytes of the SRAM for variables and stacks. The remaining available size can be used for user's code.
- USB Requirements:
  - External Crystal or External Clock<sup>(1)</sup> with frequency of:
    - 11,289 MHz
    - 12,000 MHz
    - 16,000 MHz
    - 18,432 MHz
- UART0 requirements: None

Note: 1. must be 2500 ppm and 1.8V Square Wave Signal.

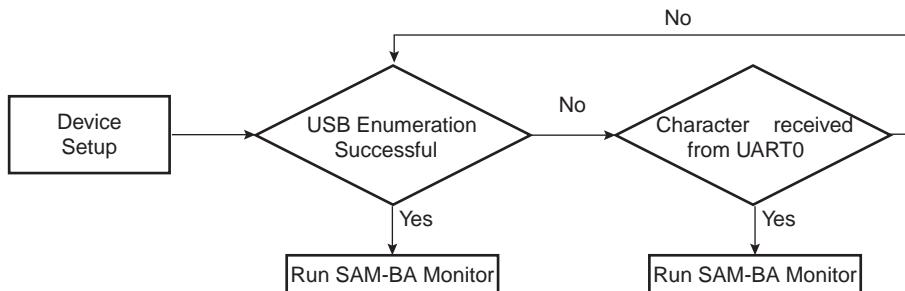
**Table 25-1. Pins Driven during Boot Program Execution**

Peripheral	Pin	PIO Line
UART0	URXD0	PA9
UART0	UTXD0	PA10

## 25.4 Flow Diagram

The Boot Program implements the algorithm in [Figure 25-1](#).

**Figure 25-1. Boot Program Algorithm Flow Diagram**



The SAM-BA Boot program seeks to detect a source clock either from the embedded main oscillator with external crystal (main oscillator enabled) or from a supported frequency signal applied to the XIN pin (Main oscillator in bypass mode).

If a clock is found from the two possible sources above, the boot program checks to verify that the frequency is one of the supported external frequencies. If the frequency is one of the supported external frequencies, USB activation is allowed, else (no clock or frequency other than one of the supported external frequencies), the internal 12 MHz RC oscillator is used as main clock and USB clock is not allowed due to frequency drift of the 12 MHz RC oscillator.

## 25.5 Device Initialization

Initialization follows the steps described below:

1. Stack setup
2. Setup the Embedded Flash Controller
3. External Clock detection (crystal or external clock on XIN)
4. If external crystal or clock with supported frequency, allow USB activation
5. Else, does not allow USB activation and use internal 12 MHz RC oscillator
6. Main oscillator frequency detection if no external clock detected
7. Switch Master Clock on Main Oscillator
8. C variable initialization
9. PLLA setup: PLLA is initialized to generate a 48 MHz clock
10. Disable of the Watchdog
11. Initialization of UART0 (115200 bauds, 8, N, 1)
12. Initialization of the USB Device Port (in case of USB activation allowed)
13. Wait for one of the following events:
  1. Check if USB device enumeration has occurred
  2. Check if characters have been received in UART0
14. Jump to SAM-BA Monitor (see [Section 25.6 "SAM-BA Monitor"](#))

## 25.6 SAM-BA Monitor

The SAM-BA boot principle:

Once the communication interface is identified, to run in an infinite loop waiting for different commands as shown in [Table 25-2](#).

**Table 25-2. Commands Available through the SAM-BA Boot**

Command	Action	Argument(s)	Example
<b>N</b>	Set Normal mode	No argument	<b>N#</b>
<b>T</b>	Set Terminal mode	No argument	<b>T#</b>
<b>O</b>	Write a byte	Address, Value#	<b>O200001,CA#</b>
<b>o</b>	Read a byte	Address,#	<b>o200001,#</b>
<b>H</b>	Write a half word	Address, Value#	<b>H200002,CAFE#</b>
<b>h</b>	Read a half word	Address,#	<b>h200002,#</b>
<b>W</b>	Write a word	Address, Value#	<b>W200000,CAFEDECA#</b>
<b>w</b>	Read a word	Address,#	<b>w200000,#</b>
<b>S</b>	Send a file	Address,#	<b>S200000,#</b>
<b>R</b>	Receive a file	Address, NbOfBytes#	<b>R200000,1234#</b>
<b>G</b>	Go	Address#	<b>G200200#</b>
<b>V</b>	Display version	No argument	<b>V#</b>

- Mode commands:
  - Normal mode configures SAM-BA Monitor to send/receive data in binary format
  - Terminal mode configures SAM-BA Monitor to send/receive data in ASCII format
- Write commands: Write a byte (**O**), a halfword (**H**) or a word (**W**) to the target
  - *Address*: Address in hexadecimal
  - *Value*: Byte, halfword or word to write in hexadecimal
- Read commands: Read a byte (**o**), a halfword (**h**) or a word (**w**) from the target
  - *Address*: Address in hexadecimal
  - *Output*: The byte, halfword or word read in hexadecimal
- Send a file (**S**): Send a file to a specified address
  - *Address*: Address in hexadecimal

Note: There is a time-out on this command which is reached when the prompt '>' appears before the end of the command execution.

- Receive a file (**R**): Receive data into a file from a specified address
  - *Address*: Address in hexadecimal
  - *NbOfBytes*: Number of bytes in hexadecimal to receive
- Go (**G**): Jump to a specified address and execute the code
  - *Address*: Address to jump in hexadecimal
- Get Version (**V**): Return the SAM-BA boot version

Note: In Terminal mode, when the requested command is performed, SAM-BA Monitor adds the following prompt sequence to its answer: <LF>+<CR>+'.'

### 25.6.1 UART0 Serial Port

Communication is performed through the UART0 initialized to 115200 Baud, 8, n, 1.

The Send and Receive File commands use the Xmodem protocol to communicate. Any terminal performing this protocol can be used to send the application file to the target. The size of the binary file to send depends on the SRAM size embedded in the product. In all cases, the size of the binary file must be lower than the SRAM size because the Xmodem protocol requires some SRAM memory to work. See, [Section 25.3 "Hardware and Software Constraints"](#)

### 25.6.2 Xmodem Protocol

The Xmodem protocol supported is the 128-byte length block. This protocol uses a two-character CRC-16 to guarantee detection of a maximum bit error.

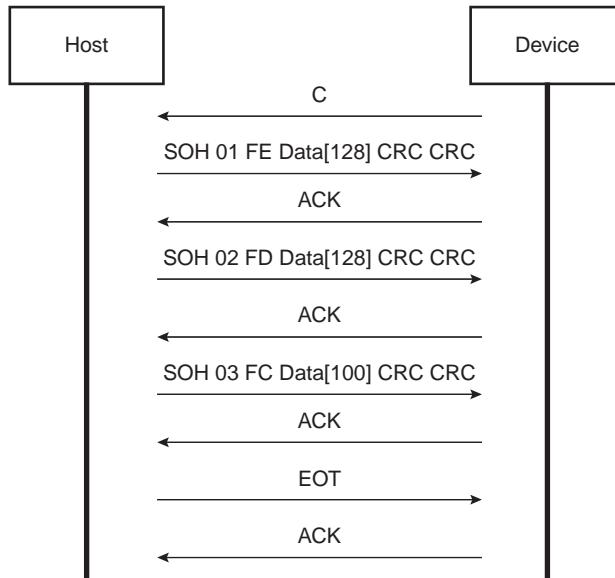
Xmodem protocol with CRC is accurate provided both sender and receiver report successful transmission. Each block of the transfer looks like:

<SOH><blk #><255-blk #><--128 data bytes--><checksum> in which:

- <SOH> = 01 hex
- <blk #> = binary number, starts at 01, increments by 1, and wraps 0FFH to 00H (not to 01)
- <255-blk #> = 1's complement of the blk#.
- <checksum> = 2 bytes CRC16

[Figure 25-2](#) shows a transmission using this protocol.

**Figure 25-2. Xmodem Transfer Example**



### 25.6.3 USB Device Port

The device uses the USB communication device class (CDC) drivers to take advantage of the installed PC RS-232 software to talk over the USB. The CDC class is implemented in all releases of Windows®, from Windows 98SE to Windows XP. The CDC document, available at [www.usb.org](http://www.usb.org), describes a way to implement devices such as ISDN modems and virtual COM ports.

The Vendor ID (VID) is Atmel's vendor ID 0x03EB. The product ID (PID) is 0x6124. These references are used by the host operating system to mount the correct driver. On Windows systems, the INF files contain the correspondence between vendor ID and product ID.

For More details about VID/PID for End Product/Systems, please refer to the Vendor ID form available from the USB Implementers Forum:

[http://www.usb.org/developers/vendor/VID\\_Only\\_Form\\_withCCAuth\\_102407b.pdf](http://www.usb.org/developers/vendor/VID_Only_Form_withCCAuth_102407b.pdf)

"Unauthorized use of assigned or unassigned USB Vendor ID Numbers and associated Product ID Numbers is strictly prohibited."

Atmel provides an INF example to see the device as a new serial port and also provides another custom driver used by the SAM-BA application: atm6124.sys. Refer to the document "USB Basic Application", [literature number 6123](#), for more details.

#### 25.6.3.1 Enumeration Process

The USB protocol is a master/slave protocol. This is the host that starts the enumeration sending requests to the device through the control endpoint. The device handles standard requests as defined in the USB Specification.

**Table 25-3. Handled Standard Requests**

Request	Definition
GET_DESCRIPTOR	Returns the current device configuration value.
SET_ADDRESS	Sets the device address for all future device access.
SET_CONFIGURATION	Sets the device configuration.
GET_CONFIGURATION	Returns the current device configuration value.
GET_STATUS	Returns status for the specified recipient.
SET_FEATURE	Set or Enable a specific feature.
CLEAR_FEATURE	Clear or Disable a specific feature.

The device also handles some class requests defined in the CDC class.

**Table 25-4. Handled Class Requests**

Request	Definition
SET_LINE_CODING	Configures DTE rate, stop bits, parity and number of character bits.
GET_LINE_CODING	Requests current DTE rate, stop bits, parity and number of character bits.
SET_CONTROL_LINE_STATE	RS-232 signal used to tell the DCE device the DTE device is now present.

Unhandled requests are STALLED.

#### 25.6.3.2 Communication Endpoints

There are two communication endpoints and endpoint 0 is used for the enumeration process. Endpoint 1 is a 64-byte Bulk OUT endpoint and endpoint 2 is a 64-byte Bulk IN endpoint. SAM-BA Boot commands are sent by the host through endpoint 1. If required, the message is split by the host into several data payloads by the host driver.

If the command requires a response, the host can send IN transactions to pick up the response.

#### 25.6.4 In Application Programming (IAP) Feature

The IAP feature is a function located in ROM that can be called by any software application.

When called, this function sends the desired FLASH command to the EEFC and waits for the Flash to be ready (looping while the FRDY bit is not set in the MC\_FSR register).

Since this function is executed from ROM, this allows Flash programming (such as sector write) to be done by code running in Flash.

The IAP function entry point is retrieved by reading the NMI vector in ROM (0x00800008).

This function takes one argument in parameter: the command to be sent to the EEFC.

This function returns the value of the MC\_FSR register.

IAP software code example:

```
(unsigned int) (*IAP_Function)(unsigned long);
void main (void){

    unsigned long FlashSectorNum = 200; //
    unsigned long flash_cmd = 0;
    unsigned long flash_status = 0;
    unsigned long EFCIndex = 0; // 0:EEFC0, 1: EEFC1

    /* Initialize the function pointer (retrieve function address from NMI vector)
     */
    IAP_Function = ((unsigned long) (*)(unsigned long))
0x00800008;

    /* Send your data to the sector here */

    /* build the command to send to EEFC */

    flash_cmd = (0x5A << 24) | (FlashSectorNum << 8) |
AT91C_MC_FCMD_EWP;

    /* Call the IAP function with appropriate command */

    flash_status = IAP_Function (EFCIndex, flash_cmd);

}
```

## 26. Bus Matrix (MATRIX)

### 26.1 Description

The Bus Matrix (MATRIX) implements a multi-layer AHB, based on the AHB-Lite protocol, that enables parallel access paths between multiple AHB masters and slaves in a system, thus increasing the overall bandwidth. The Bus Matrix interconnects up to 7 AHB masters to up to 6 AHB slaves. The normal latency to connect a master to a slave is one cycle except for the default master of the accessed slave which is connected directly (zero cycle latency). The Bus Matrix user interface is compliant with ARM Advanced Peripheral Bus.

### 26.2 Embedded Characteristics

- Configurable Number of Masters (up to 7)
- Configurable Number of Slaves (up to 6)
- One Decoder for Each Master
- Several Possible Boot Memories for Each Master before Remap
- One Remap Function for Each Master
- Support for Long Bursts of 32, 64, 128 and up to the 256-beat Word Burst AHB Limit
- Enhanced Programmable Mixed Arbitration for Each Slave
  - Round-Robin
  - Fixed Priority
- Programmable Default Master for Each Slave
  - No Default Master
  - Last Accessed Default Master
  - Fixed Default Master
- Deterministic Maximum Access Latency for Masters
- Zero or One Cycle Arbitration Latency for the First Access of a Burst
- Bus Lock Forwarding to Slaves
- Master Number Forwarding to Slaves
- One Special Function Register for Each Slave (not dedicated)

Write Protection of User Interface Registers

### 26.2.1 Matrix Masters

The Bus Matrix manages 7 masters, which means that each master can perform an access concurrently with others, to an available slave.

Each master has its own decoder, which is defined specifically for each master. In order to simplify the addressing, all the masters have the same decodings.

### 26.2.2 Matrix Slaves

The Bus Matrix manages 6 slaves. Each slave has its own arbiter, allowing a different arbitration per slave.

**Table 26-1. List of Bus Matrix Slaves**

Slave 0	Internal SRAM
Slave 1	Internal ROM
Slave 2	Internal Flash
Slave 3	Peripheral Bridge 0
Slave 4	Peripheral Bridge 1
Slave 5	External Bus Interface (EBI)

### 26.2.3 Master to Slave Access

All the Masters can normally access all the Slaves. However, some paths do not make sense, for example allowing access from the Cortex-M4 S Bus to the Internal SRAM. Thus, these paths are forbidden or simply not wired, and shown as “-” in [Table 26-2](#).

**Table 26-2. Master to Slave Access**

Slaves	Masters	0	1	2	3	4	5	6
		Cortex-M4 I/D Bus	Cortex-M4 S Bus	PDC0	PDC1	DMAC	Reserved	EMAC
0	Internal SRAM	-	X	X	X	X	-	X
1	Internal ROM	X	-	X	X	X	-	X
2	Internal Flash	X	-	-	-	-	-	-
3	Peripheral Bridge 0	-	X	X	-	X	-	-
4	Peripheral Bridge 1	-	X	-	X	X	-	-
5	External Bus Interface (EBI)	-	X	X	X	X	-	X

## 26.3 Memory Mapping

The Bus Matrix provides one decoder for every AHB master interface. The decoder offers each AHB master several memory mappings. Each memory area may be assigned to several slaves. Booting at the same address while using different AHB slaves (i.e. external RAM, internal ROM or internal Flash, etc.) becomes possible.

The Bus Matrix user interface provides the Master Remap Control Register (MATRIX\_MRCR), that performs remap action for every master independently.

The Bus Matrix user interface provides Master Remap Control Register (MATRIX\_MRCR) that performs remap action for every master independently.

## 26.4 Special Bus Granting Mechanism

The Bus Matrix provides some speculative bus granting techniques in order to anticipate access requests from masters. This mechanism reduces latency at first access of a burst, or for a single transfer, as long as the slave is free from any other master access. It does not provide any benefit if the slave is continuously accessed by more than one master, since arbitration is pipelined and has no negative effect on the slave bandwidth or access latency.

This bus granting mechanism sets a different default master for every slave.

At the end of the current access, if no other request is pending, the slave remains connected to its associated default master. A slave can be associated with three kinds of default masters:

- No default master
- Last access master
- Fixed default master

To change from one type of default master to another, the Bus Matrix user interface provides the Slave Configuration Registers, one for every slave, that set a default master for each slave. The Slave Configuration Register contains two fields: DEFMSTR\_TYPE and FIXED\_DEFMSTR. The 2-bit DEFMSTR\_TYPE field selects the default master type (no default, last access master, fixed default master), whereas the 4-bit FIXED\_DEFMSTR field selects a fixed default master provided that DEFMSTR\_TYPE is set to fixed default master. Refer to [Section 26.12.2 “Bus Matrix Slave Configuration Registers”](#).

## 26.5 No Default Master

After the end of the current access, if no other request is pending, the slave is disconnected from all masters.

This configuration incurs one latency clock cycle for the first access of a burst after bus Idle. Arbitration without default master may be used for masters that perform significant bursts or several transfers with no Idle in between, or if the slave bus bandwidth is widely used by one or more masters.

This configuration provides no benefit on access latency or bandwidth when reaching maximum slave bus throughput whatever the number of requesting masters.

## 26.6 Last Access Master

After the end of the current access, if no other request is pending, the slave remains connected to the last master that performed an access request.

This allows the Bus Matrix to remove the one latency cycle for the last master that accessed the slave. Other non privileged masters still get one latency clock cycle if they want to access the same slave. This technique is useful for masters that mainly perform single accesses or short bursts with some Idle cycles in between.

This configuration provides no benefit on access latency or bandwidth when reaching maximum slave bus throughput whatever is the number of requesting masters.

## 26.7 Fixed Default Master

After the end of the current access, if no other request is pending, the slave connects to its fixed default master. Unlike the last access master, the fixed default master does not change unless the user modifies it by software (FIXED\_DEFMSTR field of the related MATRIX\_SCFG).

This allows the Bus Matrix arbiters to remove the one latency clock cycle for the fixed default master of the slave. All requests attempted by the fixed default master do not cause any arbitration latency, whereas other non-privileged masters will get one latency cycle. This technique is useful for a master that mainly performs single accesses or short bursts with Idle cycles in between.

This configuration provides no benefit on access latency or bandwidth when reaching maximum slave bus throughput, regardless of the number of requesting masters.

## 26.8 Arbitration

The Bus Matrix provides an arbitration mechanism that reduces latency when conflict cases occur, i.e. when two or more masters try to access the same slave at the same time. One arbiter per AHB slave is provided, thus arbitrating each slave specifically.

The Bus Matrix provides the user with the possibility of choosing between 2 arbitration types or mixing them for each slave:

1. Round-robin Arbitration (default)
2. Fixed Priority Arbitration

The resulting algorithm may be complemented by selecting a default master configuration for each slave.

When re-arbitration must be done, specific conditions apply. See [Section 26.8.1 "Arbitration Scheduling"](#).

### 26.8.1 Arbitration Scheduling

Each arbiter has the ability to arbitrate between two or more different master requests. In order to avoid burst breaking and also to provide the maximum throughput for slave interfaces, arbitration may only take place during the following cycles:

1. Idle Cycles: When a slave is not connected to any master or is connected to a master which is not currently accessing it.
2. Single Cycles: When a slave is currently doing a single access.
3. End of Burst Cycles: When the current cycle is the last cycle of a burst transfer. For defined length burst, predicted end of burst matches the size of the transfer but is managed differently for undefined length burst. See [Section 26.8.1.1 "Undefined Length Burst Arbitration"](#)
4. Slot Cycle Limit: When the slot cycle counter has reached the limit value indicating that the current master access is too long and must be broken. See [Section 26.8.1.2 "Slot Cycle Limit Arbitration"](#).

#### 26.8.1.1 Undefined Length Burst Arbitration

In order to prevent long AHB burst lengths that can lock the access to the slave for an excessive period of time, the user can trigger the re-arbitration before the end of the incremental bursts. The re-arbitration period can be selected from the following Undefined Length Burst Type (ULBT) possibilities:

1. Unlimited: no predetermined end of burst is generated. This value enables 1-Kbyte burst lengths.
2. 1-beat bursts: predetermined end of burst is generated at each single transfer during the INCR transfer.
3. 4-beat bursts: predetermined end of burst is generated at the end of each 4-beat boundary during INCR transfer.
4. 8-beat bursts: predetermined end of burst is generated at the end of each 8-beat boundary during INCR transfer.
5. 16-beat bursts: predetermined end of burst is generated at the end of each 16-beat boundary during INCR transfer.
6. 32-beat bursts: predetermined end of burst is generated at the end of each 32-beat boundary during INCR transfer.
7. 64-beat bursts: predetermined end of burst is generated at the end of each 64-beat boundary during INCR transfer.
8. 128-beat bursts: predetermined end of burst is generated at the end of each 128-beat boundary during INCR transfer.

The use of undefined length 16-beat bursts, or less, is discouraged since this generally decreases significantly the overall bus bandwidth due to arbitration and slave latencies at each first access of a burst.

If the master does not permanently and continuously request the same slave or has an intrinsically limited average throughput, the ULBT should be left at its default unlimited value, knowing that the AHB specification natively limits all word bursts to 256 beats and double-word bursts to 128 beats because of its 1 Kilobyte address boundaries.

Unless duly needed, the ULBT should be left at its default value of 0 for power saving.

This selection can be done through the ULBT field of the Master Configuration Registers (MATRIX\_MCFG).

#### 26.8.1.2 Slot Cycle Limit Arbitration

The Bus Matrix contains specific logic to break long accesses, such as very long bursts on a very slow slave (e.g., an external low speed memory). At each arbitration time, a counter is loaded with the value previously written in the SLOT\_CYCLE field of the related Slave Configuration Register (MATRIX\_SCFG) and decreased at each clock cycle. When the counter elapses, the arbiter has the ability to re-arbitrate at the end of the current AHB bus access cycle.

Unless a master has a very tight access latency constraint, which could lead to data overflow or underflow due to a badly undersized internal FIFO with respect to its throughput, the Slot Cycle Limit should be disabled (SLOT\_CYCLE = 0) or set to its default maximum value in order not to inefficiently break long bursts performed by some Atmel masters.

In most cases, this feature is not needed and should be disabled for power saving.

**Warning:** This feature cannot prevent any slave from locking its access indefinitely.

#### 26.8.2 Arbitration Priority Scheme

The bus Matrix arbitration scheme is organized in priority pools.

Round-robin priority is used in the highest and lowest priority pools, whereas fixed level priority is used between priority pools and in the intermediate priority pools.

For each slave, each master is assigned to one of the slave priority pools through the priority registers for slaves (MxPR fields of MATRIX\_PRAS and MATRIX\_PRBS). When evaluating master requests, this programmed priority level always takes precedence.

After reset, all the masters belong to the lowest priority pool (MxPR = 0) and are therefore granted bus access in a true round-robin order.

The highest priority pool must be specifically reserved for masters requiring very low access latency. If more than one master belongs to this pool, they will be granted bus access in a biased round-robin manner which allows tight and deterministic maximum access latency from AHB bus requests. In the worst case, any currently occurring high-priority master request will be granted after the current bus master access has ended and other high priority pool master requests, if any, have been granted once each.

The lowest priority pool shares the remaining bus bandwidth between AHB Masters.

Intermediate priority pools allow fine priority tuning. Typically, a moderately latency-critical master or a bandwidth-only critical master will use such a priority level. The higher the priority level (MxPR value), the higher the master priority.

All combinations of MxPR values are allowed for all masters and slaves. For example, some masters might be assigned the highest priority pool (round-robin), and remaining masters the lowest priority pool (round-robin), with no master for intermediate fix priority levels.

If more than one master requests the slave bus, regardless of the respective masters priorities, no master will be granted the slave bus for two consecutive runs. A master can only get back-to-back grants so long as it is the only requesting master.

#### 26.8.2.1 Fixed Priority Arbitration

Fixed priority arbitration algorithm is the first and only arbitration algorithm applied between masters from distinct priority pools. It is also used in priority pools other than the highest and lowest priority pools (intermediate priority pools).

Fixed priority arbitration allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave by using the fixed priority defined by the user in the MxPR field for each master in the Priority Registers, MATRIX\_PRAS and MATRIX\_PRBS. If two or more master requests are active at the same time, the master with the highest priority MxPR number is serviced first.

In intermediate priority pools, if two or more master requests with the same priority are active at the same time, the master with the highest number is serviced first.

#### 26.8.2.2 Round-Robin Arbitration

This algorithm is only used in the highest and lowest priority pools. It allows the Bus Matrix arbiters to properly dispatch requests from different masters to the same slave. If two or more master requests are active at the same time in the priority pool, they are serviced in a round-robin increasing master number order.

## 26.9 System I/O Configuration

The System I/O Configuration register (CCFG\_SYSIO) allows to configure some I/O lines in System I/O mode (such as JTAG, ERASE, USB, etc...) or as general purpose I/O lines. Enabling or disabling the corresponding I/O lines in peripheral mode or in PIO mode (PIO\_PER or PIO\_PDR registers) in the PIO controller as no effect. However, the direction (input or output), pull-up, pull-down and other mode control is still managed by the PIO controller.

## 26.10 SMC NAND Flash Chip Select Configuration

The SMC Nand Flash Chip Select Configuration Register (CCFG\_SMCNFCS) allow to manage the chip select signal (NCSx) as assigned to NAND Flash or not.

Each NCSx can be individually assigned to Nand Flash or not. When the NCSx is assigned to NANDFLASH, the signals NANDOE and NANDWE are used for the NCSx signals selected.

## 26.11 Write Protect Registers

To prevent any single software error that may corrupt the Bus Matrix behavior, the entire Bus Matrix address space can be write-protected by setting the WPEN bit in the Bus Matrix Write Protect Mode Register (MATRIX\_WPMR).

If WPEN is at one and a write access in the Bus Matrix address space is detected, then the WPVS flag in the Bus Matrix Write Protect Status Register (MATRIX\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the Bus Matrix Write Protect Mode Register (MATRIX\_WPMR) with the appropriate access key WPKEY.

The protected registers are:

["Bus Matrix Master Configuration Registers"](#)

["Bus Matrix Slave Configuration Registers"](#)

["Bus Matrix Priority Registers A For Slaves"](#)

["Bus Matrix Master Remap Control Register"](#)

["Write Protect Mode Register"](#)

## 26.12 Bus Matrix (MATRIX) User Interface

Table 26-3. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Master Configuration Register 0	MATRIX_MCFG0	Read-write	0x00000001
0x0004	Master Configuration Register 1	MATRIX_MCFG1	Read-write	0x00000000
0x0008	Master Configuration Register 2	MATRIX_MCFG2	Read-write	0x00000000
0x000C	Master Configuration Register 3	MATRIX_MCFG3	Read-write	0x00000000
0x0010	Master Configuration Register 4	MATRIX_MCFG4	Read-write	0x00000000
0x0014	Master Configuration Register 5	MATRIX_MCFG5	Read-write	0x00000000
0x0018	Master Configuration Register 6	MATRIX_MCFG6	Read-write	0x00000000
0x001C-0x003C	Reserved	—	—	—
0x0040	Slave Configuration Register 0	MATRIX_SCFG0	Read-write	0x000001FF
0x0044	Slave Configuration Register 1	MATRIX_SCFG1	Read-write	0x000001FF
0x0048	Slave Configuration Register 2	MATRIX_SCFG2	Read-write	0x000001FF
0x004C	Slave Configuration Register 3	MATRIX_SCFG3	Read-write	0x000001FF
0x0050	Slave Configuration Register 4	MATRIX_SCFG4	Read-write	0x000001FF
0x0054	Slave Configuration Register 5	MATRIX_SCFG5	Read-write	0x000001FF
0x0058-0x007C	Reserved	—	—	—
0x0080	Priority Register A for Slave 0	MATRIX_PRAS0	Read-write	0x33333333 <sup>(1)</sup>
0x0084	Reserved	—	—	—
0x0088	Priority Register A for Slave 1	MATRIX_PRAS1	Read-write	0x33333333 <sup>(1)</sup>
0x008C	Reserved	—	—	—
0x0090	Priority Register A for Slave 2	MATRIX_PRAS2	Read-write	0x33333333 <sup>(1)</sup>
0x0094	Reserved	—	—	—
0x0098	Priority Register A for Slave 3	MATRIX_PRAS3	Read-write	0x33333333 <sup>(1)</sup>
0x009C	Reserved	—	—	—
0x00A0	Priority Register A for Slave 4	MATRIX_PRAS4	Read-write	0x33333333 <sup>(1)</sup>
0x00A4	Reserved	—	—	—
0x00A8	Priority Register A for Slave 5	MATRIX_PRAS5	Read-write	0x33333333 <sup>(1)</sup>
0x00AC	Reserved	—	—	—
0x00B4-0x00FC	Reserved	—	—	—
0x0100	Master Remap Control Register	MATRIX_MRCR	Read-write	0x00000000
0x0104 - 0x010C	Reserved	—	—	—
0x0110	Reserved	—	—	—
0x0114	System I/O Configuration Register	CCFG_SYSIO	Read-write	0x00000000
0x0118 - 0x0120	Reserved	—	—	—
0x0124	SMC NAND Flash Chip Select Configuration Register	CCFG_SMCNFCS	Read-write	0x00000000

**Table 26-3. Register Mapping (Continued)**

Offset	Register	Name	Access	Reset
0x0128 - 0x01E0	Reserved	-	-	-
0x01E4	Write Protect Mode Register	MATRIX_WPMR	Read-write	0x00000000
0x01E8	Write Protect Status Register	MATRIX_WPSR	Read-only	0x00000000

Note: 1. Values in the Bus Matrix Priority Registers are product dependent.

### 26.12.1 Bus Matrix Master Configuration Registers

**Name:** MATRIX\_MCFGx [x=0..6]

**Address:** 0x400E0200

**Access:** Read-write

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
–	–	–	–	–	–	ULBT	

This register can only be written if the WPEN bit is cleared in the [“Write Protect Mode Register”](#).

- **ULBT: Undefined Length Burst Type**

0: Unlimited Length Burst

No predicted end of burst is generated, therefore INCR bursts coming from this master can only be broken if the Slave Slot Cycle Limit is reached. If the Slot Cycle Limit is not reached, the burst is normally completed by the master, at the latest, on the next AHB 1-Kbyte address boundary, allowing up to 256-beat word bursts or 128-beat double-word bursts.

This value should not be used in the very particular case of a master capable of performing back-to-back undefined length bursts on a single slave, since this could indefinitely freeze the slave arbitration and thus prevent another master from accessing this slave.

1: Single Access

The undefined length burst is treated as a succession of single accesses, allowing re-arbitration at each beat of the INCR burst or bursts sequence.

2: 4-beat Burst

The undefined length burst or bursts sequence is split into 4-beat bursts or less, allowing re-arbitration every 4 beats.

3: 8-beat Burst

The undefined length burst or bursts sequence is split into 8-beat bursts or less, allowing re-arbitration every 8 beats.

4: 16-beat Burst

The undefined length burst or bursts sequence is split into 16-beat bursts or less, allowing re-arbitration every 16 beats.

5: 32-beat Burst

The undefined length burst or bursts sequence is split into 32-beat bursts or less, allowing re-arbitration every 32 beats.

6: 64-beat Burst

The undefined length burst or bursts sequence is split into 64-beat bursts or less, allowing re-arbitration every 64 beats.

7: 128-beat Burst

The undefined length burst or bursts sequence is split into 128-beat bursts or less, allowing re-arbitration every 128 beats.

Unless duly needed, the ULBT should be left at its default 0 value for power saving.

### 26.12.2 Bus Matrix Slave Configuration Registers

**Name:** MATRIX\_SCFGx [x=0..5]

**Address:** 0x400E0240

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in the “[Write Protect Mode Register](#)” .

- **SLOT\_CYCLE: Maximum Bus Grant Duration for Masters**

When SLOT\_CYCLE AHB clock cycles have elapsed since the last arbitration, a new arbitration takes place to let another master access this slave. If another master is requesting the slave bus, then the current master burst is broken.

If SLOT\_CYCLE = 0, the Slot Cycle Limit feature is disabled and bursts always complete unless broken according to the ULBT. This limit has been placed in order to enforce arbitration so as to meet potential latency constraints of masters waiting for slave access.

This limit must not be too small. Unreasonably small values break every burst and the Bus Matrix arbitrates without performing any data transfer. The default maximum value is usually an optimal conservative choice.

In most cases, this feature is not needed and should be disabled for power saving.

See “[Slot Cycle Limit Arbitration](#)” for details.

- **DEFMSTR\_TYPE: Default Master Type**

0: No Default Master

At the end of the current slave access, if no other master request is pending, the slave is disconnected from all masters.

This results in a one clock cycle latency for the first access of a burst transfer or for a single access.

1: Last Default Master

At the end of the current slave access, if no other master request is pending, the slave stays connected to the last master having accessed it.

This results in not having one clock cycle latency when the last master tries to access the slave again.

2: Fixed Default Master

At the end of the current slave access, if no other master request is pending, the slave connects to the fixed master the number that has been written in the FIXED\_DEFMSTR field.

This results in not having one clock cycle latency when the fixed master tries to access the slave again.

- **FIXED\_DEFMSTR: Fixed Default Master**

This is the number of the Default Master for this slave. Only used if DEFMSTR\_TYPE is 2. Specifying the number of a master which is not connected to the selected slave is equivalent to setting DEFMSTR\_TYPE to 0.

### 26.12.3 Bus Matrix Priority Registers A For Slaves

**Name:** MATRIX\_PRAS $x$  [x=0..5]

**Address:** 0x400E0280 [0], 0x400E0288 [1], 0x400E0290 [2], 0x400E0298 [3], 0x400E02A0 [4], 0x400E02A8 [5]

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	M7PR	-	-	-	M6PR	
23	22	21	20	19	18	17	16
-	-	M5PR	-	-	-	M4PR	
15	14	13	12	11	10	9	8
-	-	M3PR	-	-	-	M2PR	
7	6	5	4	3	2	1	0
-	-	M1PR	-	-	-	M0PR	

This register can only be written if the WPE bit is cleared in the “[Write Protect Mode Register](#)” .

- **MxPR: Master x Priority**

Fixed priority of Master x for accessing the selected slave. The higher the number, the higher the priority.

All the masters programmed with the same MxPR value for the slave make up a priority pool.

Round-robin arbitration is used in the lowest (MxPR = 0) and highest (MxPR = 3) priority pools.

Fixed priority is used in intermediate priority pools (MxPR = 1) and (MxPR = 2).

See “[Arbitration Priority Scheme](#)” for details.

#### 26.12.4 Bus Matrix Master Remap Control Register

**Name:** MATRIX\_MRCR

**Address:** 0x400E0300

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
RCB15	RCB14	RCB13	RCB12	RCB11	RCB10	RCB9	RCB8
7	6	5	4	3	2	1	0
RCB7	RCB6	RCB5	RCB4	RCB3	RCB2	RCB1	RCB0

This register can only be written if the WPEN bit is cleared in the “[Write Protect Mode Register](#)” .

- **RCBx: Remap Command Bit for Master x**

0: Disable remapped address decoding for the selected Master

1: Enable remapped address decoding for the selected Master

## 26.12.5 System I/O Configuration Register

**Name:** CCFG\_SYSIO

**Address:** 0x400E0314

**Access** Read-write

**Reset:** 0x0000\_0000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	SYSIO12	SYSIO11	SYSIO10	-	-
7	6	5	4	3	2	1	0
SYSIO7	SYSIO6	SYSIO5	SYSIO4	-	-	-	-

- **SYSIO4: PB4 or TDI Assignment**

0 = TDI function selected.

1 = PB4 function selected.

- **SYSIO5: PB5 or TDO/TRACESWO Assignment**

0 = TDO/TRACESWO function selected.

1 = PB5 function selected.

- **SYSIO6: PB6 or TMS/SWDIO Assignment**

0 = TMS/SWDIO function selected.

1 = PB6 function selected.

- **SYSIO7: PB7 or TCK/SWCLK Assignment**

0 = TCK/SWCLK function selected.

1 = PB7 function selected.

- **SYSIO10: PB10 or DDM Assignment**

0 = DDM function selected.

1 = PB10 function selected.

- **SYSIO11: PB11 or DDP Assignment**

0 = DDP function selected.

1 = PB11 function selected.

- **SYSIO12: PB12 or ERASE Assignment**

0 = ERASE function selected.

1 = PB12 function selected.

## 26.12.6 SMC NAND Flash Chip Select Configuration Register

**Name:** CCFG\_SMCNFCS

**Address:** 0x400E0324

**Type:** Read-write

**Reset:** 0x0000\_0000

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
-	-	-	-	SMC_NFCS3	SMC_NFCS2	SMC_NFCS1	SMC_NFCS0

- **SMC\_NFCS0: SMC NAND Flash Chip Select 0 Assignment**

0 = NCS0 is not assigned to a NAND Flash (NANDOE and NANWE not used for NCS0)

1 = NCS0 is assigned to a NAND Flash (NANDOE and NANWE used for NCS0)

- **SMC\_NFCS1: SMC NAND Flash Chip Select 1 Assignment**

0 = NCS1 is not assigned to a NAND Flash (NANDOE and NANWE not used for NCS1)

1 = NCS1 is assigned to a NAND Flash (NANDOE and NANWE used for NCS1)

- **SMC\_NFCS2: SMC NAND Flash Chip Select 2 Assignment**

0 = NCS2 is not assigned to a NAND Flash (NANDOE and NANWE not used for NCS2)

1 = NCS2 is assigned to a NAND Flash (NANDOE and NANWE used for NCS2)

- **SMC\_NFCS3: SMC NAND Flash Chip Select 3 Assignment**

0 = NCS3 is not assigned to a NAND Flash (NANDOE and NANWE not used for NCS3)

1 = NCS3 is assigned to a NAND Flash (NANDOE and NANWE used for NCS3)

### 26.12.7 Write Protect Mode Register

**Name:** MATRIX\_WPMR

**Address:** 0x400E03E4

**Access:** Read-write

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

For more details on MATRIX\_WPMR, please refer to [Section 26.11 “Write Protect Registers”](#).

The protected registers are:

[“Bus Matrix Master Configuration Registers”](#)

[“Bus Matrix Slave Configuration Registers”](#)

[“Bus Matrix Priority Registers A For Slaves”](#)

[“Bus Matrix Master Remap Control Register”](#)

[“Write Protect Mode Register”](#)

- **WPEN: Write Protect Enable**

0: Disables the Write Protect if WPKEY corresponds to 0x4D4154 (“MAT” in ASCII).

1: Enables the Write Protect if WPKEY corresponds to 0x4D4154 (“MAT” in ASCII).

Protects the entire Bus Matrix address space from address offset 0x000 to 0x1FC.

- **WPKEY: Write Protect KEY (Write-only)**

Value	Name	Description
0x4D4154	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

## 26.12.8 Write Protect Status Register

**Name:** MATRIX\_WPSR

**Address:** 0x400E03E8

**Access:** Read-only

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

For more details on MATRIX\_WPSR, please refer to [Section 26.11 “Write Protect Registers”](#).

- **WPVS: Write Protect Violation Status**

0: No Write Protect Violation has occurred since the last write of the MATRIX\_WPMR.

1: At least one Write Protect Violation has occurred since the last write of the MATRIX\_WPMR.

- **WPVSR: Write Protect Violation Source**

When WPVS is active, this field indicates the register address offset in which a write access has been attempted.

Otherwise it reads as 0.

## 27. DMA Controller (DMAC)

### 27.1 Description

The DMA Controller (DMAC) is an AHB-central DMA controller core that transfers data from a source peripheral to a destination peripheral over one or more AMBA buses. One channel is required for each source/destination pair. In the most basic configuration, the DMAC has one master interface and one channel. The master interface reads the data from a source and writes it to a destination. Two AMBA transfers are required for each DMAC data transfer. This is also known as a dual-access transfer.

The DMAC is programmed via the APB interface.

### 27.2 Embedded Characteristics

- 1 AHB-Lite Master Interfaces
- DMA Module Supports the Following Transfer Schemes: Peripheral-to-Memory, Memory-to-Peripheral, Peripheral-to-Peripheral and Memory-to-Memory
- Source and Destination Operate independently on BYTE (8-bit), HALF-WORD (16-bit) and WORD (32-bit)
- Supports Hardware and Software Initiated Transfers
- Supports Multiple Buffer Chaining Operations
- Supports Incrementing/decrementing/fixed Addressing Mode Independently for Source and Destination
- Programmable Arbitration Policy, Modified Round Robin and Fixed Priority are Available
- Supports Specified Length and Unspecified Length AMBA AHB Burst Access to Maximize Data Bandwidth
- AMBA APB Interface Used to Program the DMA Controller
- 4 DMA Channels
- 16 External Request Lines
- Embedded FIFO
- Channel Locking and Bus Locking Capability

The DMA Controller can handle the transfer between peripherals and memory and so receives the triggers from the peripherals listed below.

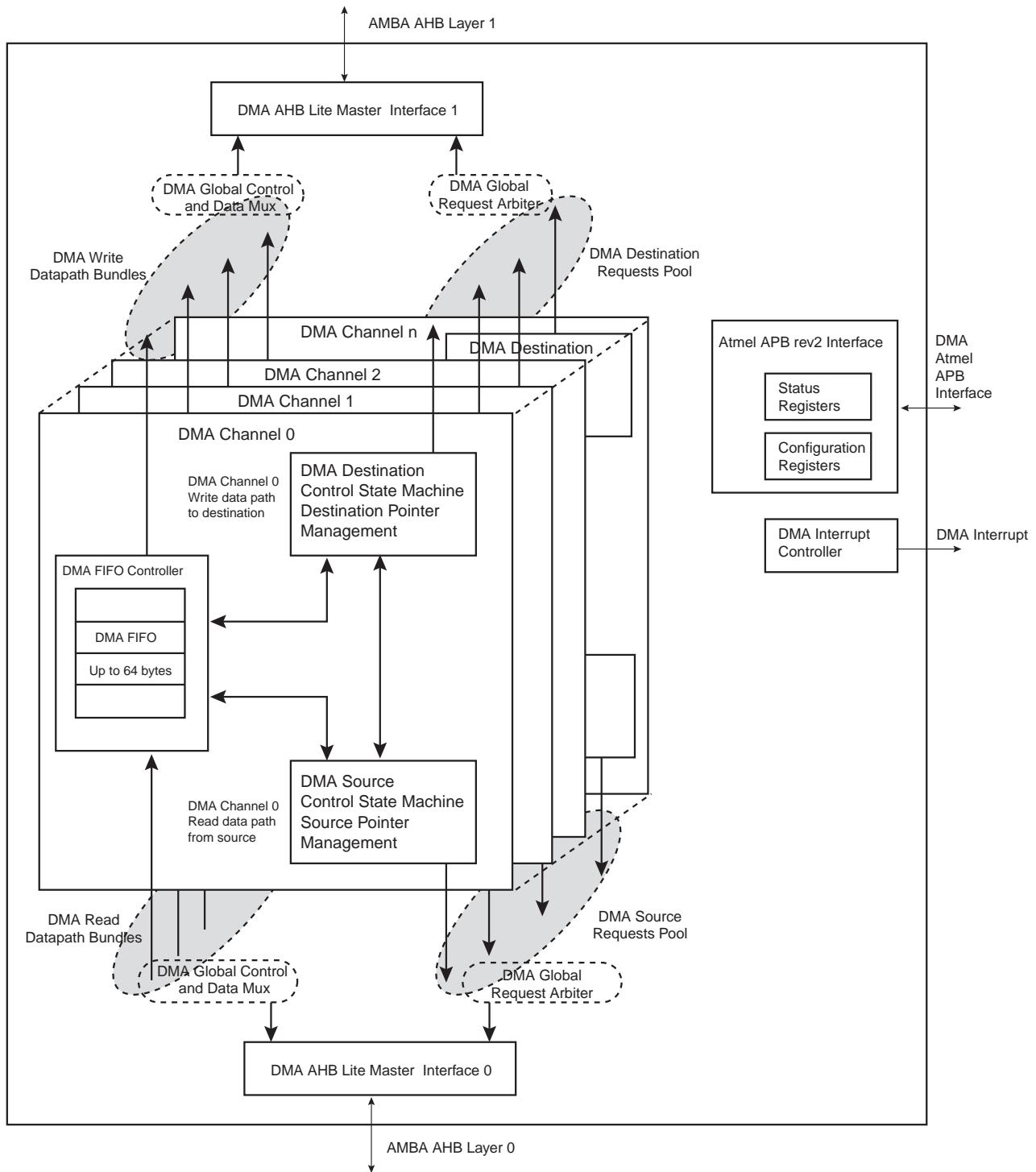
The hardware interface numbers are given in [Table 27-1](#).

Table 27-1. DMA Controller

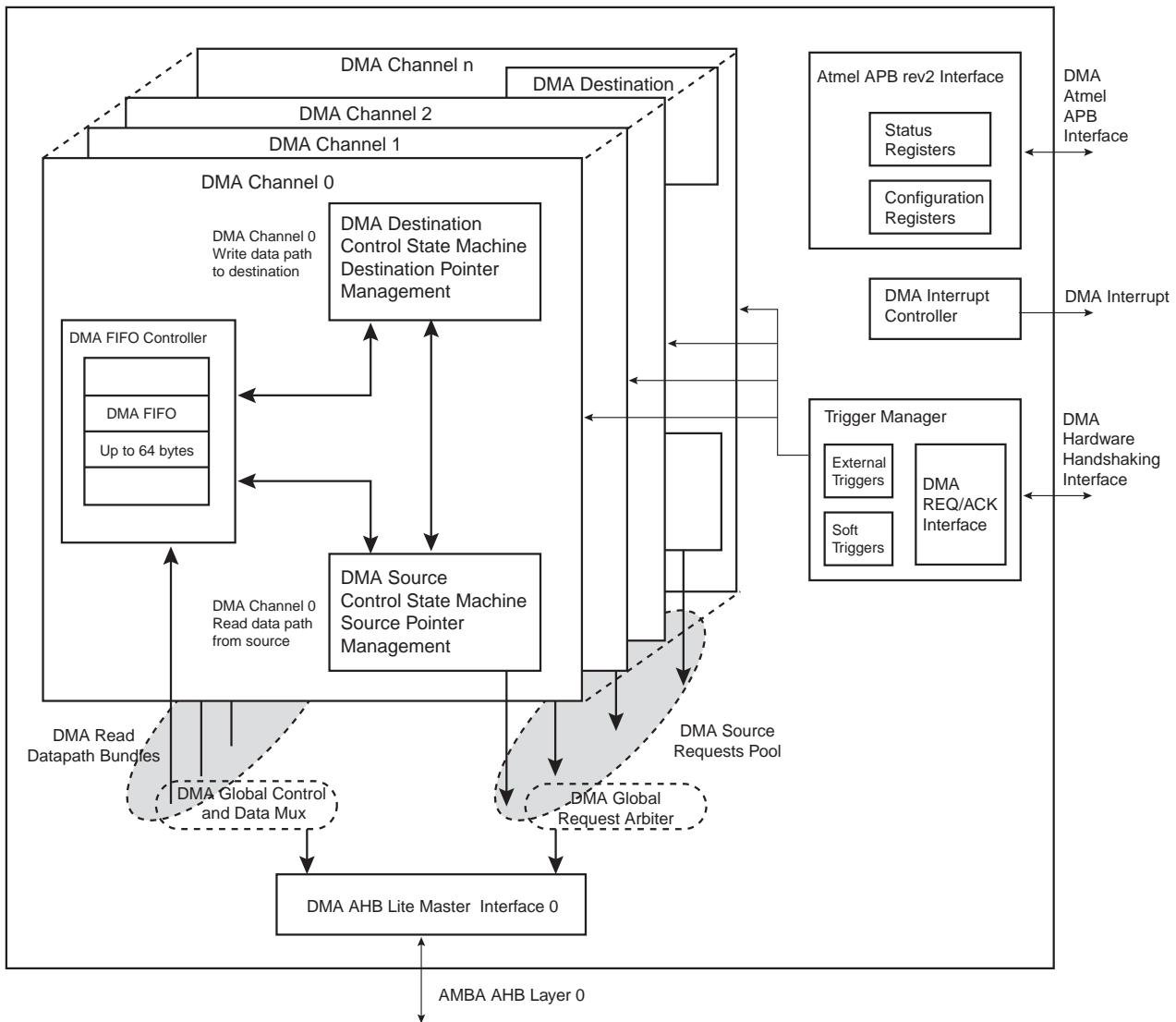
Instance Name	Channel T/R	DMA Channel HW Interface Number
HSMCI	Transmit/Receive	0
SPI	Transmit	1
SPI	Receive	2
USART0	Transmit	3
USART0	Receive	4
USART1	Transmit	5
USART1	Receive	6
AES	Transmit	11
AES	Receive	12
PWM	Transmit	13

## 27.3 Block Diagram

Figure 27-1. DMA Controller (DMAC) Block Diagram



**Figure 27-2. DMA Controller (DMAC) Block Diagram**



## 27.4 Functional Description

### 27.4.1 Basic Definitions

**Source peripheral:** Device on an AMBA layer from where the DMAC reads data, which is then stored in the channel FIFO. The source peripheral teams up with a destination peripheral to form a channel.

**Destination peripheral:** Device to which the DMAC writes the stored data from the FIFO (previously read from the source peripheral).

**Memory:** Source or destination that is always “ready” for a DMAC transfer and does not require a handshaking interface to interact with the DMAC.

**Programmable Arbitration Policy:** Modified Round Robin and Fixed Priority are available by means of the ARB\_CFG bit in the Global Configuration Register (DMAC\_GCFG). The fixed priority is linked to the channel number. The highest DMAC channel number has the highest priority.

**Channel:** Read/write datapath between a source peripheral on one configured AMBA layer and a destination peripheral on the same or different AMBA layer that occurs through the channel FIFO. If the source peripheral is not memory, then a source handshaking interface is assigned to the channel. If the destination peripheral is not memory, then a destination handshaking interface is assigned to the channel. Source and destination handshaking interfaces can be assigned dynamically by programming the channel registers.

**Master interface:** DMAC is a master on the AHB bus reading data from the source and writing it to the destination over the AHB bus.

**Slave interface:** The APB interface over which the DMAC is programmed. The slave interface in practice could be on the same layer as any of the master interfaces or on a separate layer.

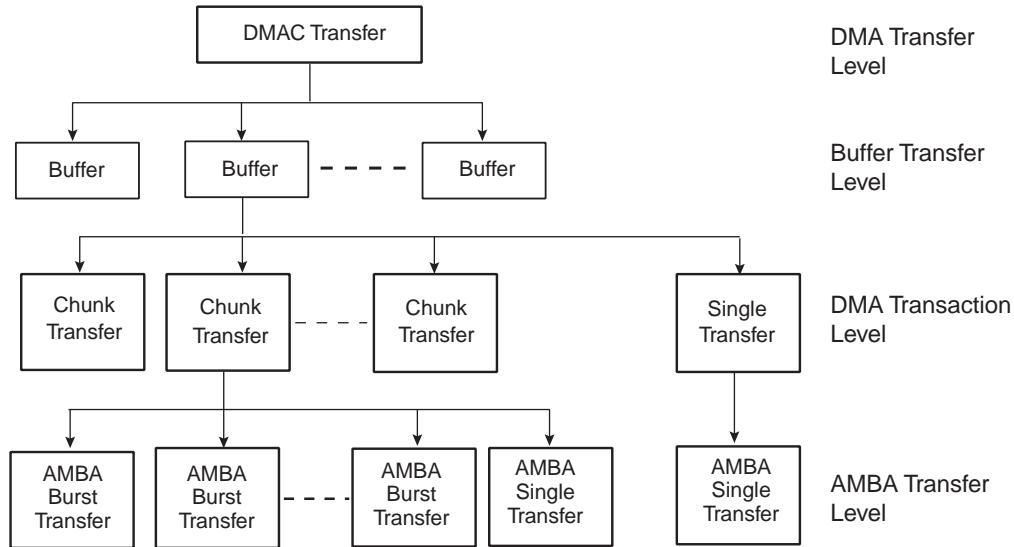
**Handshaking interface:** A set of signal registers that conform to a protocol and *handshake* between the DMAC and source or destination peripheral to control the transfer of a single or chunk transfer between them. This interface is used to request, acknowledge, and control a DMAC transaction. A channel can receive a request through one of two types of handshaking interface: hardware or software.

**Hardware handshaking interface:** Uses hardware signals to control the transfer of a single or chunk transfer between the DMAC and the source or destination peripheral.

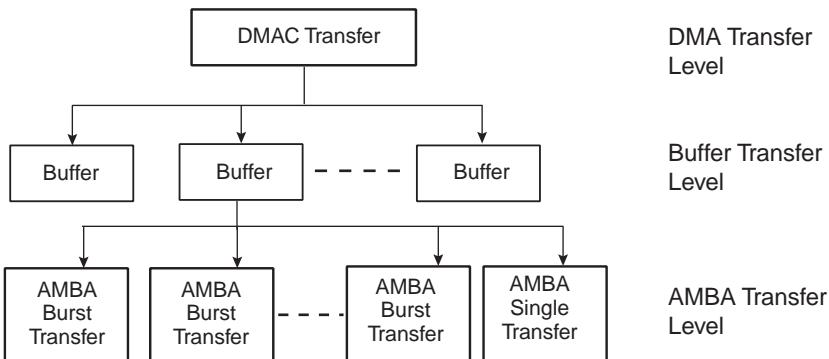
**Software handshaking interface:** Uses software registers to control the transfer of a single or chunk transfer between the DMAC and the source or destination peripheral. No special DMAC handshaking signals are needed on the I/O of the peripheral. This mode is useful for interfacing an existing peripheral to the DMAC without modifying it.

**Transfer hierarchy:** [Figure 27-3 on page 431](#) illustrates the hierarchy between DMAC transfers, buffer transfers, chunk or single, and AMBA transfers (single or burst) for non-memory peripherals. [Figure 27-4 on page 431](#) shows the transfer hierarchy for memory.

**Figure 27-3. DMAC Transfer Hierarchy for Non-Memory Peripheral**



**Figure 27-4. DMAC Transfer Hierarchy for Memory**



**Buffer:** A buffer of DMAC data. The amount of data (length) is determined by the flow controller. For transfers between the DMAC and memory, a buffer is broken directly into a sequence of AMBA bursts and AMBA single transfers.

For transfers between the DMAC and a non-memory peripheral, a buffer is broken into a sequence of DMAC transactions (single and chunks). These are in turn broken into a sequence of AMBA transfers.

**Transaction:** A basic unit of a DMAC transfer as determined by either the hardware or software handshaking interface. A transaction is only relevant for transfers between the DMAC and a source or destination peripheral if the source or destination peripheral is a non-memory device. There are two types of transactions: single transfer and chunk transfer.

- **Single transfer:** The length of a single transaction is always 1 and is converted to a single AMBA access.
- **Chunk transfer:** The length of a chunk is programmed into the DMAC. The chunk is then converted into a sequence of AHB access. DMAC executes each AMBA burst transfer by performing incremental bursts that are no longer than 16 beats.

**DMAC transfer:** Software controls the number of buffers in a DMAC transfer. Once the DMAC transfer has completed, then hardware within the DMAC disables the channel and can generate an interrupt to signal the completion of the DMAC transfer. You can then re-program the channel for a new DMAC transfer.

**Single-buffer DMAC transfer:** Consists of a single buffer.

**Multi-buffer DMAC transfer:** A DMAC transfer may consist of multiple DMAC buffers. Multi-buffer DMAC transfers are supported through buffer chaining (linked list pointers), auto-reloading of channel registers, and contiguous buffers. The source and destination can independently select which method to use.

- **Linked lists (buffer chaining)** – A descriptor pointer (DSCR) points to the location in system memory where the next linked list item (LLI) exists. The LLI is a set of registers that describe the next buffer (buffer descriptor) and a descriptor pointer register. The DMAC fetches the LLI at the beginning of every buffer when buffer chaining is enabled.
- **Contiguous buffers** – Where the address of the next buffer is selected to be a continuation from the end of the previous buffer.

**Channel locking:** Software can program a channel to keep the AHB master interface by locking the arbitration for the master bus interface for the duration of a DMAC transfer, buffer, or chunk.

**Bus locking:** Software can program a channel to maintain control of the AMBA bus by asserting hmastlock for the duration of a DMAC transfer, buffer, or transaction (single or chunk). Channel locking is asserted for the duration of bus locking at a minimum.

#### 27.4.2 Memory Peripherals

Figure 27-4 on page 431 shows the DMAC transfer hierarchy of the DMAC for a memory peripheral. There is no handshaking interface with the DMAC, and therefore the memory peripheral can never be a flow controller. Once the channel is enabled, the transfer proceeds immediately without waiting for a transaction request. The alternative to not having a transaction-level handshaking interface is to allow the DMAC to attempt AMBA transfers to the peripheral once the channel is enabled. If the peripheral slave cannot accept these AMBA transfers, it inserts wait states onto the bus until it is ready; it is not recommended that more than 16 wait states be inserted onto the bus. By using the handshaking interface, the peripheral can signal to the DMAC that it is ready to transmit/receive data, and then the DMAC can access the peripheral without the peripheral inserting wait states onto the bus.

#### 27.4.3 Handshaking Interface

Handshaking interfaces are used at the transaction level to control the flow of single or chunk transfers. The operation of the handshaking interface is different and depends on whether the peripheral or the DMAC is the flow controller.

The peripheral uses the handshaking interface to indicate to the DMAC that it is ready to transfer/accept data over the AMBA bus. A non-memory peripheral can request a DMAC transfer through the DMAC using one of two handshaking interfaces:

- Hardware handshaking
- Software handshaking

Software selects between the hardware or software handshaking interface on a per-channel basis. Software handshaking is accomplished through memory-mapped registers, while hardware handshaking is accomplished using a dedicated handshaking interface.

##### 27.4.3.1 Software Handshaking

When the slave peripheral requires the DMAC to perform a DMAC transaction, it communicates this request by sending an interrupt to the CPU or interrupt controller.

The interrupt service routine then uses the software registers to initiate and control a DMAC transaction. These software registers are used to implement the software handshaking interface.

The SRC\_H2SEL/DST\_H2SEL bit in the DMAC\_CFGx channel configuration register must be set to zero to enable software handshaking.

When the peripheral is not the flow controller, then the last transaction register DMAC\_LAST is not used, and the values in these registers are ignored.

##### Chunk Transactions

Writing a 1 to the DMAC\_CREQ[2x] register starts a source chunk transaction request, where x is the channel number. Writing a 1 to the DMAC\_CREQ[2x+1] register starts a destination chunk transfer request, where x is the channel number.

Upon completion of the chunk transaction, the hardware clears the DMAC\_CREQ[2x] or DMAC\_CREQ[2x+1].

## *Single Transactions*

Writing a 1 to the DMAC\_SREQ[2x] register starts a source single transaction request, where x is the channel number. Writing a 1 to the DMAC\_SREQ[2x+1] register starts a destination single transfer request, where x is the channel number.

Upon completion of the chunk transaction, the hardware clears the DMAC\_SREQ[x] or DMAC\_SREQ[2x+1].

The software can poll the relevant channel bit in the DMAC\_CREQ[2x]/DMAC\_CREQ[2x+1] and DMAC\_SREQ[x]/DMAC\_SREQ[2x+1] registers. When both are 0, then either the requested chunk or single transaction has completed.

### **27.4.4 DMAC Transfer Types**

A DMAC transfer may consist of single or multi-buffer transfers. On successive buffers of a multi-buffer transfer, the DMAC\_SADDRx/DMAC\_DADDRx registers in the DMAC are reprogrammed using either of the following methods:

- Buffer chaining using linked lists
- Contiguous address between buffers

On successive buffers of a multi-buffer transfer, the DMAC\_CTRLAx and DMAC\_CONTROLBx registers in the DMAC are re-programmed using either of the following methods:

- Buffer chaining using linked lists

When buffer chaining using linked lists is the multi-buffer method of choice, and on successive buffers, the DMAC\_DSCRx register in the DMAC is re-programmed using the following method:

- Buffer chaining using linked lists

A buffer descriptor (LLI) consists of following registers, DMAC\_SADDRx, DMAC\_DADDRx, DMAC\_DSCRx, DMAC\_CTRLAx, DMAC\_CONTROLBx. These registers, along with the DMAC\_CFGx register, are used by the DMAC to set up and describe the buffer transfer.

#### **27.4.4.1 Multi-buffer Transfers**

##### *Buffer Chaining Using Linked Lists*

In this case, the DMAC re-programs the channel registers prior to the start of each buffer by fetching the buffer descriptor for that buffer from system memory. This is known as an LLI update.

DMAC buffer chaining is supported by using a Descriptor Pointer register (DMAC\_DSCRx) that stores the address in memory of the next buffer descriptor. Each buffer descriptor contains the corresponding buffer descriptor (DMAC\_SADDRx, DMAC\_DADDRx, DMAC\_DSCRx, DMAC\_CTRLAx, DMAC\_CONTROLBx).

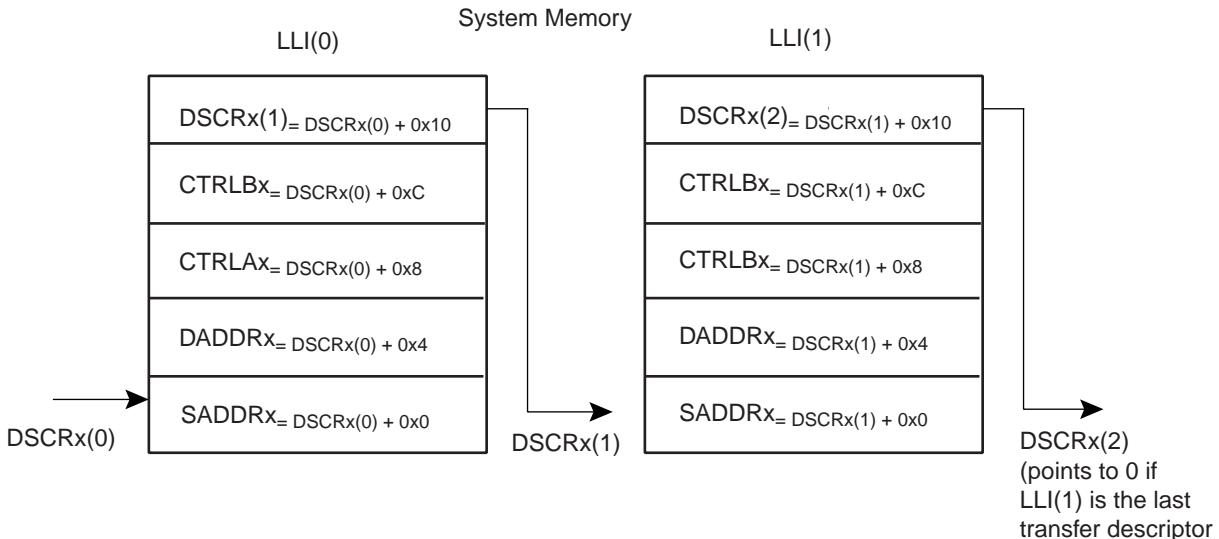
To set up buffer chaining, a sequence of linked lists must be programmed in memory.

The DMAC\_SADDRx, DMAC\_DADDRx, DMAC\_DSCRx, DMAC\_CTRLAx and DMAC\_CONTROLBx registers are fetched from system memory on an LLI update. The updated content of the DMAC\_CTRLAx register is written back to memory on buffer completion. [Figure 27-5 on page 434](#) shows how to use chained linked lists in memory to define multi-buffer transfers using buffer chaining.

The Linked List multi-buffer transfer is initiated by programming DMAC\_DSCRx with DSCRx(0) (LLI(0) base address) different from zero. Other fields and registers are ignored and overwritten when the descriptor is retrieved from memory.

The last transfer descriptor must be written to memory with its next descriptor address set to 0.

**Figure 27-5. Multi Buffer Transfer Using Linked List**



#### 27.4.4.2 Programming DMAC for Multiple Buffer Transfers

**Table 27-3. Multiple Buffers Transfer Management**

Transfer Type	SRC_DSCR	DST_DSCR	BTSIZE	DSCR	SADDR	DADDR	Other Fields
1) Single Buffer or Last buffer of a multiple buffer transfer	–	–	USR	0	USR	USR	USR
2) Multi Buffer transfer with contiguous DADDR	0	1	LLI	USR	LLI	CONT	LLI
3) Multi Buffer transfer with contiguous SADDR	1	0	LLI	USR	CONT	LLI	LLI
4) Multi Buffer transfer with LLI support	0	0	LLI	USR	LLI	LLI	LLI

Notes:

1. USR means that the register field is manually programmed by the user.
2. CONT means that address are contiguous.
3. Channel stalled is true if the relevant BTC interrupt is not masked.
4. LLI means that the register field is updated with the content of the linked list item.

#### Contiguous Address Between Buffers

In this case, the address between successive buffers is selected to be a continuation from the end of the previous buffer. Enabling the source or destination address to be contiguous between buffers is a function of DMAC\_CTRLAx.SRC\_DSCR and DMAC\_CTRLAx.DST\_DSCR registers.

#### Suspension of Transfers Between Buffers

At the end of every buffer transfer, an end of buffer interrupt is asserted if:

- The channel buffer interrupt is unmasked, DMAC\_EBCIMR.BTCx = '1', where x is the channel number.

Note: The Buffer Transfer Completed Interrupt is generated at the completion of the buffer transfer to the destination.

At the end of a chain of multiple buffers, an end of linked list interrupt is asserted if:

- The channel end of the Chained Buffer Transfer Completed Interrupt is unmasked, DMAC\_EBCIMR.CBTCx = '1', when n is the channel number.

#### 27.4.4.3 Ending Multi-buffer Transfers

All multi-buffer transfers must end as shown in Row 1 of [Table 27-3 on page 434](#). At the end of every buffer transfer, the DMAC samples the row number, and if the DMAC is in Row 1 state, then the previous buffer transferred was the last buffer and the DMAC transfer is terminated.

For rows 2, 3, 4, 5, and 6 (DMAC\_CRTL<sub>Bx</sub>.AUTO cleared), the user must set up the last buffer descriptor in memory so that LLI.DMAC\_DSCR<sub>x</sub> is set to 0.

#### 27.4.5 Programming a Channel

Four registers, the DMAC\_DSCR<sub>x</sub>, the DMAC\_CTRLA<sub>x</sub>, the DMAC\_CTRLB<sub>x</sub> and DMAC\_CFG<sub>x</sub>, need to be programmed to set up whether single or multi-buffer transfers take place, and which type of multi-buffer transfer is used. The different transfer types are shown in [Table 27-3 on page 434](#).

The “BTSIZE, SADDR and DADDR” columns indicate where the values of DMAC\_SAR<sub>x</sub>, DMAC\_DAR<sub>x</sub>, DMAC\_CTL<sub>x</sub>, and DMAC\_LL<sub>Px</sub> are obtained for the next buffer transfer when multi-buffer DMAC transfers are enabled.

##### 27.4.5.1 Programming Examples

###### *Single-buffer Transfer (Row 1)*

1. Read the Channel Handler Status Register DMAC\_CHSR.ENA<sub>x</sub> Field to choose a free (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMAC transfer by reading the interrupt status register, DMAC\_EBCISR.
3. Program the following channel registers:
  1. Write the starting source address in the DMAC\_SADDR<sub>x</sub> register for channel x.
  2. Write the starting destination address in the DMAC\_DADDR<sub>x</sub> register for channel x.
  3. Write the next descriptor address in the DMA\_DSCR<sub>x</sub> register for channel x with 0x0.
  4. Program DMAC\_CTRLA<sub>x</sub>, DMAC\_CTRLB<sub>x</sub> and DMAC\_CFG<sub>x</sub> according to Row 1 as shown in [Table 27-3 on page 434](#).
  5. Write the control information for the DMAC transfer in the DMAC\_CTRLA<sub>x</sub> and DMAC\_CTRLB<sub>x</sub> registers for channel x. For example, in the register, you can program the following:
    - i. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the FC of the DMAC\_CTRLB<sub>x</sub> register.
    - ii. Set up the transfer characteristics, such as:
      - Transfer width for the source in the SRC\_WIDTH field.
      - Transfer width for the destination in the DST\_WIDTH field.
      - Incrementing/decrementing or fixed address for source in SRC\_INC field.
      - Incrementing/decrementing or fixed address for destination in DST\_INC field.
  6. Write the channel configuration information into the DMAC\_CFG<sub>x</sub> register for channel x.
    - i. Designate the handshaking interface type (hardware or software) for the source and destination peripherals. This is not required for memory. This step requires programming the SRC\_H2SEL/DST\_H2SEL bits, respectively. Writing a ‘1’ activates the hardware handshaking interface to handle source/destination requests. Writing a ‘0’ activates the software handshaking interface to handle source/destination requests.
    - ii. If the hardware handshaking interface is activated for the source or destination peripheral, assign a handshaking interface to the source and destination peripheral. This requires programming the SRC\_PER and DST\_PER bits, respectively.
4. After the DMAC selected channel has been programmed, enable the channel by writing a ‘1’ to the DMAC\_CHER.ENA<sub>x</sub> bit, where x is the channel number. Make sure that bit 0 of DMAC\_EN.ENABLE register is enabled.
5. Source and destination request single and chunk DMAC transactions to transfer the buffer of data (assuming non-memory peripherals). The DMAC acknowledges at the completion of every transaction (chunk and single) in the buffer and carries out the buffer transfer.

- Once the transfer completes, the hardware sets the interrupts and disables the channel. At this time, you can either respond to the Buffer Transfer Completed Interrupt or Chained Buffer Transfer Completed Interrupt, or poll for the Channel Handler Status Register (DMAC\_CHSR.ENAx) bit until it is cleared by hardware, to detect when the transfer is complete.

*Multi-buffer Transfer with Linked List for Source and Linked List for Destination (Row 4)*

- Read the Channel Handler Status register to choose a free (disabled) channel.
- Set up the chain of Linked List Items (otherwise known as buffer descriptors) in memory. Write the control information in the LLI.DMAC\_CTRLAx and LLI.DMAC\_CTRLBx registers location of the buffer descriptor for each LLI in memory (see [Figure 27-6 on page 437](#)) for channel x. For example, in the register, you can program the following:
  - Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the FC of the DMAC\_CTRLBx register.
  - Set up the transfer characteristics, such as:
    - i. Transfer width for the source in the SRC\_WIDTH field.
    - ii. Transfer width for the destination in the DST\_WIDTH field.
    - v. Incrementing/decrementing or fixed address for source in SRC\_INCR field.
    - vi. Incrementing/decrementing or fixed address for destination DST\_INCR field.
- Write the channel configuration information into the DMAC\_CFGx register for channel x.
  - Designate the handshaking interface type (hardware or software) for the source and destination peripherals. This is not required for memory. This step requires programming the SRC\_H2SEL/DST\_H2SEL bits, respectively. Writing a '1' activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a '0' activates the software handshaking interface to handle source/destination requests.
  - If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral. This requires programming the SRC\_PER and DST\_PER bits, respectively.
- Make sure that the LLI.DMAC\_CTRLBx register locations of all LLI entries in memory (except the last) are set as shown in Row 4 of [Table 27-3 on page 434](#). The LLI.DMAC\_CTRLBx register of the last Linked List Item must be set as described in Row 1 of [Table 27-3 on page 434](#). [Figure 27-5 on page 434](#) shows a Linked List example with two list items.
- Make sure that the LLI.DMAC\_DSCRx register locations of all LLI entries in memory (except the last) are non-zero and point to the base address of the next Linked List Item.
- Make sure that the LLI.DMAC\_SADDRx/LLI.DMAC\_DADDRx register locations of all LLI entries in memory point to the start source/destination buffer address preceding that LLI fetch.
- Make sure that the LLI.DMAC\_CTRLAx.DONE field of the LLI.DMAC\_CTRLAx register locations of all LLI entries in memory are cleared.
- Clear any pending interrupts on the channel from the previous DMAC transfer by reading the status register: DMAC\_EBCISR.
- Program the DMAC\_CTRLBx, DMAC\_CFGx registers according to Row 4 as shown in [Table 27-3 on page 434](#).
- Program the DMAC\_DSCRx register with DMAC\_DSCRx(0), the pointer to the first Linked List item.
- Finally, enable the channel by writing a '1' to the DMAC\_CHER.ENAx bit, where x is the channel number. The transfer is performed.
- The DMAC fetches the first LLI from the location pointed to by DMAC\_DSCRx(0).

Note: The LLI.DMAC\_SADDRx, LLI.DMAC\_DADDRx, LLI.DMAC\_DSCRx, LLI.DMAC\_CTRLAx and LLI.DMAC\_CTRLBx registers are fetched. The DMAC automatically reprograms the DMAC\_SADDRx, DMAC\_DADDRx, DMAC\_DSCRx, DMAC\_CTRLBx and DMAC\_CTRLAx channel registers from the DMAC\_DSCRx(0).

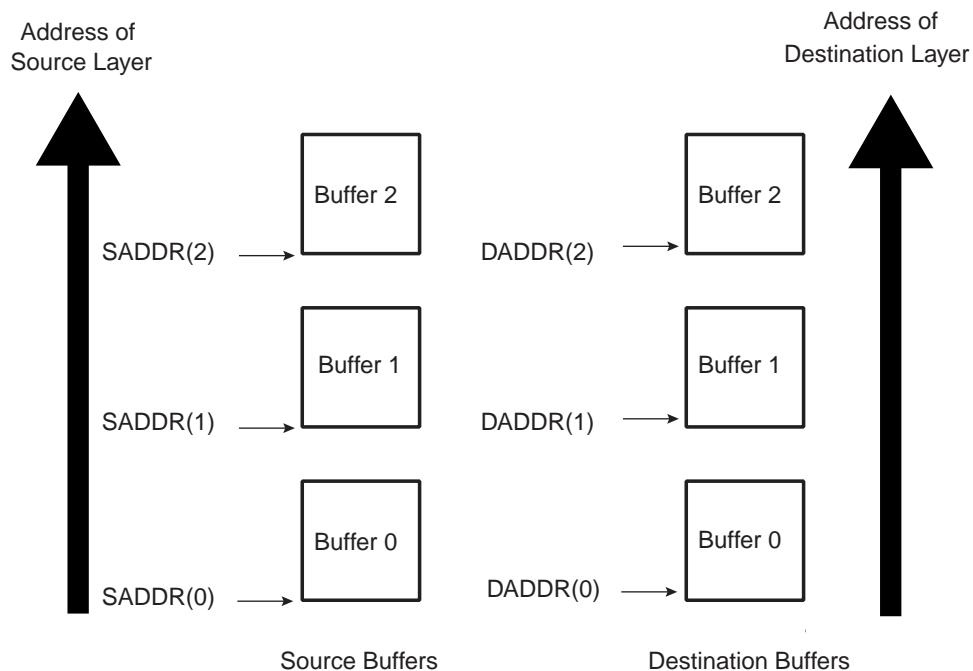
- Source and destination request single and chunk DMAC transactions to transfer the buffer of data (assuming non-memory peripheral). The DMAC acknowledges at the completion of every transaction (chunk and single) in the buffer and carries out the buffer transfer.

14. Once the buffer of data is transferred, the DMAC\_CTRLAx register is written out to system memory at the same location and on the same layer where it was originally fetched, that is, the location of the DMAC\_CTRLAx register of the linked list item fetched prior to the start of the buffer transfer. Only DMAC\_CTRLAx register is written out because only the DMAC\_CTRLAx.BTSIZE and DMAC\_CTRLAx.DONE bits have been updated by DMAC hardware. Additionally, the DMAC\_CTRLAx.DONE bit is asserted when the buffer transfer has completed.

Note: Do not poll the DMAC\_CTRLAx.DONE bit in the DMAC memory map. Instead, poll the LLI.DMAC\_CTRLAx.DONE bit in the LLI for that buffer. If the poll LLI.DMAC\_CTRLAx.DONE bit is asserted, then this buffer transfer has completed. This LLI.DMAC\_CTRLAx.DONE bit was cleared at the start of the transfer.

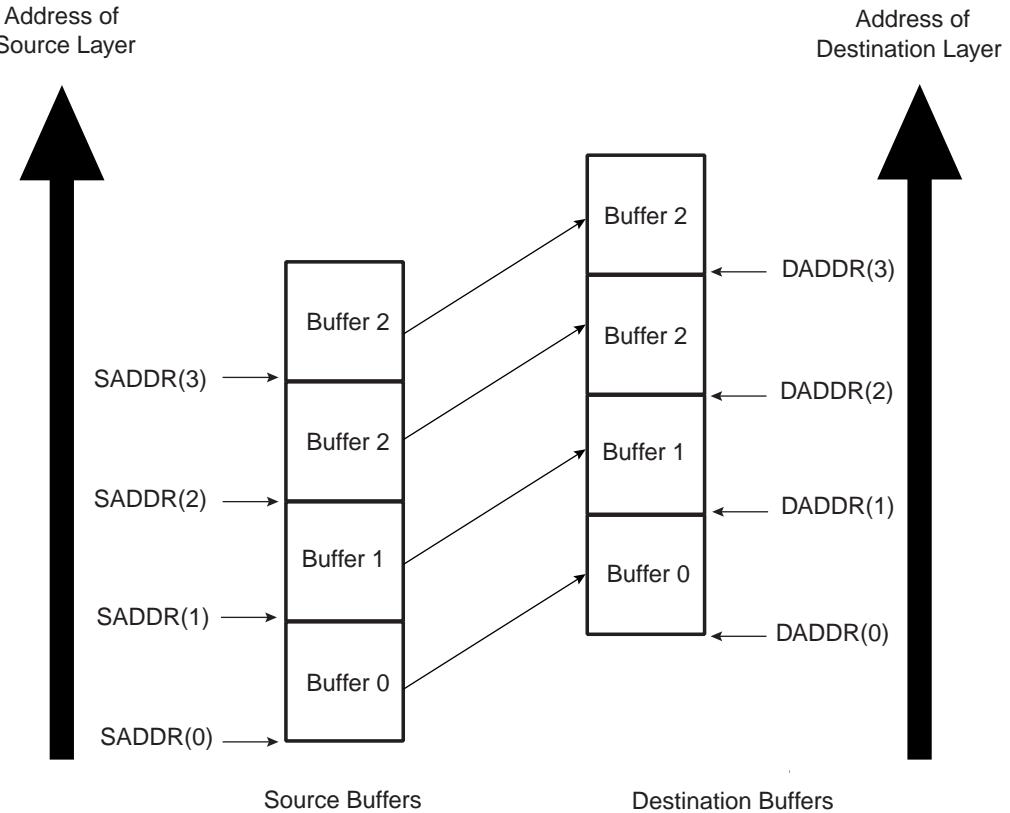
15. The DMAC does not wait for the buffer interrupt to be cleared, but continues fetching the next LLI from the memory location pointed to by current DMAC\_DSCRx register and automatically reprograms the DMAC\_SADDRx, DMAC\_DADDRx, DMAC\_DSCRx, DMAC\_CTRLAx and DMAC\_CTRLBx channel registers. The DMAC transfer continues until the DMAC determines that the DMAC\_CTRLBx and DMAC\_DSCRx registers at the end of a buffer transfer match described in Row 1 of [Table 27-3 on page 434](#). The DMAC then knows that the previous buffer transferred was the last buffer in the DMAC transfer. The DMAC transfer might look like that shown in [Figure 27-6 on page 437](#).

**Figure 27-6. Multi-buffer with Linked List Address for Source and Destination**



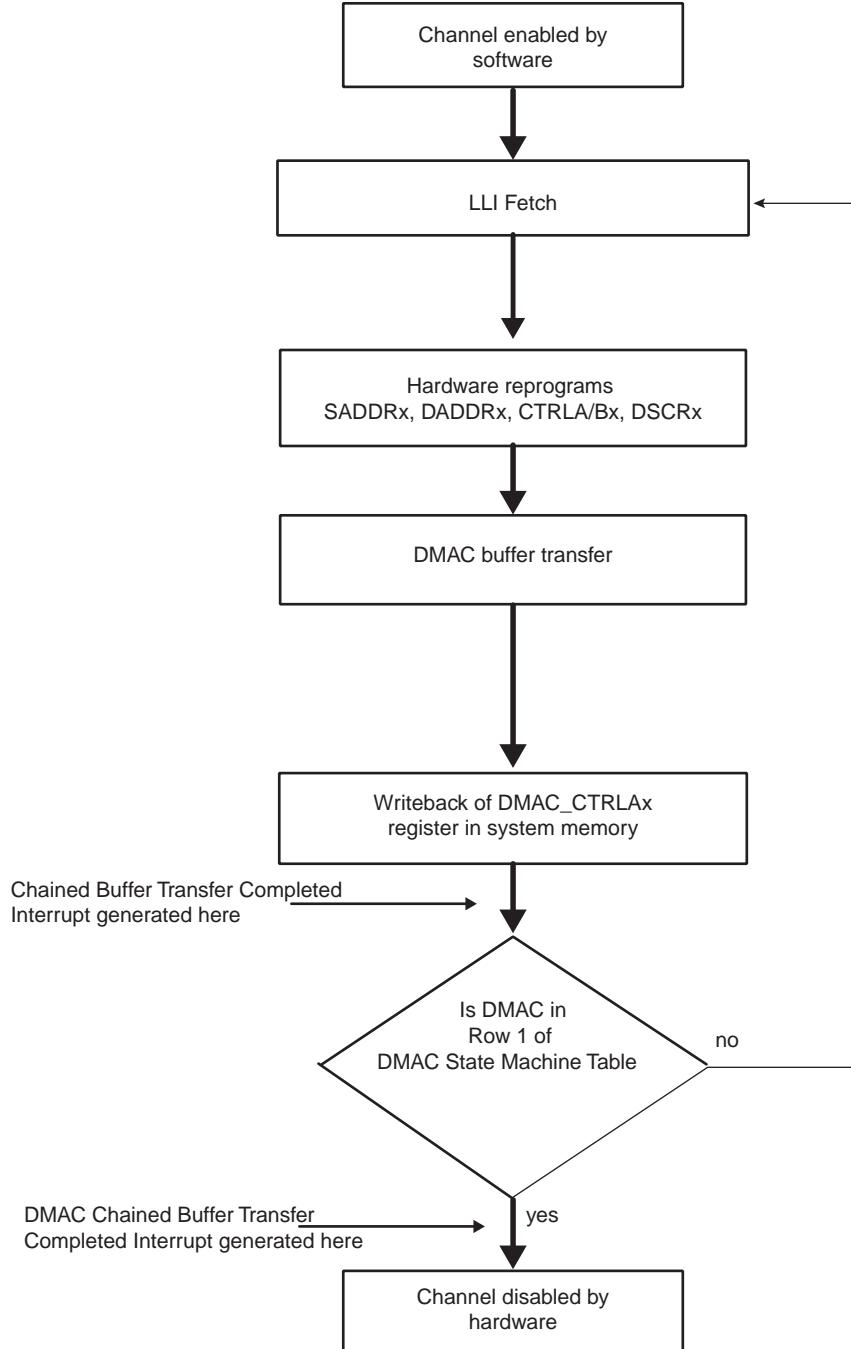
If the user needs to execute a DMAC transfer where the source and destination address are contiguous but the amount of data to be transferred is greater than the maximum buffer size DMAC\_CTRLAx.BTSIZE, then this can be achieved using the type of multi-buffer transfer as shown in [Figure 27-7 on page 438](#).

**Figure 27-7. Multi-buffer with Linked Address for Source and Destination Buffers are Contiguous**



The DMAC transfer flow is shown in [Figure 27-8 on page 439](#).

**Figure 27-8. DMAC Transfer Flow for Source and Destination Linked List Address**



## *Multi-buffer DMAC Transfer with Linked List for Source and Contiguous Destination Address (Row 2)*

1. Read the Channel Handler Status register to choose a free (disabled) channel.
2. Set up the linked list in memory. Write the control information in the LLI.DMAC\_CTRLAx and LLI.DMAC\_CTRLBx register location of the buffer descriptor for each LLI in memory for channel x. For example, in the register, you can program the following:
  1. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the FC of the DMAC\_CTRLBx register.
  2. Set up the transfer characteristics, such as:
    - i. Transfer width for the source in the SRC\_WIDTH field.
    - ii. Transfer width for the destination in the DST\_WIDTH field.
    - v. Incrementing/decrementing or fixed address for source in SRC\_INCR field.
    - vi. Incrementing/decrementing or fixed address for destination DST\_INCR field.
3. Write the starting destination address in the DMAC\_DADDRx register for channel x.

Note: The values in the LLI.DMAC\_DADDRx register location of each Linked List Item (LLI) in memory, although fetched during an LLI fetch, are not used.

4. Write the channel configuration information into the DMAC\_CFGx register for channel x.
  1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals. This is not required for memory. This step requires programming the SRC\_H2SEL/DST\_H2SEL bits, respectively. Writing a '1' activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a '0' activates the software handshaking interface to handle source/destination requests.
  2. If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripherals. This requires programming the SRC\_PER and DST\_PER bits, respectively.
5. Make sure that all LLI.DMAC\_CTRLBx register locations of the LLI (except the last) are set as shown in Row 2 of [Table 27-3 on page 434](#), while the LLI.DMAC\_CTRLBx register of the last Linked List item must be set as described in Row 1 of [Table 27-3 on page 434](#). [Figure 27-5 on page 434](#) shows a Linked List example with two list items.
6. Make sure that the LLI.DMAC\_DSCRx register locations of all LLIs in memory (except the last) are non-zero and point to the next Linked List Item.
7. Make sure that the LLI.DMAC\_SADDRx register locations of all LLIs in memory point to the start source buffer address proceeding that LLI fetch.
8. Make sure that the LLI.DMAC\_CTRLAx.DONE field of the LLI.DMAC\_CTRLAx register locations of all LLIs in memory is cleared.
9. Clear any pending interrupts on the channel from the previous DMAC transfer by reading the interrupt status register.
10. Program the DMAC\_CTRLAx, DMAC\_CTRLBx and DMAC\_CFGx registers according to Row 2 as shown in [Table 27-3 on page 434](#).
11. Program the DMAC\_DSCRx register with DMAC\_DSCRx(0), the pointer to the first Linked List item.
12. Finally, enable the channel by writing a '1' to the DMAC\_CHER.ENAx bit. The transfer is performed. Make sure that bit 0 of the DMAC\_EN register is enabled.
13. The DMAC fetches the first LLI from the location pointed to by DMAC\_DSCRx(0).

Note: The LLI.DMAC\_SADDRx, LLI.DMAC\_DADDRx, LLI.DMAC\_DSCRx and LLI.DMAC\_CTRLA/Bx registers are fetched. The LLI.DMAC\_DADDRx register location of the LLI, although fetched, is not used. The DMAC\_DADDRx register in the DMAC remains unchanged.

14. Source and destination requests single and chunk DMAC transactions to transfer the buffer of data (assuming non-memory peripherals). The DMAC acknowledges at the completion of every transaction (chunk and single) in the buffer and carries out the buffer transfer.

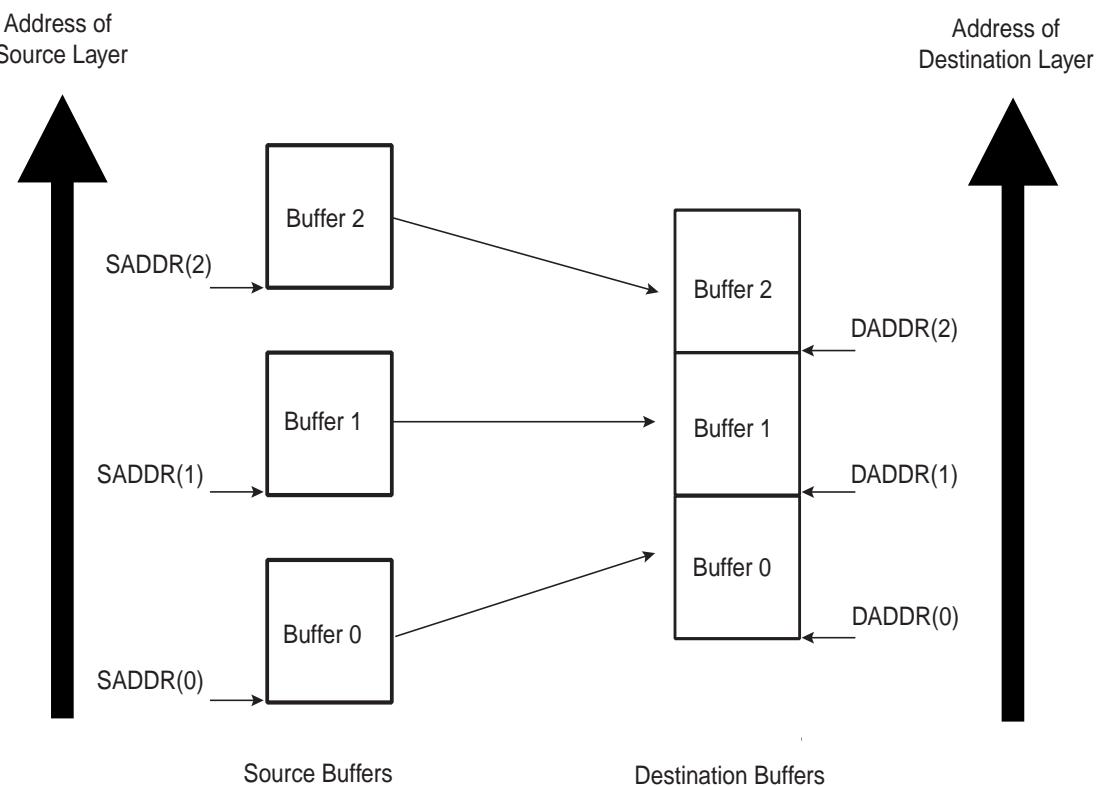
15. Once the buffer of data is transferred, the DMAC\_CTRLAx register is written out to the system memory at the same location and on the same layer (DMAC\_DSCRx.DSCR\_IF) where it was originally fetched, that is, the location of the DMAC\_CTRLAx register of the linked list item fetched prior to the start of the buffer transfer. Only DMAC\_CTRLAx register is written out because only the DMAC\_CTRLAx.BTSIZE and DMAC\_CTRLAx.DONE fields have been updated by DMAC hardware. Additionally, the DMAC\_CTRLAx.DONE bit is asserted when the buffer transfer has completed.

Note: Do not poll the DMAC\_CTRLAx.DONE bit in the DMAC memory map. Instead, poll the LLI.DMAC\_CTRLAx.DONE bit in the LLI for that buffer. If the poll LLI.DMAC\_CTRLAx.DONE bit is asserted, then this buffer transfer has completed. This LLI.DMAC\_CTRLAx.DONE bit was cleared at the start of the transfer.

16. The DMAC does not wait for the buffer interrupt to be cleared, but continues and fetches the next LLI from the memory location pointed to by the current DMAC\_DSCRx register, then automatically reprograms the DMAC\_SADDRx, DMAC\_CTRLAx, DMAC\_CTRLBx and DMAC\_DSCRx channel registers. The DMAC\_DADDRx register is left unchanged. The DMAC transfer continues until the DMAC samples the DMAC\_CTRLAx, DMAC\_CTRLBx and DMAC\_DSCRx registers at the end of a buffer transfer match that described in Row 1 of [Table 27-3 on page 434](#). The DMAC then knows that the previous buffer transferred was the last buffer in the DMAC transfer.

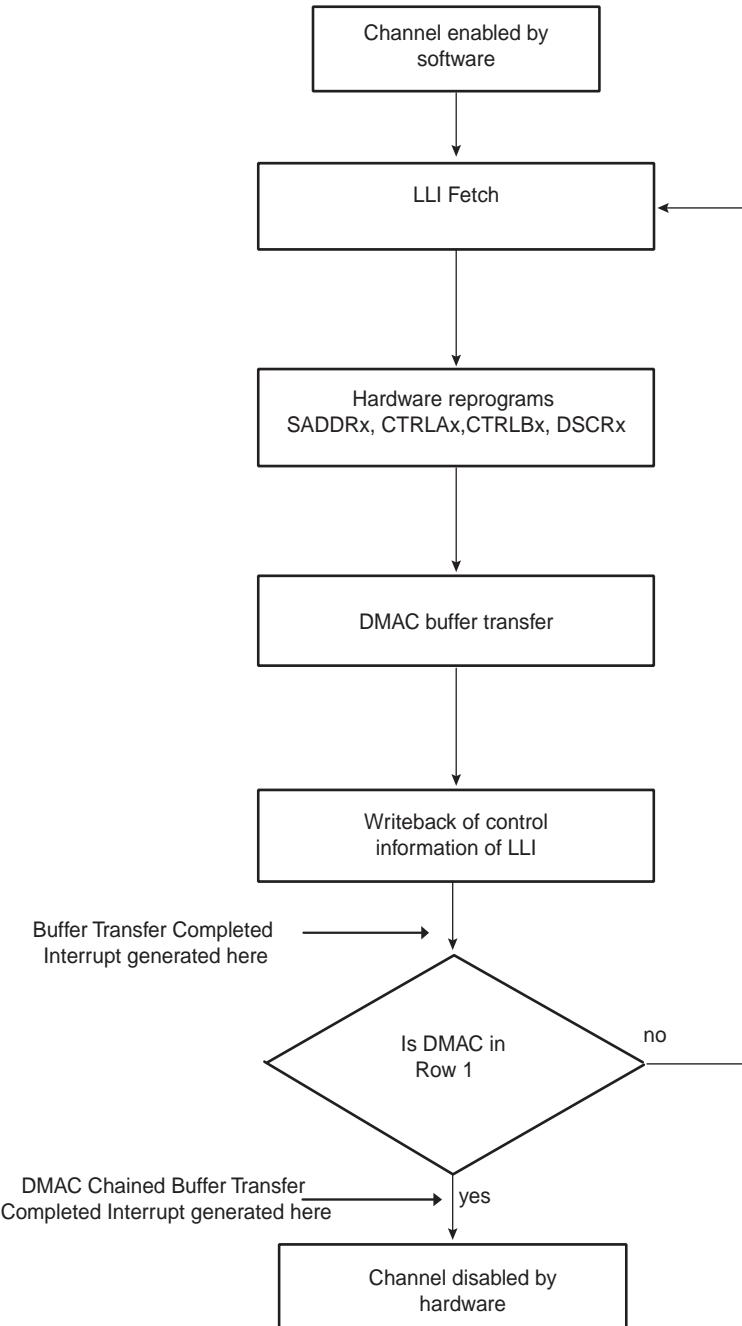
The DMAC transfer might look like that shown in [Figure 27-9 on page 441](#). Note that the destination address is decrementing.

**Figure 27-9. DMAC Transfer with Linked List Source Address and Contiguous Destination Address**



The DMAC transfer flow is shown in [Figure 27-10 on page 442](#).

**Figure 27-10. DMAC Transfer Flow for Linked List Source Address and Contiguous Destination Address**



## 27.4.6 Disabling a Channel Prior to Transfer Completion

Under normal operation, the software enables a channel by writing a '1' to the Channel Handler Enable Register, DMAC\_CHER.ENAx, and the hardware disables a channel on transfer completion by clearing the DMAC\_CHSR.ENAx register bit.

The recommended way for software to disable a channel without losing data is to use the SUSPx bit in conjunction with the EMPTx bit in the Channel Handler Status Register.

1. If the software wishes to disable a channel n prior to the DMAC transfer completion, then it can set the DMAC\_CHER.SUSPx bit to tell the DMAC to halt all transfers from the source peripheral. Therefore, the channel FIFO receives no new data.
2. The software can now poll the DMAC\_CHSR.EMPTx bit until it indicates that the channel n FIFO is empty, where n is the channel number.
3. The DMAC\_CHER.ENAx bit can then be cleared by software once the channel n FIFO is empty, where n is the channel number.

When DMAC\_CTRLAx.SRC\_WIDTH is less than DMAC\_CTRLAx.DST\_WIDTH and the DMAC\_CHSRx.SUSPx bit is high, the DMAC\_CHSRx.EMPTx is asserted once the contents of the FIFO does not permit a single word of DMAC\_CTRLAx.DST\_WIDTH to be formed. However, there may still be data in the channel FIFO but not enough to form a single transfer of DMAC\_CTLx.DST\_WIDTH width. In this configuration, once the channel is disabled, the remaining data in the channel FIFO are not transferred to the destination peripheral. It is permitted to remove the channel from the suspension state by writing a '1' to the DMAC\_CHER.RESx field register. The DMAC transfer completes in the normal manner. n defines the channel number.

Note: If a channel is disabled by software, an active single or chunk transaction is not guaranteed to receive an acknowledgement.

### 27.4.6.1 Abnormal Transfer Termination

A DMAC transfer may be terminated abruptly by software by clearing the channel enable bit, DMAC\_CHDR.ENAx, where x is the channel number. This does not mean that the channel is disabled immediately after the DMAC\_CHSR.ENAx bit is cleared over the APB interface. Consider this as a request to disable the channel. The DMAC\_CHSR.ENAx must be polled and then it must be confirmed that the channel is disabled by reading back 0.

The software may terminate all channels abruptly by clearing the global enable bit in the DMAC Configuration Register (DMAC\_EN.ENABLE bit). Again, this does not mean that all channels are disabled immediately after the DMAC\_EN.ENABLE is cleared over the APB slave interface. Consider this as a request to disable all channels. The DMAC\_CHSR.ENABLE must be polled and then it must be confirmed that all channels are disabled by reading back '0'.

Note: If the channel enable bit is cleared while there is data in the channel FIFO, this data is not sent to the destination peripheral and is not present when the channel is re-enabled. For read sensitive source peripherals, such as a source FIFO, this data is therefore lost. When the source is not a read sensitive device (i.e., memory), disabling a channel without waiting for the channel FIFO to empty may be acceptable as the data is available from the source peripheral upon request and is not lost.

Note: If a channel is disabled by software, an active single or chunk transaction is not guaranteed to receive an acknowledgement.

## 27.5 DMAC Software Requirements

- There must not be any write operation to Channel registers in an active channel after the channel enable is made HIGH. If any channel parameters must be reprogrammed, this can only be done after disabling the DMAC channel.
- You must program the DMAC\_SADDRx and DMAC\_DADDRx channel registers with a byte, half-word and word aligned address depending on the source width and destination width.
- After the software disables a channel by writing into the channel disable register, it must re-enable the channel only after it has polled a 0 in the corresponding channel enable status register. This is because the current AHB Burst must terminate properly.
- If you program the BTSIZE field in the DMAC\_CTRLA as zero, and the DMAC has been defined as the flow controller, then the channel is automatically disabled.
- When hardware handshaking interface protocol is fully implemented, a peripheral is expected to deassert any sreq or breq signals on receiving the ack signal irrespective of the request the ack was asserted in response to.
- Multiple Transfers involving the same peripheral must not be programmed and enabled on different channels, unless this peripheral integrates several hardware handshaking interfaces.
- When a Peripheral has been defined as the flow controller, the targeted DMAC Channel must be enabled before the Peripheral. If you do not ensure this and the First DMAC request is also the last transfer, the DMAC Channel might miss a Last Transfer Flag.

## 27.6 Write Protection Registers

To prevent any single software error that may corrupt the DMAC behavior, the DMAC address space can be write-protected by setting the WPEN bit in the “[DMAC Write Protect Mode Register](#)” (DMAC\_WPMR).

If a write access to anywhere in the DMAC address space is detected, then the WPVS flag in the DMAC Write Protect Status Register (MCI\_WPSR) is set, and the WPVSR field indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the DMAC Write Protect Mode Register (DMAC\_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- “[DMAC Global Configuration Register](#)” on page 447
- “[DMAC Enable Register](#)” on page 448
- “[DMAC Channel x \[x = 0..3\] Source Address Register](#)” on page 459
- “[DMAC Channel x \[x = 0..3\] Destination Address Register](#)” on page 460
- “[DMAC Channel x \[x = 0..3\] Descriptor Address Register](#)” on page 461
- “[DMAC Channel x \[x = 0..3\] Control A Register](#)” on page 462
- “[DMAC Channel x \[x = 0..3\] Control B Register](#)” on page 463
- “[DMAC Channel x \[x = 0..3\] Configuration Register](#)” on page 465

## 27.7 DMA Controller (DMAC) User Interface

Table 27-4. Register Mapping

Offset	Register	Name	Access	Reset
0x000	DMAC Global Configuration Register	DMAC_GCFG	Read-write	0x10
0x004	DMAC Enable Register	DMAC_EN	Read-write	0x0
0x008	DMAC Software Single Request Register	DMAC_SREQ	Read-write	0x0
0x00C	DMAC Software Chunk Transfer Request Register	DMAC_CREQ	Read-write	0x0
0x010	DMAC Software Last Transfer Flag Register	DMAC_LAST	Read-write	0x0
0x014	Reserved			
0x018	DMAC Error, Chained Buffer Transfer Completed Interrupt and Buffer Transfer Completed Interrupt Enable register.	DMAC_EBCIER	Write-only	–
0x01C	DMAC Error, Chained Buffer Transfer Completed Interrupt and Buffer Transfer Completed Interrupt Disable register.	DMAC_EBCIDR	Write-only	–
0x020	DMAC Error, Chained Buffer Transfer Completed Interrupt and Buffer transfer completed Mask Register.	DMAC_EBCIMR	Read-only	0x0
0x024	DMAC Error, Chained Buffer Transfer Completed Interrupt and Buffer transfer completed Status Register.	DMAC_EBCISR	Read-only	0x0
0x028	DMAC Channel Handler Enable Register	DMAC_CHER	Write-only	–
0x02C	DMAC Channel Handler Disable Register	DMAC_CHDR	Write-only	–
0x030	DMAC Channel Handler Status Register	DMAC_CHSR	Read-only	0x00FF0000
0x034	Reserved	–	–	–
0x038	Reserved	–	–	–
0x03C+ch_num*(0x28)+(0x0)	DMAC Channel Source Address Register	DMAC_SADDR	Read-write	0x0
0x03C+ch_num*(0x28)+(0x4)	DMAC Channel Destination Address Register	DMAC_DADDR	Read-write	0x0
0x03C+ch_num*(0x28)+(0x8)	DMAC Channel Descriptor Address Register	DMAC_DSCR	Read-write	0x0
0x03C+ch_num*(0x28)+(0xC)	DMAC Channel Control A Register	DMAC_CTRLA	Read-write	0x0
0x03C+ch_num*(0x28)+(0x10)	DMAC Channel Control B Register	DMAC_CTRLB	Read-write	0x0
0x03C+ch_num*(0x28)+(0x14)	DMAC Channel Configuration Register	DMAC_CFG	Read-write	0x01000000
0x03C+ch_num*(0x28)+(0x18)	Reserved	–	–	–
0x03C+ch_num*(0x28)+(0x1C)	Reserved	–	–	–
0x03C+ch_num*(0x28)+(0x20)	Reserved	–	–	–
0x03C+ch_num*(0x28)+(0x24)	Reserved	–	–	–
0x1E4	DMAC Write Protect Mode Register	DMAC_WPMR	Read-write	0x0
0x1E8	DMAC Write Protect Status Register	DMAC_WPSR	Read-only	0x0
0x01EC- 0x1FC	Reserved	–	–	–

### 27.7.1 DMAC Global Configuration Register

**Name:** DMAC\_GCFG

**Address:** 0x400C0000

**Access:** Read-write

**Reset:** 0x00000010

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
-	-	-	ARB_CFG	-	-	-	-

**Note:** Bit fields 0, 1, 2, 3, have a default value of 0. This should not be changed.

This register can only be written if the WPEN bit is cleared in “[DMAC Write Protect Mode Register](#)” .

- **ARB\_CFG: Arbiter Configuration**

0 (FIXED): Fixed priority arbiter.

1 (ROUND\_ROBIN): Modified round robin arbiter.

### 27.7.2 DMAC Enable Register

**Name:** DMAC\_EN

**Address:** 0x400C0004

**Access:** Read-write

**Reset:** 0x00000000

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
-	-	-	-	-	-	-	ENABLE

This register can only be written if the WPEN bit is cleared in “DMAC Write Protect Mode Register” .

- **ENABLE: General Enable of DMA**

0: DMA Controller is disabled.

1: DMA Controller is enabled.

### 27.7.3 DMAC Software Single Request Register

**Name:** DMAC\_SREQ

**Address:** 0x400C0008

**Access:** Read-write

**Reset:** 0x00000000

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
DSREQ3	SSREQ3	DSREQ2	SSREQ2	DSREQ1	SSREQ1	DSREQ0	SSREQ0

- **DSREQx: Destination Request**

Request a destination single transfer on channel i.

- **SSREQx: Source Request**

Request a source single transfer on channel i.

#### 27.7.4 DMAC Software Chunk Transfer Request Register

**Name:** DMAC\_CREQ

**Address:** 0x400C000C

**Access:** Read-write

**Reset:** 0x00000000

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
DCREQ3	SCREQ3	DCREQ2	SCREQ2	DCREQ1	SCREQ1	DCREQ0	SCREQ0

- **DCREQx: Destination Chunk Request**

Request a destination chunk transfer on channel i.

- **SCREQx: Source Chunk Request**

Request a source chunk transfer on channel i.

### 27.7.5 DMAC Software Last Transfer Flag Register

**Name:** DMAC\_LAST

**Address:** 0x400C0010

**Access:** Read-write

**Reset:** 0x00000000

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
DLAST3	SLAST3	DLAST2	SLAST2	DLAST1	SLAST1	DLAST0	SLAST0

- **DLASTx: Destination Last**

Writing one to DLASTx prior to writing one to DSREQx or DCREQx indicates that this destination request is the last transfer of the buffer.

- **SLASTx: Source Last**

Writing one to SLASTx prior to writing one to SSREQx or SCREQx indicates that this source request is the last transfer of the buffer.

### 27.7.6 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Enable Register

**Name:** DMAC\_EBCIER

**Address:** 0x400C0018

**Access:** Write-only

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
-	-	-	-	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
-	-	-	-	BTC3	BTC2	BTC1	BTC0

- **BTC<sub>x</sub>: Buffer Transfer Completed [3:0]**

Buffer Transfer Completed Interrupt Enable Register. Set the relevant bit in the BTC field to enable the interrupt for channel i.

- **CBTC<sub>x</sub>: Chained Buffer Transfer Completed [3:0]**

Chained Buffer Transfer Completed Interrupt Enable Register. Set the relevant bit in the CBTC field to enable the interrupt for channel i.

- **ERR<sub>x</sub>: Access Error [3:0]**

Access Error Interrupt Enable Register. Set the relevant bit in the ERR field to enable the interrupt for channel i.

### 27.7.7 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Disable Register

**Name:** DMAC\_EBCIDR

**Address:** 0x400C001C

**Access:** Write-only

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
-	-	-	-	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
-	-	-	-	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [3:0]**

Buffer transfer completed Disable Interrupt Register. When set, a bit of the BTC field disables the interrupt from the relevant DMAC channel.

- **CBTCx: Chained Buffer Transfer Completed [3:0]**

Chained Buffer transfer completed Disable Register. When set, a bit of the CBTC field disables the interrupt from the relevant DMAC channel.

- **ERRx: Access Error [3:0]**

Access Error Interrupt Disable Register. When set, a bit of the ERR field disables the interrupt from the relevant DMAC channel.

### 27.7.8 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Mask Register

**Name:** DMAC\_EBCIMR

**Address:** 0x400C0020

**Access:** Read-only

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
-	-	-	-	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
-	-	-	-	BTC3	BTC2	BTC1	BTC0

- **BTC<sub>x</sub>: Buffer Transfer Completed [3:0]**

0: Buffer Transfer Completed Interrupt is disabled for channel i.

1: Buffer Transfer Completed Interrupt is enabled for channel i.

- **CBTC<sub>x</sub>: Chained Buffer Transfer Completed [3:0]**

0: Chained Buffer Transfer interrupt is disabled for channel i.

1: Chained Buffer Transfer interrupt is enabled for channel i.

- **ERR<sub>x</sub>: Access Error [3:0]**

0: Transfer Error Interrupt is disabled for channel i.

1: Transfer Error Interrupt is enabled for channel i.

### 27.7.9 DMAC Error, Buffer Transfer and Chained Buffer Transfer Status Register

**Name:** DMAC\_EBCISR

**Address:** 0x400C0024

**Access:** Read-only

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
-	-	-	-	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
-	-	-	-	BTC3	BTC2	BTC1	BTC0

- **BTC<sub>x</sub>: Buffer Transfer Completed [3:0]**

When BTC[*i*] is set, Channel *i* buffer transfer has terminated.

- **CBTC<sub>x</sub>: Chained Buffer Transfer Completed [3:0]**

When CBTC[*i*] is set, Channel *i* Chained buffer has terminated. LLI Fetch operation is disabled.

- **ERR<sub>x</sub>: Access Error [3:0]**

When ERR[*i*] is set, Channel *i* has detected an AHB Read or Write Error Access.

### 27.7.10 DMAC Channel Handler Enable Register

**Name:** DMAC\_CHER

**Address:** 0x400C0028

**Access:** Write-only

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	KEEP3	KEEP2	KEEP1	KEEP0
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	SUSP3	SUSP2	SUSP1	SUSP0
7	6	5	4	3	2	1	0
-	-	-	-	ENA3	ENA2	ENA1	ENA0

- **ENAx: Enable [3:0]**

When set, a bit of the ENA field enables the relevant channel.

- **SUSPx: Suspend [3:0]**

When set, a bit of the SUSP field freezes the relevant channel and its current context.

- **KEEPx: Keep on [3:0]**

When set, a bit of the KEEP field resumes the current channel from an automatic stall state.

### 27.7.11 DMAC Channel Handler Disable Register

**Name:** DMAC\_CHDR

**Address:** 0x400C002C

**Access:** Write-only

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	RES3	RES2	RES1	RES0
7	6	5	4	3	2	1	0
-	-	-	-	DIS3	DIS2	DIS1	DIS0

- **DISx: Disable [3:0]**

Write one to this field to disable the relevant DMAC Channel. The content of the FIFO is lost and the current AHB access is terminated. Software must poll DIS[3:0] field in the DMAC\_CHSR register to be sure that the channel is disabled.

- **RESx: Resume [3:0]**

Write one to this field to resume the channel transfer restoring its context.

### 27.7.12 DMAC Channel Handler Status Register

**Name:** DMAC\_CHSR

**Address:** 0x400C0030

**Access:** Read-only

**Reset:** 0x00FF0000

31	30	29	28	27	26	25	24
–	–	–	–	STAL3	STAL2	STAL1	STAL0
23	22	21	20	19	18	17	16
–	–	–	–	EMPT3	EMPT2	EMPT1	EMPT0
15	14	13	12	11	10	9	8
–	–	–	–	SUSP3	SUSP2	SUSP1	SUSP0
7	6	5	4	3	2	1	0
–	–	–	–	ENA3	ENA2	ENA1	ENA0

- **ENAx: Enable [3:0]**

A one in any position of this field indicates that the relevant channel is enabled.

- **SUSPx: Suspend [3:0]**

A one in any position of this field indicates that the channel transfer is suspended.

- **EMPTx: Empty [3:0]**

A one in any position of this field indicates that the relevant channel is empty.

- **STALx: Stalled [3:0]**

A one in any position of this field indicates that the relevant channel is stalling.

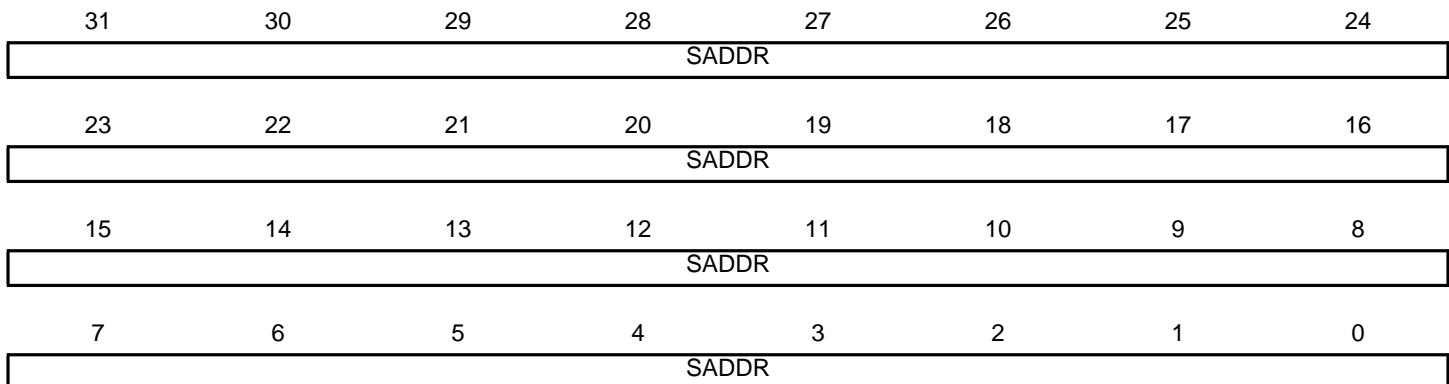
### 27.7.13 DMAC Channel x [x = 0..3] Source Address Register

**Name:** DMAC\_SADDRx [x = 0..3]

**Address:** 0x400C003C [0], 0x400C0064 [1], 0x400C008C [2], 0x400C00B4 [3]

**Access:** Read-write

**Reset:** 0x00000000



This register can only be written if the WPEN bit is cleared in “[DMAC Write Protect Mode Register](#)” .

- **SADDR: Channel x Source Address**

This register must be aligned with the source transfer width.

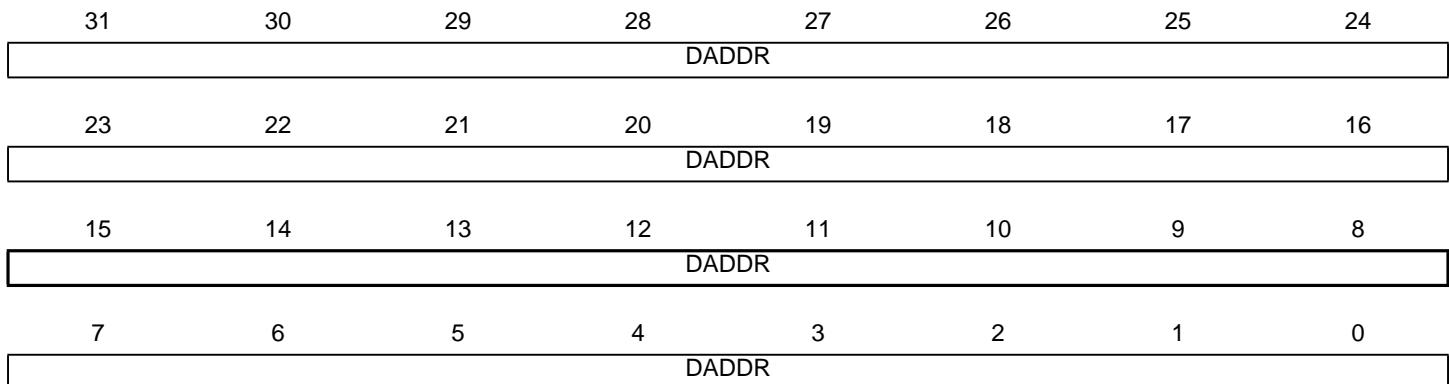
### 27.7.14 DMAC Channel x [x = 0..3] Destination Address Register

**Name:** DMAC\_DADDRx [x = 0..3]

**Address:** 0x400C0040 [0], 0x400C0068 [1], 0x400C0090 [2], 0x400C00B8 [3]

**Access:** Read-write

**Reset:** 0x00000000



This register can only be written if the WPEN bit is cleared in “[DMAC Write Protect Mode Register](#)” .

- **DADDR: Channel x Destination Address**

This register must be aligned with the destination transfer width.

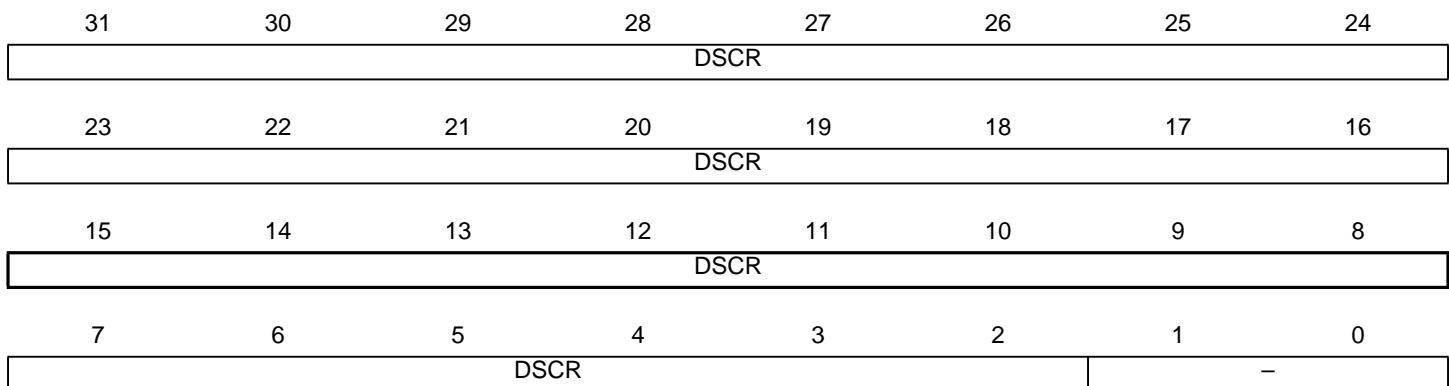
### 27.7.15 DMAC Channel x [x = 0..3] Descriptor Address Register

**Name:** DMAC\_DSCR<sub>x</sub> [x = 0..3]

**Address:** 0x400C0044 [0], 0x400C006C [1], 0x400C0094 [2], 0x400C00BC [3]

**Access:** Read-write

**Reset:** 0x00000000



This register can only be written if the WPEN bit is cleared in “DMAC Write Protect Mode Register” .

- **DSCR: Buffer Transfer Descriptor Address**

This address is word aligned.

### 27.7.16 DMAC Channel x [x = 0..3] Control A Register

**Name:** DMAC\_CTRLAx [x = 0..3]

**Address:** 0x400C0048 [0], 0x400C0070 [1], 0x400C0098 [2], 0x400C00C0 [3]

**Access:** Read-write

**Reset:** 0x00000000

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

This register can only be written if the WPEN bit is cleared in “DMAC Write Protect Mode Register” on page 467

- **BTSIZE: Buffer Transfer Size**

The transfer size relates to the number of transfers to be performed, that is, for writes it refers to the number of source width transfers to perform when DMAC is flow controller. For Reads, BTSIZE refers to the number of transfers completed on the Source Interface. When this field is set to 0, the DMAC module is automatically disabled when the relevant channel is enabled.

- **SRC\_WIDTH: Transfer Width for the Source**

Value	Name	Description
00	BYTE	the transfer size is set to 8-bit width
01	HALF_WORD	the transfer size is set to 16-bit width
1X	WORD	the transfer size is set to 32-bit width

- **DST\_WIDTH: Transfer Width for the Destination**

Value	Name	Description
00	BYTE	the transfer size is set to 8-bit width
01	HALF_WORD	the transfer size is set to 16-bit width
1X	WORD	the transfer size is set to 32-bit width

- **DONE: Current Descriptor Stop Command and Transfer Completed Memory Indicator**

0: The transfer is performed.

1: If SOD field of DMAC\_CFG register is set to true, then the DMAC is automatically disabled when an LLI updates the content of this register.

The DONE field is written back to memory at the end of the current descriptor transfer.

### 27.7.17 DMAC Channel x [x = 0..3] Control B Register

**Name:** DMAC\_CTRLBx [x = 0..3]

**Address:** 0x400C004C [0], 0x400C0074 [1], 0x400C009C [2], 0x400C00C4 [3]

**Access:** Read-write

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
–	IEN	DST_INCR	–	–	–	SRC_INCR	–
23	22	21	20	19	18	17	16
–	FC	DST_DSCR	–	–	–	–	SRC_DSCR
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in “[DMAC Write Protect Mode Register](#)” .

- **SRC\_DSCR: Source Address Descriptor**

0 (FETCH\_FROM\_MEM): Source address is updated when the descriptor is fetched from the memory.

1 (FETCH\_DISABLE): Buffer Descriptor Fetch operation is disabled for the source.

- **DST\_DSCR: Destination Address Descriptor**

0 (FETCH\_FROM\_MEM): Destination address is updated when the descriptor is fetched from the memory.

1 (FETCH\_DISABLE): Buffer Descriptor Fetch operation is disabled for the destination.

- **FC: Flow Control**

This field defines which device controls the size of the buffer transfer, also referred to as the Flow Controller.

Value	Name	Description
00	MEM2MEM_DMA_FC	Memory-to-Memory Transfer DMAC is flow controller
01	MEM2PER_DMA_FC	Memory-to-Peripheral Transfer DMAC is flow controller
10	PER2MEM_DMA_FC	Peripheral-to-Memory Transfer DMAC is flow controller
11	PER2PER_DMA_FC	Peripheral-to-Peripheral Transfer DMAC is flow controller

- **SRC\_INCR: Incrementing, Decrementing or Fixed Address for the Source**

Value	Name	Description
00	INCREMENTING	The source address is incremented
01	DECREMENTING	The source address is decremented
10	FIXED	The source address remains unchanged

- **DST\_INCR: Incrementing, Decrementing or Fixed Address for the Destination**

Value	Name	Description
00	INCREMENTING	The destination address is incremented
01	DECREMENTING	The destination address is decremented
10	FIXED	The destination address remains unchanged

- **IEN: Interrupt Enable Not**

0: When the buffer transfer is completed, the BTCx flag is set in the EBCISR status register. This bit is active low.

1: When the buffer transfer is completed, the BTCx flag is not set.

If this bit is cleared, when the buffer transfer is completed, the BTCx flag is set in the EBCISR status register.

### 27.7.18 DMAC Channel x [x = 0..3] Configuration Register

**Name:** DMAC\_CFGx [x = 0..3]

**Address:** 0x400C0050 [0], 0x400C0078 [1], 0x400C00A0 [2], 0x400C00C8 [3]

**Access:** Read-write

**Reset:** 0x0100000000

31	30	29	28	27	26	25	24
-	-	FIFO CFG	-	-	AHB PROT		
23	22	21	20	19	18	17	16
-	LOCK_IF_L	LOCK_B	LOCK_IF	-	-	-	SOD
15	14	13	12	11	10	9	8
-	-	DST_H2SEL	-	-	-	SRC_H2SEL	-
7	6	5	4	3	2	1	0
		DST_PER			SRC_PER		

This register can only be written if the WPEN bit is cleared in “DMAC Write Protect Mode Register” on page 467

- **SRC\_PER: Source with Peripheral identifier**

Channel x Source Request is associated with peripheral identifier coded SRC\_PER handshaking interface.

- **DST\_PER: Destination with Peripheral identifier**

Channel x Destination Request is associated with peripheral identifier coded DST\_PER handshaking interface.

- **SRC\_H2SEL: Software or Hardware Selection for the Source**

0 (SW): Software handshaking interface is used to trigger a transfer request.

1 (HW): Hardware handshaking interface is used to trigger a transfer request.

- **DST\_H2SEL: Software or Hardware Selection for the Destination**

0 (SW): Software handshaking interface is used to trigger a transfer request.

1 (HW): Hardware handshaking interface is used to trigger a transfer request.

- **SOD: Stop On Done**

0 (DISABLE): STOP ON DONE disabled, the descriptor fetch operation ignores DONE Field of CTRLA register.

1 (ENABLE): STOP ON DONE activated, the DMAC module is automatically disabled if DONE FIELD is set to 1.

- **LOCK\_IF: Interface Lock**

0 (DISABLE): Interface Lock capability is disabled

1 (ENABLE): Interface Lock capability is enabled

- **LOCK\_B: Bus Lock**

0 (DISABLE): AHB Bus Locking capability is disabled.

1 (ENABLE): AHB Bus Locking capability is enabled.

- **LOCK\_IF\_L: Master Interface Arbiter Lock**

0 (CHUNK): The Master Interface Arbiter is locked by the channel x for a chunk transfer.

1 (BUFFER): The Master Interface Arbiter is locked by the channel x for a buffer transfer.

- **AHB\_PROT: AHB Protection**

AHB\_PROT field provides additional information about a bus access and is primarily used to implement some level of protection.

HPROT[3]	HPROT[2]	HPROT[1]	HPROT[0]	Description
			1	Data access
		AHB_PROT[0]		0: User Access 1: Privileged Access
	AHB_PROT[1]			0: Not Bufferable 1: Bufferable
AHB_PROT[2]				0: Not cacheable 1: Cacheable

- **FIFO CFG: FIFO Configuration**

Value	Name	Description
00	ALAP_CFG	The largest defined length AHB burst is performed on the destination AHB interface.
01	HALF_CFG	When half FIFO size is available/filled, a source/destination request is serviced.
10	ASAP_CFG	When there is enough space/data available to perform a single AHB access, then the request is serviced.

### 27.7.19 DMAC Write Protect Mode Register

**Name:** DMAC\_WPMR

**Address:** 0x400C01E4

**Access:** Read-write

**Reset:** See [Table 27-4](#)

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x444D41 (“DMA” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x444D41 (“DMA” in ASCII).

Protects the registers:

- “[DMAC Global Configuration Register](#)” on page 447
- “[DMAC Enable Register](#)” on page 448
- “[DMAC Channel x \[x = 0..3\] Source Address Register](#)” on page 459
- “[DMAC Channel x \[x = 0..3\] Destination Address Register](#)” on page 460
- “[DMAC Channel x \[x = 0..3\] Descriptor Address Register](#)” on page 461
- “[DMAC Channel x \[x = 0..3\] Control A Register](#)” on page 462
- “[DMAC Channel x \[x = 0..3\] Control B Register](#)” on page 463
- “[DMAC Channel x \[x = 0..3\] Configuration Register](#)” on page 465

- **WPKEY: Write Protect KEY**

Should be written at value 0x444D41 (“DMA” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 27.7.20 DMAC Write Protect Status Register

**Name:** DMAC\_WPSR

**Address:** 0x400C01E8

**Access:** Read-only

**Reset:** See [Table 27-4](#)

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
WPVSRC							
WPVSRC							
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	WPVS
7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	WPVS

- **WPVS: Write Protect Violation Status**

0 = No Write Protect Violation has occurred since the last read of the DMAC\_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the DMAC\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Note: Reading DMAC\_WPSR automatically clears all fields.

## 28. Static Memory Controller (SMC)

### 28.1 Description

The External Bus Interface is designed to ensure the successful data transfer between several external devices and the Cortex-M4 based device. The External Bus Interface of the SAM4E consists of a Static Memory Controller (SMC).

This SMC is capable of handling several types of external memory and peripheral devices, such as SRAM, PSRAM, PROM, EPROM, EEPROM, LCD Module, NOR Flash and NAND Flash.

The Static Memory Controller (SMC) generates the signals that control the access to the external memory devices or peripheral devices. It has 4 Chip Selects, a 24-bit address bus, and an 8-bit data bus. Separate read and write control signals allow for direct memory and peripheral interfacing. Read and write signal waveforms are fully adjustable.

The SMC can manage wait requests from external devices to extend the current access. The SMC is provided with an automatic slow clock mode. In slow clock mode, it switches from user-programmed waveforms to slow-rate specific waveforms on read and write signals. The SMC supports asynchronous burst read in page mode access for page size up to 32 bytes.

The External Data Bus can be scrambled/unscrambled by means of user keys.

### 28.2 Embedded Characteristics

- 4 Chip Selects Available
- 16-Mbyte Address Space per Chip Select
- 8-bit Data Bus
- Zero Wait State Scrambling/Unscrambling function with User Key
- Word, Halfword, Byte Transfers
- Programmable Setup, Pulse And Hold Time for Read Signals per Chip Select
- Programmable Setup, Pulse And Hold Time for Write Signals per Chip Select
- Programmable Data Float Time per Chip Select
- External Data Bus Scrambling/Unscrambling Function
- External Wait Request
- Automatic Switch to Slow Clock Mode
- Asynchronous Read in Page Mode Supported: Page Size Ranges from 4 to 32 Bytes
- Write Protected Registers

## 28.3 I/O Lines Description

Table 28-1. I/O Line Description

Name	Description	Type	Active Level
NCS[3:0]	Static Memory Controller Chip Select Lines	Output	Low
NRD	Read Signal	Output	Low
NWE	Write Enable Signal	Output	Low
A[23:0]	Address Bus	Output	
D[7:0]	Data Bus	I/O	
NWAIT	External Wait Signal	Input	Low
NANDCS	NAND Flash Chip Select Line	Output	Low
NANDOE	NAND Flash Output Enable	Output	Low
NANDWE	NAND Flash Write Enable	Output	Low

## 28.4 Product Dependencies

### 28.4.1 I/O Lines

The pins used for interfacing the Static Memory Controller are multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the Static Memory Controller pins to their peripheral function. If I/O Lines of the SMC are not used by the application, they can be used for other purposes by the PIO Controller.

### 28.4.2 Power Management

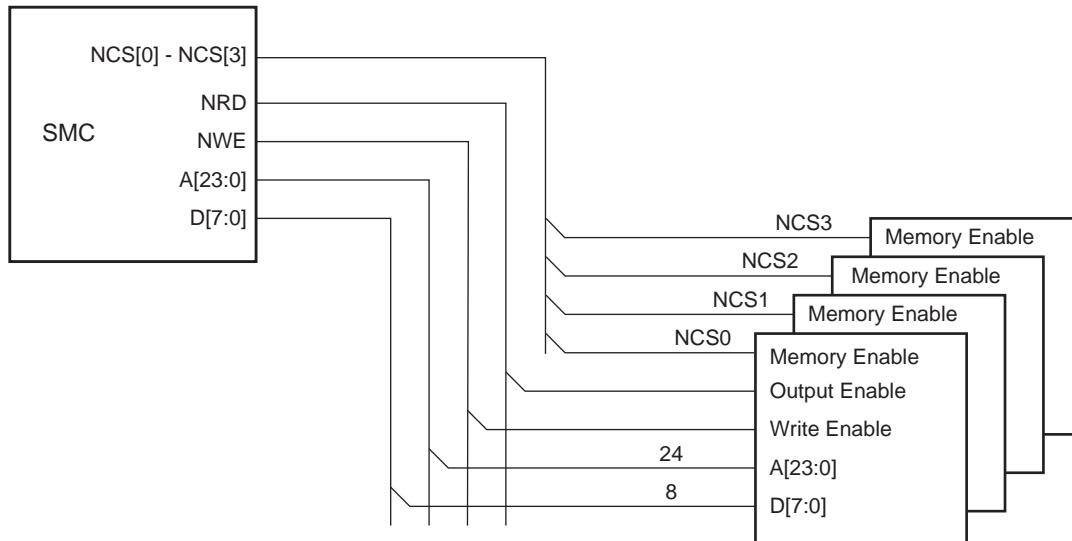
The SMC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SMC clock.

## 28.5 External Memory Mapping

The SMC provides up to 24 address lines, A[23:0]. This allows each chip select line to address up to 16 Mbytes of memory.

If the physical memory device connected on one chip select is smaller than 16 Mbytes, it wraps around and appears to be repeated within this space. The SMC correctly handles any valid access to the memory device within the page (see [Figure 28-1](#)).

**Figure 28-1. Memory Connections for Four External Devices**



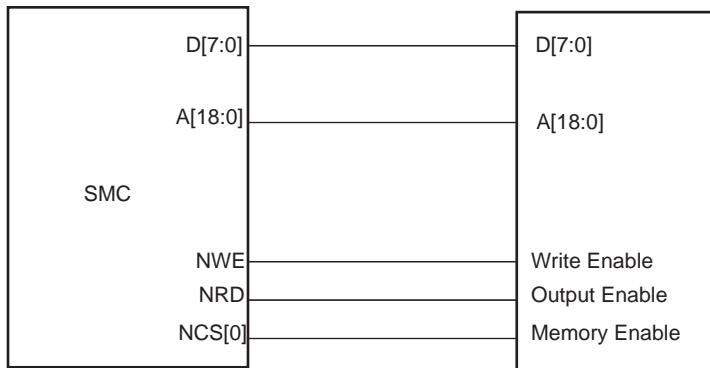
## 28.6 Connection to External Devices

### 28.6.1 Data Bus Width

The data bus width is 8 bits.

Figure 28-2 shows how to connect a 512K x 8-bit memory on NCS0.

Figure 28-2. Memory Connection for an 8-bit Data Bus



#### 28.6.1.1 NAND Flash Support

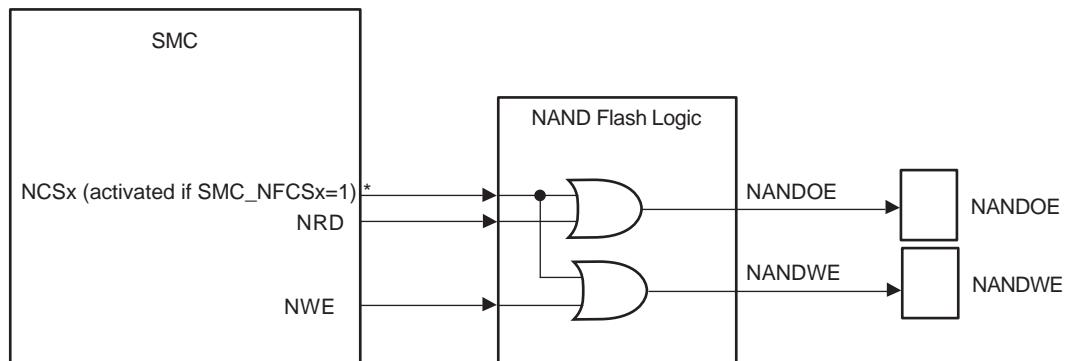
The SMC integrates circuitry that interfaces to NAND Flash devices.

The NAND Flash logic is driven by the Static Memory Controller. It depends on the programming of the SMC\_NFCSx field in the CCFG\_SMCNFCS Register on the Bus Matrix User Interface. For details on this register, refer to the Bus Matrix User Interface section. Access to an external NAND Flash device via the address space reserved to the chip select programmed.

The user can connect up to 4 NAND Flash devices with separated chip select.

The NAND Flash logic drives the read and write command signals of the SMC on the NANDOE and NANDWE signals when the NCSx programmed is active. NANDOE and NANDWE are disabled as soon as the transfer address fails to lie in the NCSx programmed address space.

Figure 28-3. NAND Flash Signal Multiplexing on SMC Pins



\* in CCFG\_SMCNFCS Matrix register

**Note:** When NAND Flash logic is activated, (SMCNFCSx=1), NWE pin cannot be used in PIO Mode but only in peripheral mode (NWE function). If NWE function is not used for other external memories (SRAM, LCD), it must be configured in one of the following modes.

- PIO Input with pull-up enabled (default state after reset)
- PIO Output set at level 1

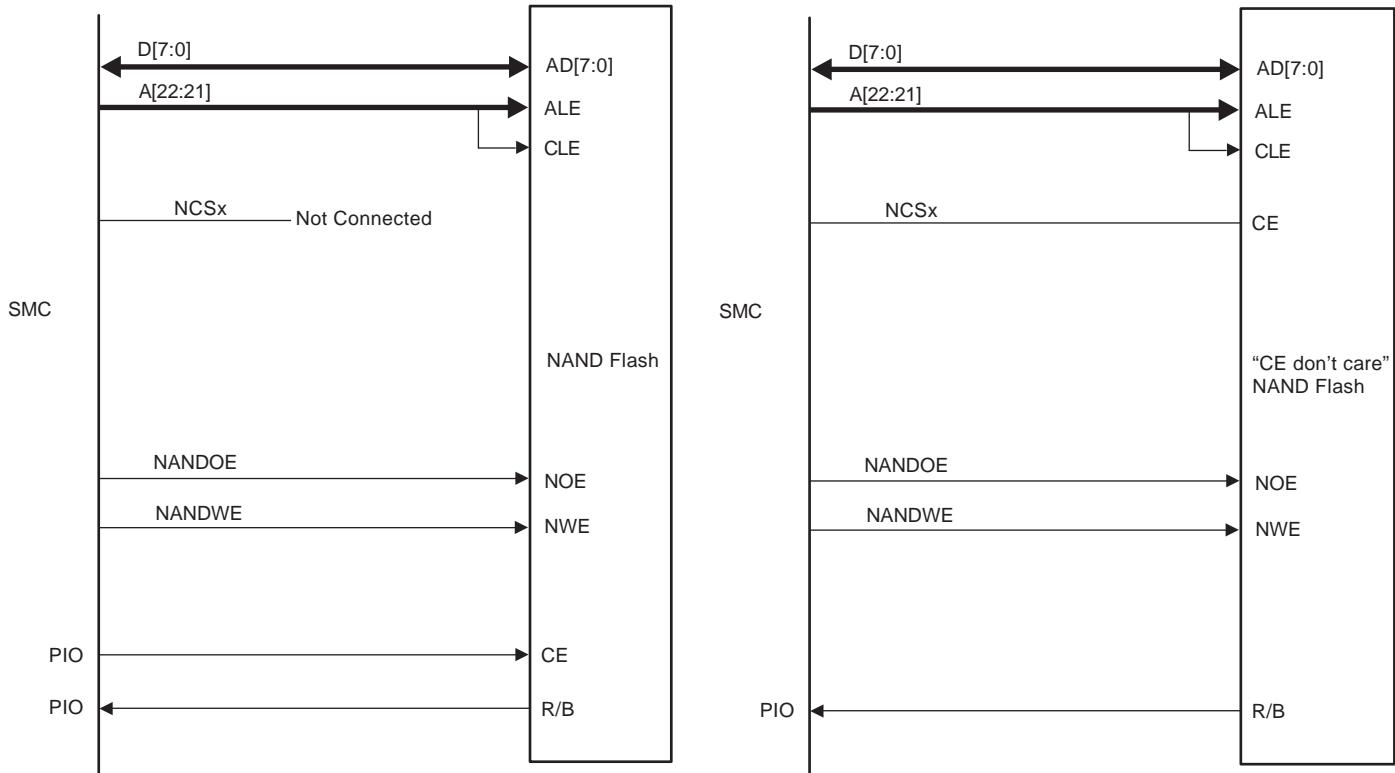
The address latch enable and command latch enable signals on the NAND Flash device are driven by address bits A22 and A21 of the address bus. Any bit of the address bus can also be used for this purpose. The command, address or data words on the data bus of the NAND Flash device use their own addresses within the NCSx address space (configured by CCFG\_SMCNFCS Register on the Bus Matrix User Interface). The chip enable (CE) signal of the device and the ready/busy (R/B) signals are connected to PIO lines. The CE signal then remains asserted even when NCS3 is not selected, preventing the device from returning to standby mode. The NANDCS output signal should be used in accordance with the external NAND Flash device type.

Two types of CE behavior exist depending on the NAND flash device:

- Standard NAND Flash devices require that the CE pin remains asserted Low continuously during the read busy period to prevent the device from returning to standby mode. Since the Static Memory Controller (SMC) asserts the NCSx signal High, it is necessary to connect the CE pin of the NAND Flash device to a GPIO line, in order to hold it low during the busy period preceding data read out.
- This restriction has been removed for “CE don’t care” NAND Flash devices. The NCSx signal can be directly connected to the CE pin of the NAND Flash device.

Figure 28-4 illustrates both topologies: Standard and “CE don’t care” NAND Flash.

**Figure 28-4. Standard and “CE don’t care” NAND Flash Application Examples**



## 28.7 Application Example

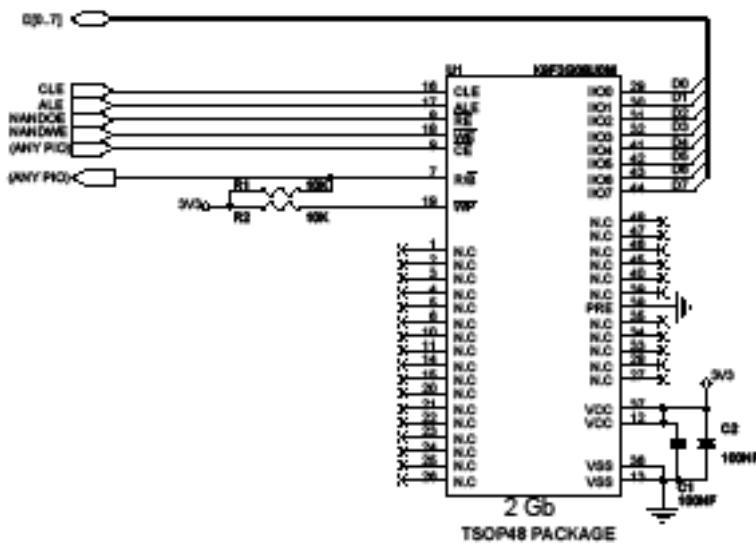
### 28.7.1 Implementation Examples

Hardware configurations are given for illustration only. The user should refer to the manufacturer web site to check for memory device availability.

For hardware implementation examples, refer to SAM4E-EK schematics, which show examples of a connection to an LCD module and NAND Flash.

#### 28.7.1.1 8-bit NAND Flash

##### *Hardware Configuration*



##### *Software Configuration*

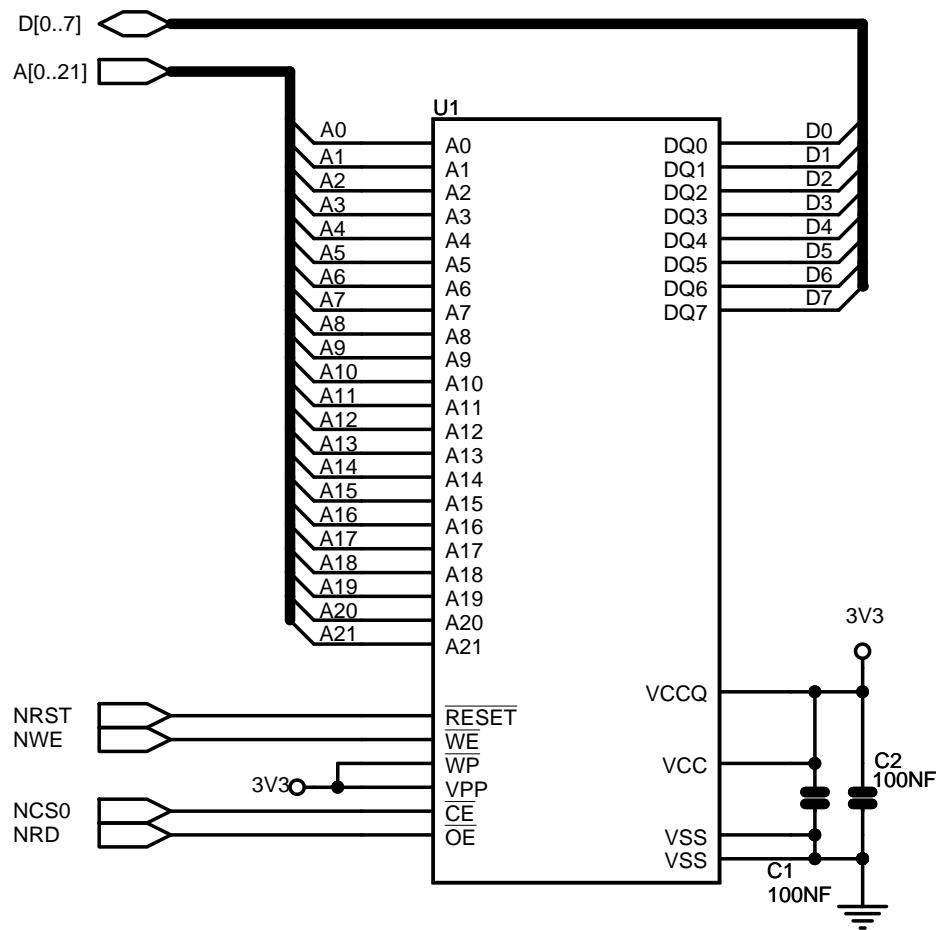
Perform the following configuration:

- Assign the SMC\_NFCx (for example SMC\_NFCS3) field in the CCFG\_SMCNFCS Register on the Bus Matrix User Interface.
- Reserve A21 / A22 for ALE / CLE functions. Address and Command Latches are controlled respectively by setting to 1 the address bits A21 and A22 during accesses.
- NANDOE and NANDWE signals are multiplexed with PIO lines. Thus, the dedicated PIOs must be programmed in peripheral mode in the PIO controller.
- Configure a PIO line as an input to manage the Ready/Busy signal.
- Configure Static Memory Controller CS3 Setup, Pulse, Cycle and Mode according to NAND Flash timings, the data bus width and the system bus frequency.

In this example, the NAND Flash is not addressed as a “CE don’t care”. To address it as a “CE don’t care”, connect NCS3 (if SMC\_NFCS3 is set) to the NAND Flash CE.

## 28.7.1.2 NOR Flash

### Hardware Configuration



### Software Configuration

Configure the Static Memory Controller CS0 Setup, Pulse, Cycle and Mode depending on Flash timings and system bus frequency.

## 28.8 Standard Read and Write Protocols

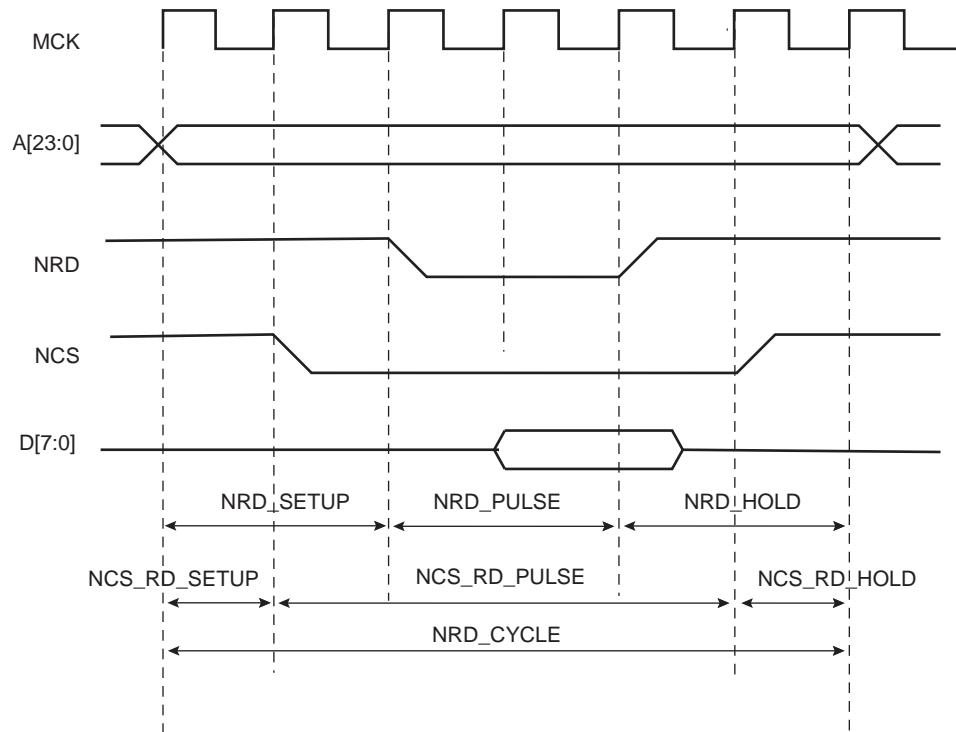
In the following sections, NCS represents one of the NCS[0..3] chip select lines.

### 28.8.1 Read Waveforms

The read cycle is shown on [Figure 28-5](#).

The read cycle starts with the address setting on the memory address bus.

**Figure 28-5. Standard Read Cycle**



#### 28.8.1.1 NRD Waveform

The NRD signal is characterized by a setup timing, a pulse width and a hold timing.

1. NRD\_SETUP: the NRD setup time is defined as the setup of address before the NRD falling edge;
2. NRD\_PULSE: the NRD pulse length is the time between NRD falling edge and NRD rising edge;
3. NRD\_HOLD: the NRD hold time is defined as the hold time of address after the NRD rising edge.

#### 28.8.1.2 NCS Waveform

Similarly, the NCS signal can be divided into a setup time, pulse length and hold time:

1. NCS\_RD\_SETUP: the NCS setup time is defined as the setup time of address before the NCS falling edge;
2. NCS\_RD\_PULSE: the NCS pulse length is the time between NCS falling edge and NCS rising edge;
3. NCS\_RD\_HOLD: the NCS hold time is defined as the hold time of address after the NCS rising edge.

#### 28.8.1.3 Read Cycle

The NRD\_CYCLE time is defined as the total duration of the read cycle, i.e., from the time where address is set on the address bus to the point where address may change. The total read cycle time is equal to:

$$\begin{aligned} \text{NRD\_CYCLE} &= \text{NRD\_SETUP} + \text{NRD\_PULSE} + \text{NRD\_HOLD} \\ &= \text{NCS\_RD\_SETUP} + \text{NCS\_RD\_PULSE} + \text{NCS\_RD\_HOLD} \end{aligned}$$

All NRD and NCS timings are defined separately for each chip select as an integer number of Master Clock cycles. To ensure that the NRD and NCS timings are coherent, user must define the total read cycle instead of the hold timing. NRD\_CYCLE implicitly defines the NRD hold time and NCS hold time as:

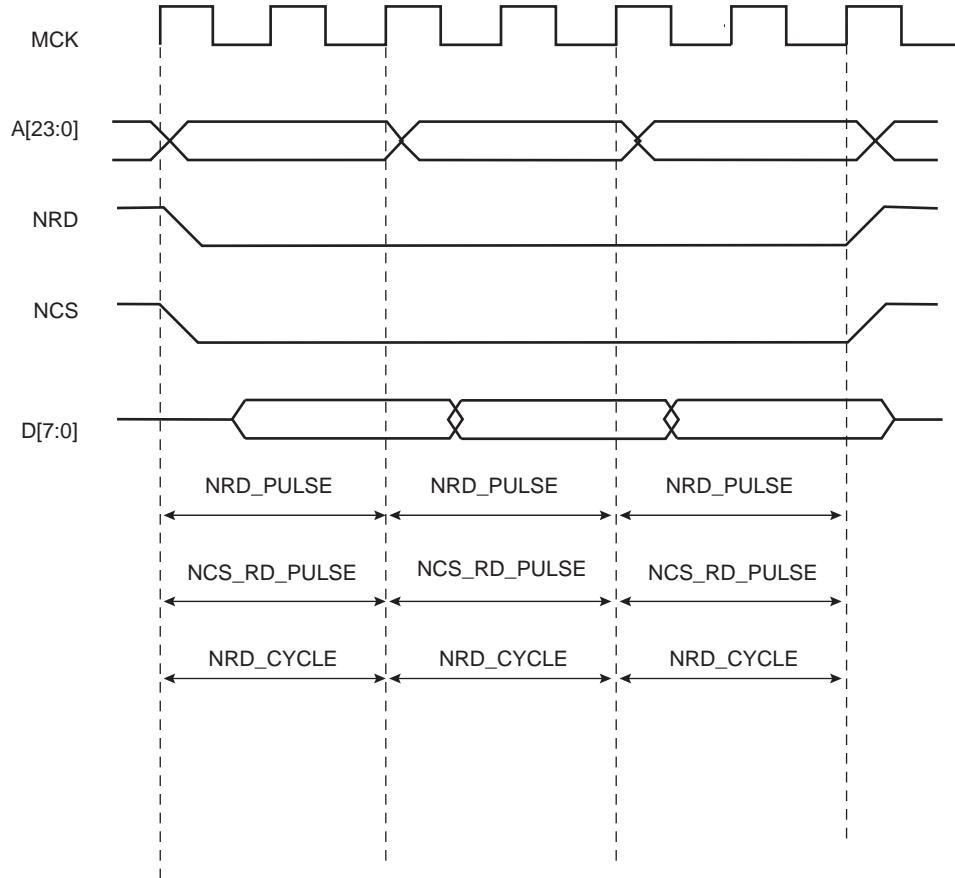
$$\text{NRD\_HOLD} = \text{NRD\_CYCLE} - \text{NRD\_SETUP} - \text{NRD\_PULSE}$$

$$\text{NCS\_RD\_HOLD} = \text{NRD\_CYCLE} - \text{NCS\_RD\_SETUP} - \text{NCS\_RD\_PULSE}$$

#### 28.8.1.4 Null Delay Setup and Hold

If null setup and hold parameters are programmed for NRD and/or NCS, NRD and NCS remain active continuously in case of consecutive read cycles in the same memory (see [Figure 28-6](#)).

**Figure 28-6. No Setup, No Hold on NRD and NCS Read Signals**



#### 28.8.1.5 Null Pulse

Programming null pulse is not permitted. Pulse must be at least set to 1. A null value leads to unpredictable behavior.

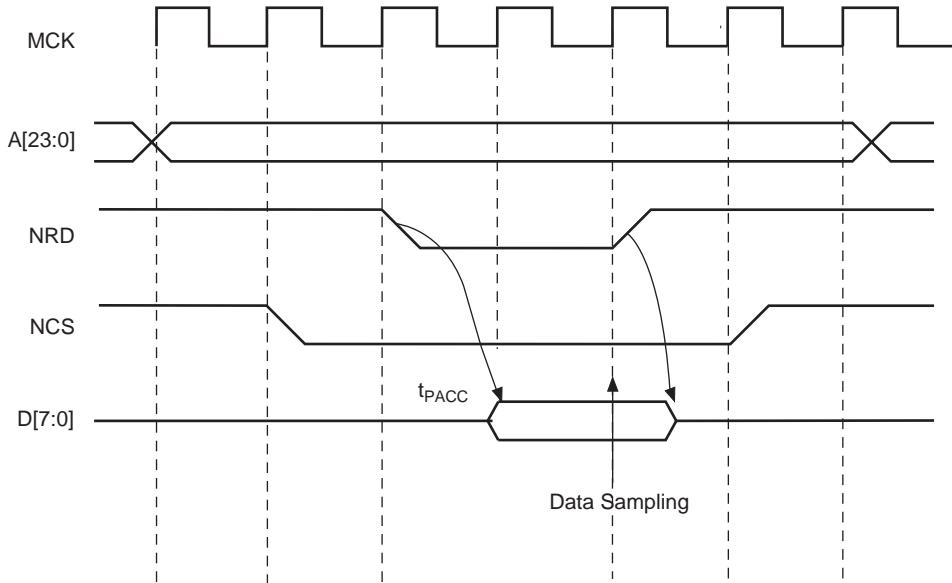
#### 28.8.2 Read Mode

As NCS and NRD waveforms are defined independently of one other, the SMC needs to know when the read data is available on the data bus. The SMC does not compare NCS and NRD timings to know which signal rises first. The READ\_MODE parameter in the SMC\_MODE register of the corresponding chip select indicates which signal of NRD and NCS controls the read operation.

### 28.8.2.1 Read is Controlled by NRD (READ\_MODE = 1):

Figure 28-7 shows the waveforms of a read operation of a typical asynchronous RAM. The read data is available  $t_{PACC}$  after the falling edge of NRD, and turns to 'Z' after the rising edge of NRD. In this case, the READ\_MODE must be set to 1 (read is controlled by NRD), to indicate that data is available with the rising edge of NRD. The SMC samples the read data internally on the rising edge of Master Clock that generates the rising edge of NRD, whatever the programmed waveform of NCS may be.

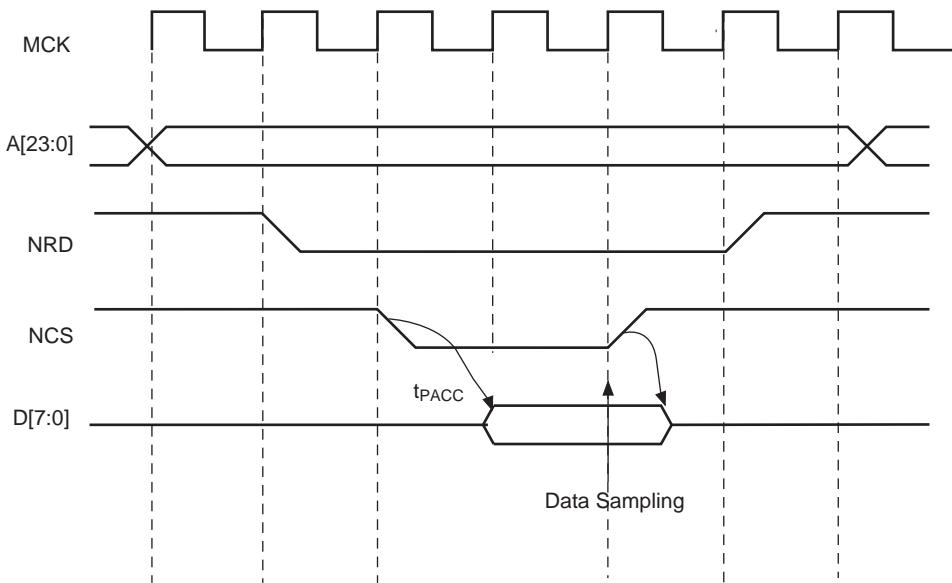
**Figure 28-7. READ\_MODE = 1: Data is sampled by SMC before the rising edge of NRD**



### 28.8.2.2 Read is Controlled by NCS (READ\_MODE = 0)

Figure 28-8 shows the typical read cycle of an LCD module. The read data is valid  $t_{PACC}$  after the falling edge of the NCS signal and remains valid until the rising edge of NCS. Data must be sampled when NCS is raised. In that case, the READ\_MODE must be set to 0 (read is controlled by NCS): the SMC internally samples the data on the rising edge of Master Clock that generates the rising edge of NCS, whatever the programmed waveform of NRD may be.

**Figure 28-8. READ\_MODE = 0: Data is sampled by SMC before the rising edge of NCS**



### 28.8.3 Write Waveforms

The write protocol is similar to the read protocol. It is depicted in [Figure 28-9](#). The write cycle starts with the address setting on the memory address bus.

#### 28.8.3.1 NWE Waveforms

The NWE signal is characterized by a setup timing, a pulse width and a hold timing.

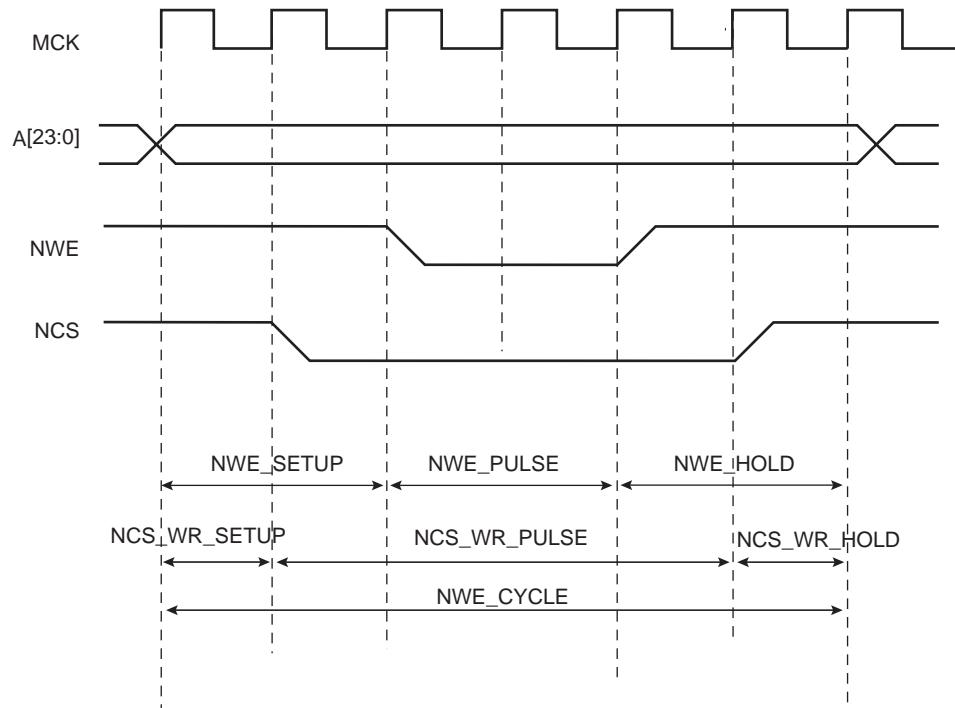
1. NWE\_SETUP: the NWE setup time is defined as the setup of address and data before the NWE falling edge;
2. NWE\_PULSE: The NWE pulse length is the time between NWE falling edge and NWE rising edge;
3. NWE\_HOLD: The NWE hold time is defined as the hold time of address and data after the NWE rising edge.

#### 28.8.3.2 NCS Waveforms

The NCS signal waveforms in write operation are not the same than those applied in read operations, but are separately defined:

1. NCS\_WR\_SETUP: the NCS setup time is defined as the setup time of address before the NCS falling edge.
2. NCS\_WR\_PULSE: the NCS pulse length is the time between NCS falling edge and NCS rising edge;
3. NCS\_WR\_HOLD: the NCS hold time is defined as the hold time of address after the NCS rising edge.

**Figure 28-9. Write Cycle**



#### 28.8.3.3 Write Cycle

The write\_cycle time is defined as the total duration of the write cycle, that is, from the time where address is set on the address bus to the point where address may change. The total write cycle time is equal to:

$$\begin{aligned} \text{NWE\_CYCLE} &= \text{NWE\_SETUP} + \text{NWE\_PULSE} + \text{NWE\_HOLD} \\ &= \text{NCS\_WR\_SETUP} + \text{NCS\_WR\_PULSE} + \text{NCS\_WR\_HOLD} \end{aligned}$$

All NWE and NCS (write) timings are defined separately for each chip select as an integer number of Master Clock cycles. To ensure that the NWE and NCS timings are coherent, the user must define the total write cycle instead of the hold timing. This implicitly defines the NWE hold time and NCS (write) hold times as:

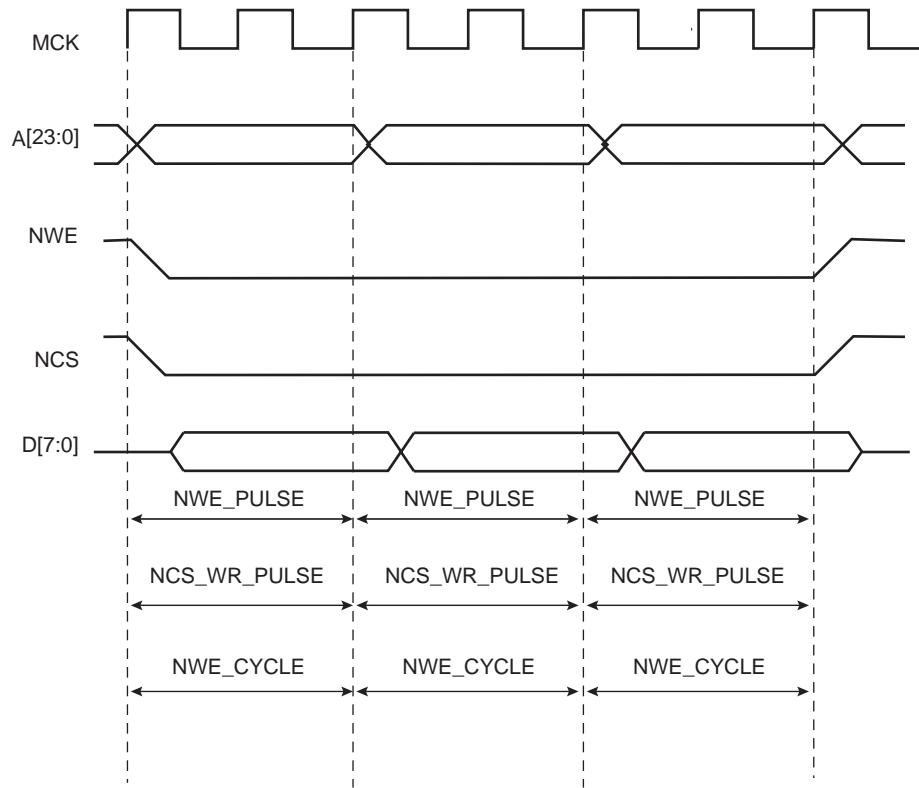
$$\text{NWE\_HOLD} = \text{NWE\_CYCLE} - \text{NWE\_SETUP} - \text{NWE\_PULSE}$$

$$\text{NCS\_WR\_HOLD} = \text{NWE\_CYCLE} - \text{NCS\_WR\_SETUP} - \text{NCS\_WR\_PULSE}$$

#### 28.8.3.4 Null Delay Setup and Hold

If null setup parameters are programmed for NWE and/or NCS, NWE and/or NCS remain active continuously in case of consecutive write cycles in the same memory (see [Figure 28-10](#)). However, for devices that perform write operations on the rising edge of NWE or NCS, either a setup or a hold must be programmed.

**Figure 28-10. Null Setup and Hold Values of NCS and NWE in Write Cycle**



#### 28.8.3.5 Null Pulse

Programming null pulse is not permitted. Pulse must be at least set to 1. A null value leads to unpredictable behavior.

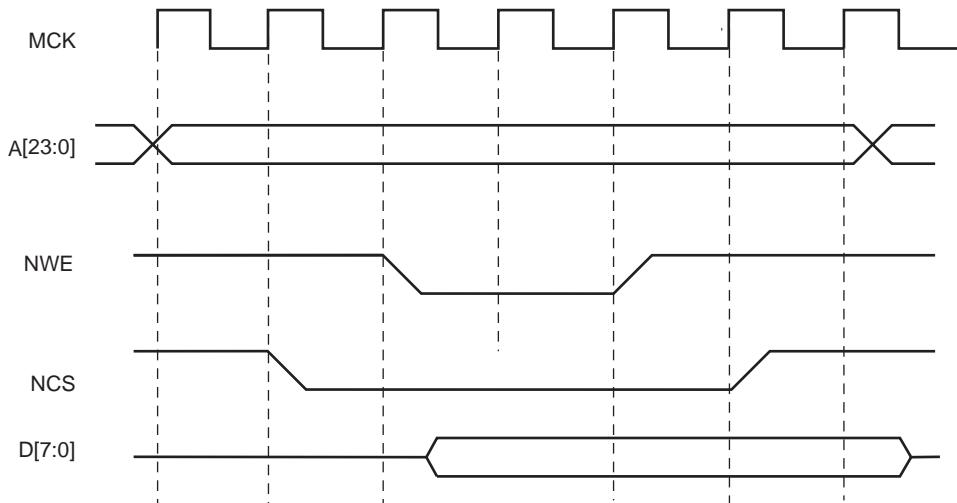
### 28.8.4 Write Mode

The **WRITE\_MODE** parameter in the **SMC\_MODE** register of the corresponding chip select indicates which signal controls the write operation.

#### 28.8.4.1 Write is Controlled by NWE (**WRITE\_MODE = 1**):

[Figure 28-11](#) shows the waveforms of a write operation with **WRITE\_MODE** set to 1. The data is put on the bus during the pulse and hold steps of the NWE signal. The internal data buffers are switched to output mode after the **NWE\_SETUP** time, and until the end of the write cycle, regardless of the programmed waveform on NCS.

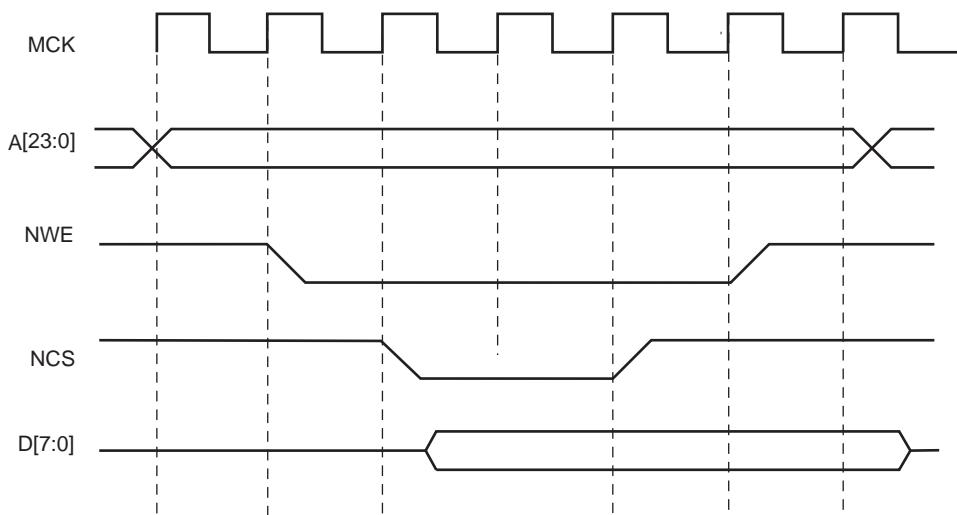
**Figure 28-11. WRITE\_MODE = 1. The write operation is controlled by NWE**



#### 28.8.4.2 Write is Controlled by NCS (WRITE\_MODE = 0)

Figure 28-12 shows the waveforms of a write operation with WRITE\_MODE set to 0. The data is put on the bus during the pulse and hold steps of the NCS signal. The internal data buffers are switched to output mode after the NCS\_WR\_SETUP time, and until the end of the write cycle, regardless of the programmed waveform on NWE.

**Figure 28-12. WRITE\_MODE = 0. The write operation is controlled by NCS**



#### 28.8.5 Write Protected Registers

To prevent any single software error that may corrupt SMC behavior, the registers listed below can be write-protected by setting the WPEN bit in the SMC Write Protect Mode Register (SMC\_WPMR).

If a write access in a write-protected register is detected, then the WPVS flag in the SMC Write Protect Status Register (SMC\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is automatically reset after reading the SMC Write Protect Status Register (SMC\_WPSR).

List of the write-protected registers:

- [Section 28.15.1 "SMC Setup Register"](#)
- [Section 28.15.2 "SMC Pulse Register"](#)
- [Section 28.15.3 "SMC Cycle Register"](#)

- Section 28.15.4 "SMC MODE Register"

### 28.8.6 Coding Timing Parameters

All timing parameters are defined for one chip select and are grouped together in one SMC\_REGISTER according to their type.

The SMC\_SETUP register groups the definition of all setup parameters:

- NRD\_SETUP, NCS\_RD\_SETUP, NWE\_SETUP, NCS\_WR\_SETUP

The SMC\_PULSE register groups the definition of all pulse parameters:

- NRD\_PULSE, NCS\_RD\_PULSE, NWE\_PULSE, NCS\_WR\_PULSE

The SMC\_CYCLE register groups the definition of all cycle parameters:

- NRD\_CYCLE, NWE\_CYCLE

**Table 28-2** shows how the timing parameters are coded and their permitted range.

**Table 28-2. Coding and Range of Timing Parameters**

Coded Value	Number of Bits	Effective Value	Permitted Range	
			Coded Value	Effective Value
setup [5:0]	6	128 x setup[5] + setup[4:0]	0 ≤ 31	0 ≤ 128+31
pulse [6:0]	7	256 x pulse[6] + pulse[5:0]	0 ≤ 63	0 ≤ 256+63
cycle [8:0]	9	256 x cycle[8:7] + cycle[6:0]	0 ≤ 127	0 ≤ 512+127 0 ≤ 768+127

### 28.8.7 Reset Values of Timing Parameters

**Table 28-3** gives the default value of timing parameters at reset.

**Table 28-3. Reset Values of Timing Parameters**

Register	Reset Value	
SMC_SETUP	0x01010101	All setup timings are set to 1
SMC_PULSE	0x01010101	All pulse timings are set to 1
SMC_CYCLE	0x00030003	The read and write operation last 3 Master Clock cycles and provide one hold cycle
WRITE_MODE	1	Write is controlled with NWE
READ_MODE	1	Read is controlled with NRD

### 28.8.8 Usage Restriction

The SMC does not check the validity of the user-programmed parameters. If the sum of SETUP and PULSE parameters is larger than the corresponding CYCLE parameter, this leads to unpredictable behavior of the SMC.

For read operations:

Null but positive setup and hold of address and NRD and/or NCS can not be guaranteed at the memory interface because of the propagation delay of these signals through external logic and pads. If positive setup and hold values must be verified, then it is strictly recommended to program non-null values so as to cover possible skews between address, NCS and NRD signals.

For write operations:

If a null hold value is programmed on NWE, the SMC can guarantee a positive hold of address and NCS signal after the rising edge of NWE. This is true for WRITE\_MODE = 1 only. See “[Early Read Wait State](#)” on page 484.

For read and write operations: a null value for pulse parameters is forbidden and may lead to unpredictable behavior.

In read and write cycles, the setup and hold time parameters are defined in reference to the address bus. For external devices that require setup and hold time between NCS and NRD signals (read), or between NCS and NWE signals (write), these setup and hold times must be converted into setup and hold times in reference to the address bus.

## 28.9 Scrambling/Unscrambling Function

The external data bus D[7:0] can be scrambled in order to prevent intellectual property data located in off-chip memories from being easily recovered by analyzing data at the package pin level of either microcontroller or memory device.

The scrambling and unscrambling are performed on-the-fly without additional wait states.

The scrambling method depends on two user-configurable key registers, SMC\_KEY1 and SMC\_KEY2. These key registers are only accessible in write mode.

The key must be securely stored in a reliable non-volatile memory in order to recover data from the off-chip memory. Any data scrambled with a given key cannot be recovered if the key is lost.

The scrambling/unscrambling function can be enabled or disabled by programming the SMC\_OCMS register.

When multiple chip selects are handled, it is possible to configure the scrambling function per chip select using the OCMS field in the SMC\_OCMS registers.

## 28.10 Automatic Wait States

Under certain circumstances, the SMC automatically inserts idle cycles between accesses to avoid bus contention or operation conflict.

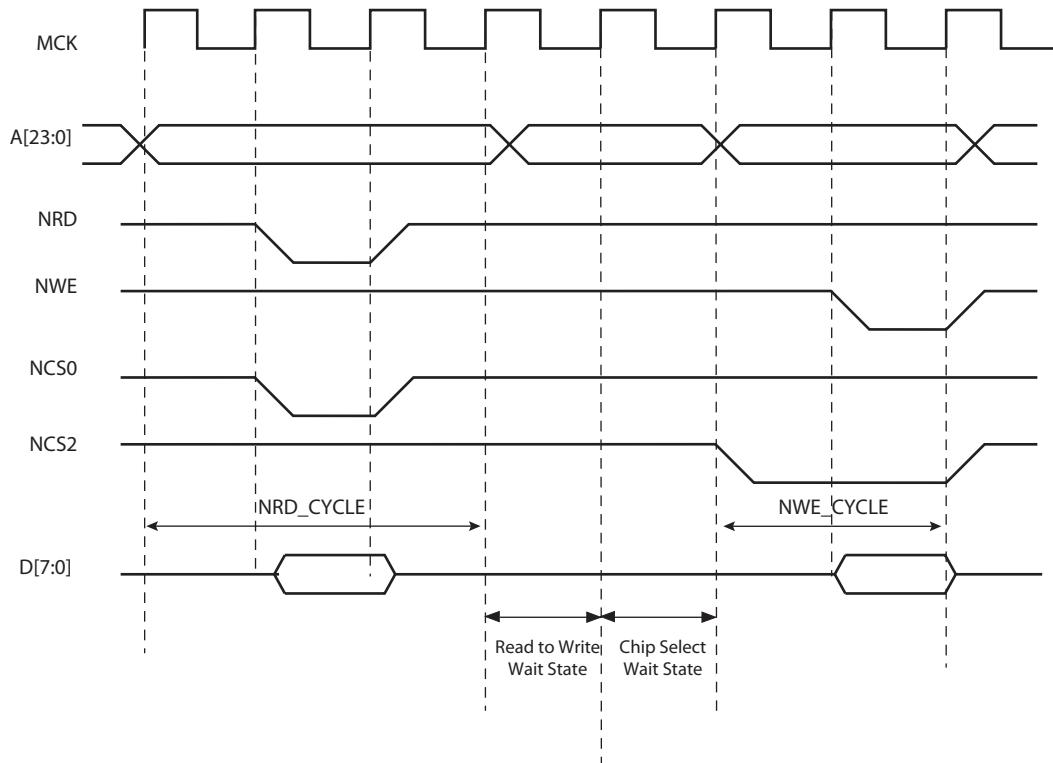
### 28.10.1 Chip Select Wait States

The SMC always inserts an idle cycle between 2 transfers on separate chip selects. This idle cycle ensures that there is no bus contention between the de-activation of one device and the activation of the next one.

During chip select wait state, all control lines are turned inactive: NWR, NCS[0..3], NRD lines are all set to 1.

[Figure 28-13](#) illustrates a chip select wait state between access on Chip Select 0 and Chip Select 2.

**Figure 28-13. Chip Select Wait State between a Read Access on NCS0 and a Write Access on NCS2**



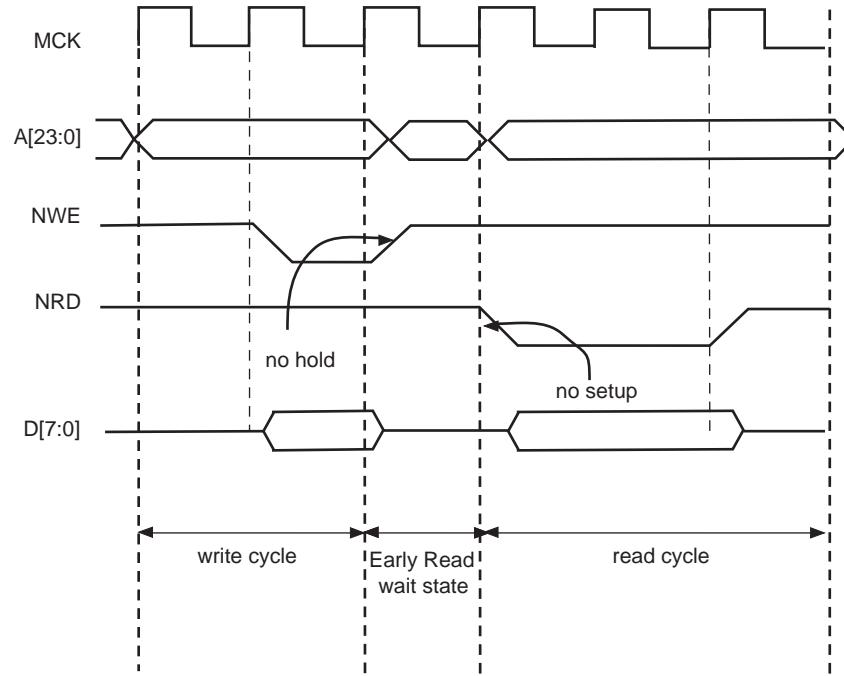
### 28.10.2 Early Read Wait State

In some cases, the SMC inserts a wait state cycle between a write access and a read access to allow time for the write cycle to end before the subsequent read cycle begins. This wait state is not generated in addition to a chip select wait state. The early read cycle thus only occurs between a write and read access to the same memory device (same chip select).

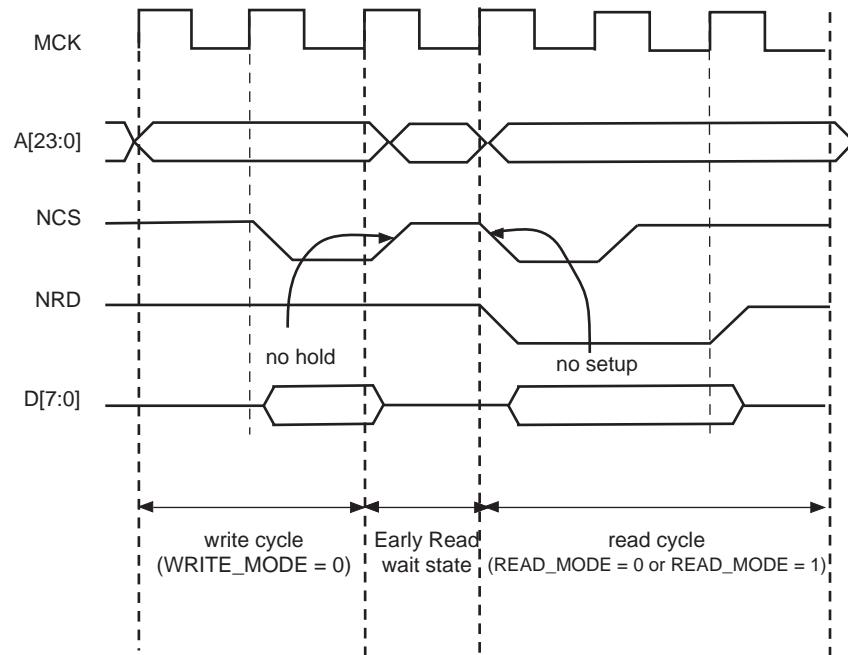
An early read wait state is automatically inserted if at least one of the following conditions is valid:

- if the write controlling signal has no hold time and the read controlling signal has no setup time ([Figure 28-14](#)).
- in NCS write controlled mode (`WRITE_MODE = 0`), if there is no hold timing on the NCS signal and the `NCS_RD_SETUP` parameter is set to 0, regardless of the read mode ([Figure 28-15](#)). The write operation must end with a NCS rising edge. Without an Early Read Wait State, the write operation could not complete properly.
- in NWE controlled mode (`WRITE_MODE = 1`) and if there is no hold timing (`NWE_HOLD = 0`), the feedback of the write control signal is used to control address, data, and chip select lines. If the external write control signal is not inactivated as expected due to load capacitances, an Early Read Wait State is inserted and address, data and control signals are maintained one more cycle. See [Figure 28-16](#).

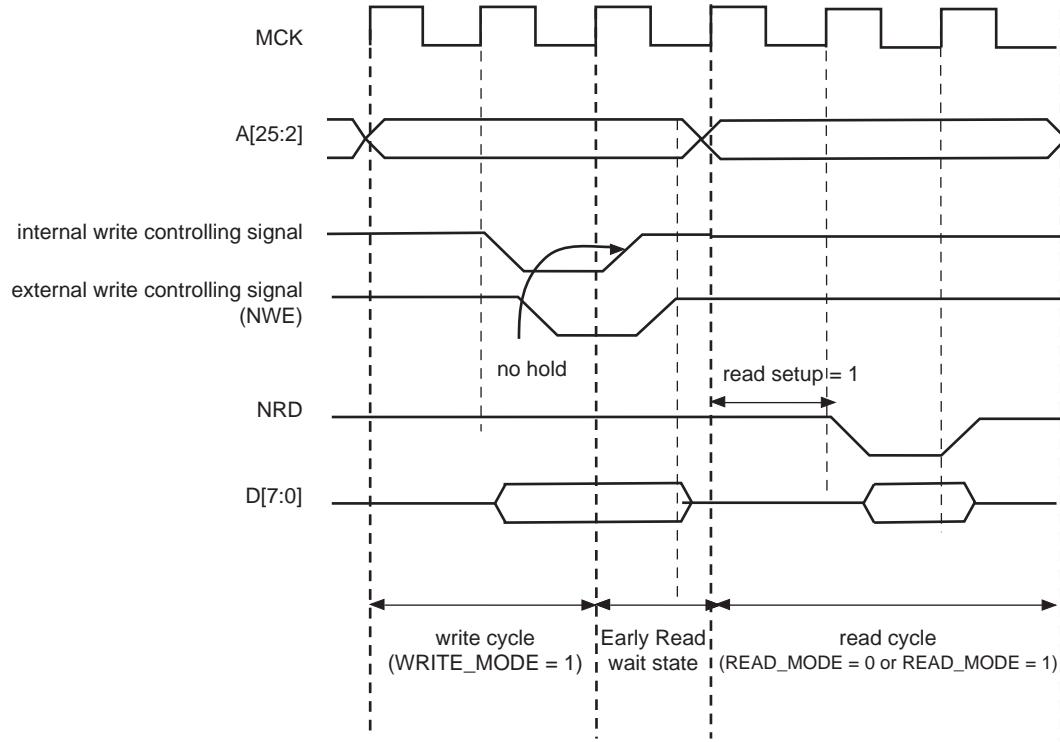
**Figure 28-14. Early Read Wait State: Write with No Hold Followed by Read with No Setup**



**Figure 28-15. Early Read Wait State: NCS Controlled Write with No Hold Followed by a Read with No NCS Setup**



**Figure 28-16. Early Read Wait State: NWE-controlled Write with No Hold Followed by a Read with one Set-up Cycle**



### 28.10.3 Reload User Configuration Wait State

The user may change any of the configuration parameters by writing the SMC user interface.

When detecting that a new user configuration has been written in the user interface, the SMC inserts a wait state before starting the next access. The so called “Reload User Configuration Wait State” is used by the SMC to load the new set of parameters to apply to next accesses.

The Reload Configuration Wait State is not applied in addition to the Chip Select Wait State. If accesses before and after re-programming the user interface are made to different devices (Chip Selects), then one single Chip Select Wait State is applied.

On the other hand, if accesses before and after writing the user interface are made to the same device, a Reload Configuration Wait State is inserted, even if the change does not concern the current Chip Select.

#### 28.10.3.1 User Procedure

To insert a Reload Configuration Wait State, the SMC detects a write access to any SMC\_MODE register of the user interface. If the user only modifies timing registers (SMC\_SETUP, SMC\_PULSE, SMC\_CYCLE registers) in the user interface, he must validate the modification by writing the SMC\_MODE, even if no change was made on the mode parameters.

The user must not change the configuration parameters of an SMC Chip Select (Setup, Pulse, Cycle, Mode) if accesses are performed on this CS during the modification. Any change of the Chip Select parameters, while fetching the code from a memory connected on this CS, may lead to unpredictable behavior. The instructions used to modify the parameters of an SMC Chip Select can be executed from the internal RAM or from a memory connected to another CS.

### 28.10.3.2 Slow Clock Mode Transition

A Reload Configuration Wait State is also inserted when the Slow Clock Mode is entered or exited, after the end of the current transfer (see “[Slow Clock Mode](#)” on page 498).

### 28.10.4 Read to Write Wait State

Due to an internal mechanism, a wait cycle is always inserted between consecutive read and write SMC accesses.

This wait cycle is referred to as a read to write wait state in this document.

This wait cycle is applied in addition to chip select and reload user configuration wait states when they are to be inserted.

See [Figure 28-13 on page 484](#).

## 28.11 Data Float Wait States

Some memory devices are slow to release the external bus. For such devices, it is necessary to add wait states (data float wait states) after a read access:

- before starting a read access to a different external memory
- before starting a write access to the same device or to a different external one.

The Data Float Output Time ( $t_{DF}$ ) for each external memory device is programmed in the TDF\_CYCLES field of the SMC\_MODE register for the corresponding chip select. The value of TDF\_CYCLES indicates the number of data float wait cycles (between 0 and 15) before the external device releases the bus, and represents the time allowed for the data output to go to high impedance after the memory is disabled.

Data float wait states do not delay internal memory accesses. Hence, a single access to an external memory with long  $t_{DF}$  will not slow down the execution of a program from internal memory.

The data float wait states management depends on the READ\_MODE and the TDF\_MODE fields of the SMC\_MODE register for the corresponding chip select.

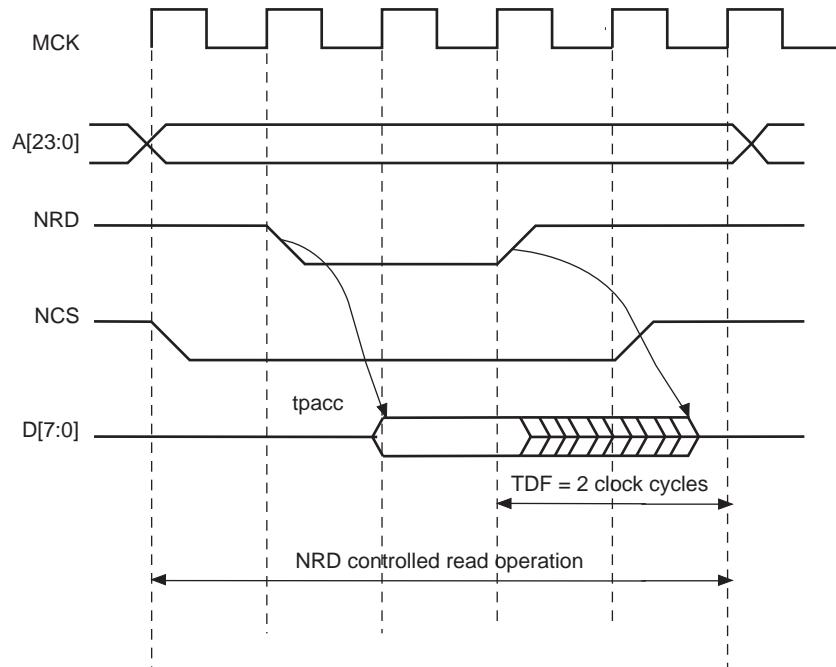
### 28.11.1 READ\_MODE

Setting the READ\_MODE to 1 indicates to the SMC that the NRD signal is responsible for turning off the tri-state buffers of the external memory device. The Data Float Period then begins after the rising edge of the NRD signal and lasts TDF\_CYCLES MCK cycles.

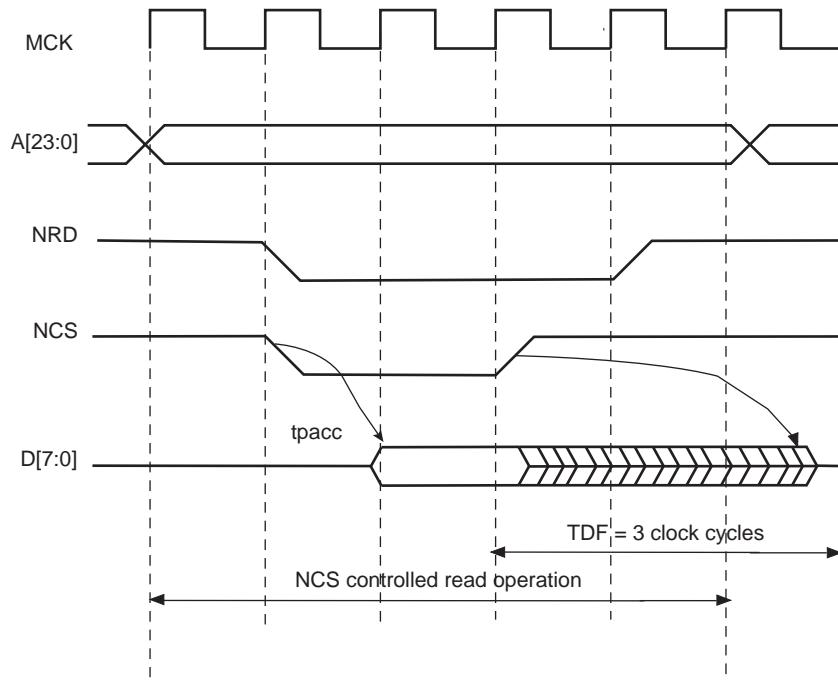
When the read operation is controlled by the NCS signal (READ\_MODE = 0), the TDF field gives the number of MCK cycles during which the data bus remains busy after the rising edge of NCS.

[Figure 28-17](#) illustrates the Data Float Period in NRD-controlled mode (READ\_MODE = 1), assuming a data float period of 2 cycles (TDF\_CYCLES = 2). [Figure 28-18](#) shows the read operation when controlled by NCS (READ\_MODE = 0) and the TDF\_CYCLES parameter equals 3.

**Figure 28-17. TDF Period in NRD Controlled Read Access (TDF = 2)**



**Figure 28-18. TDF Period in NCS Controlled Read Operation (TDF = 3)**



### 28.11.2 TDF Optimization Enabled (TDF\_MODE = 1)

When the TDF\_MODE of the SMC\_MODE register is set to 1 (TDF optimization is enabled), the SMC takes advantage of the setup period of the next access to optimize the number of wait states cycle to insert.

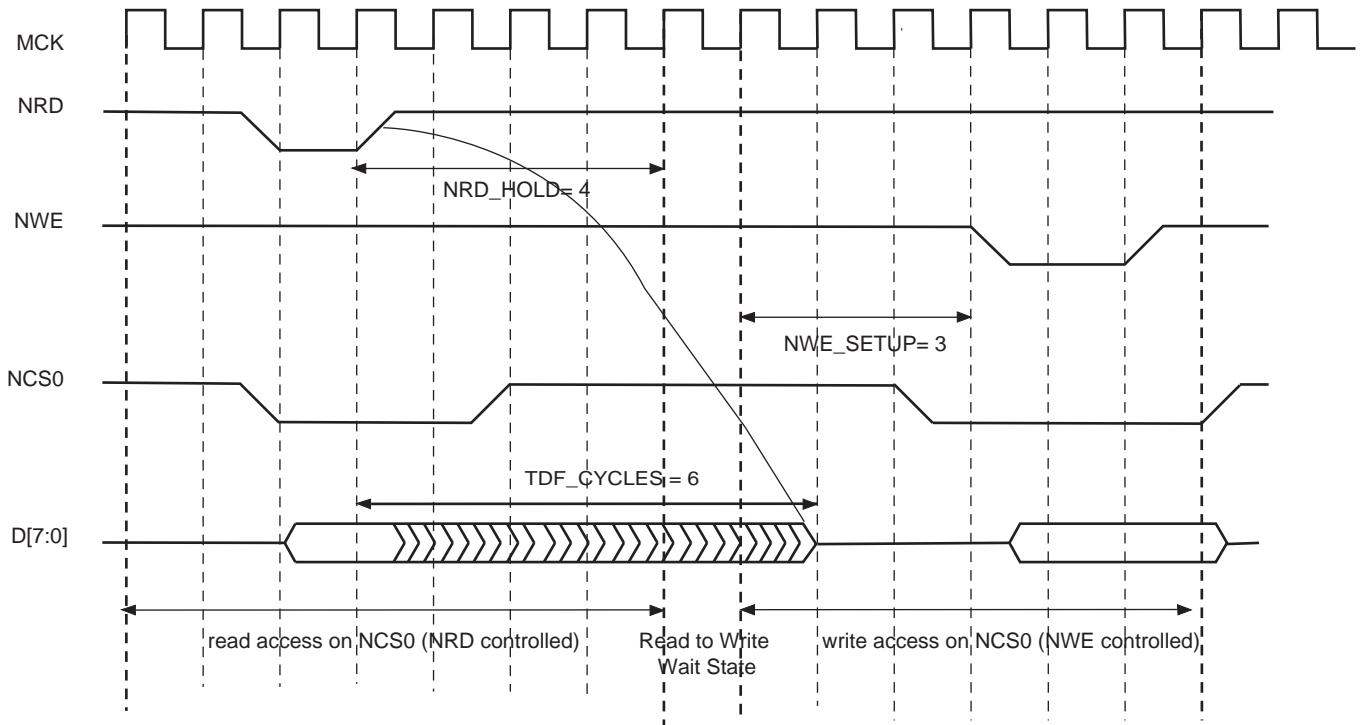
[Figure 28-19](#) shows a read access controlled by NRD, followed by a write access controlled by NWE, on Chip Select 0. Chip Select 0 has been programmed with:

NRD\_HOLD = 4; READ\_MODE = 1 (NRD controlled)

NWE\_SETUP = 3; WRITE\_MODE = 1 (NWE controlled)

TDF\_CYCLES = 6; TDF\_MODE = 1 (optimization enabled).

**Figure 28-19. TDF Optimization: No TDF wait states are inserted if the TDF period is over when the next access begins**



### 28.11.3 TDF Optimization Disabled (TDF\_MODE = 0)

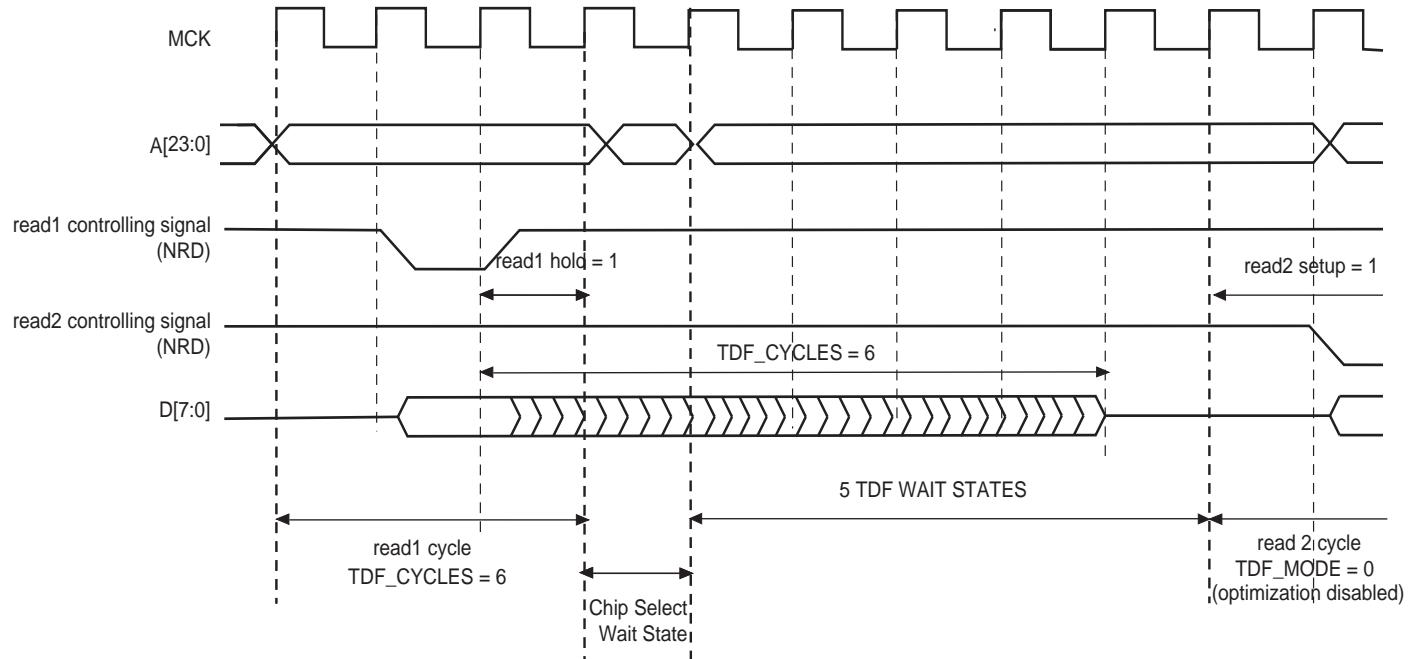
When optimization is disabled, TDF wait states are inserted at the end of the read transfer, so that the data float period is ended when the second access begins. If the hold period of the read1 controlling signal overlaps the data float period, no additional TDF wait states will be inserted.

Figure 28-20, Figure 28-21 and Figure 28-22 illustrate the cases:

- read access followed by a read access on another chip select,
- read access followed by a write access on another chip select,
- read access followed by a write access on the same chip select,

with no TDF optimization.

**Figure 28-20. TDF Optimization Disabled (TDF Mode = 0). TDF wait states between 2 read accesses on different chip selects**



**Figure 28-21. TDF Mode = 0: TDF wait states between a read and a write access on different chip selects**

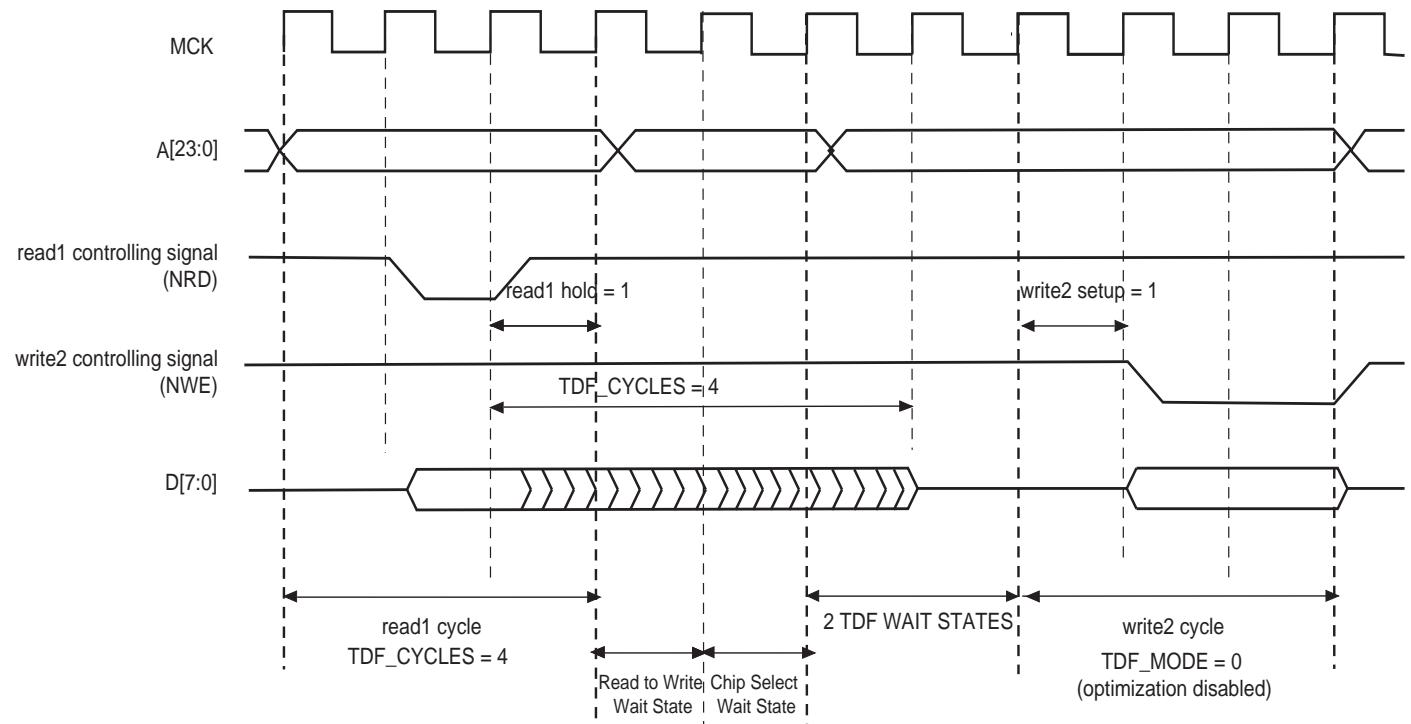
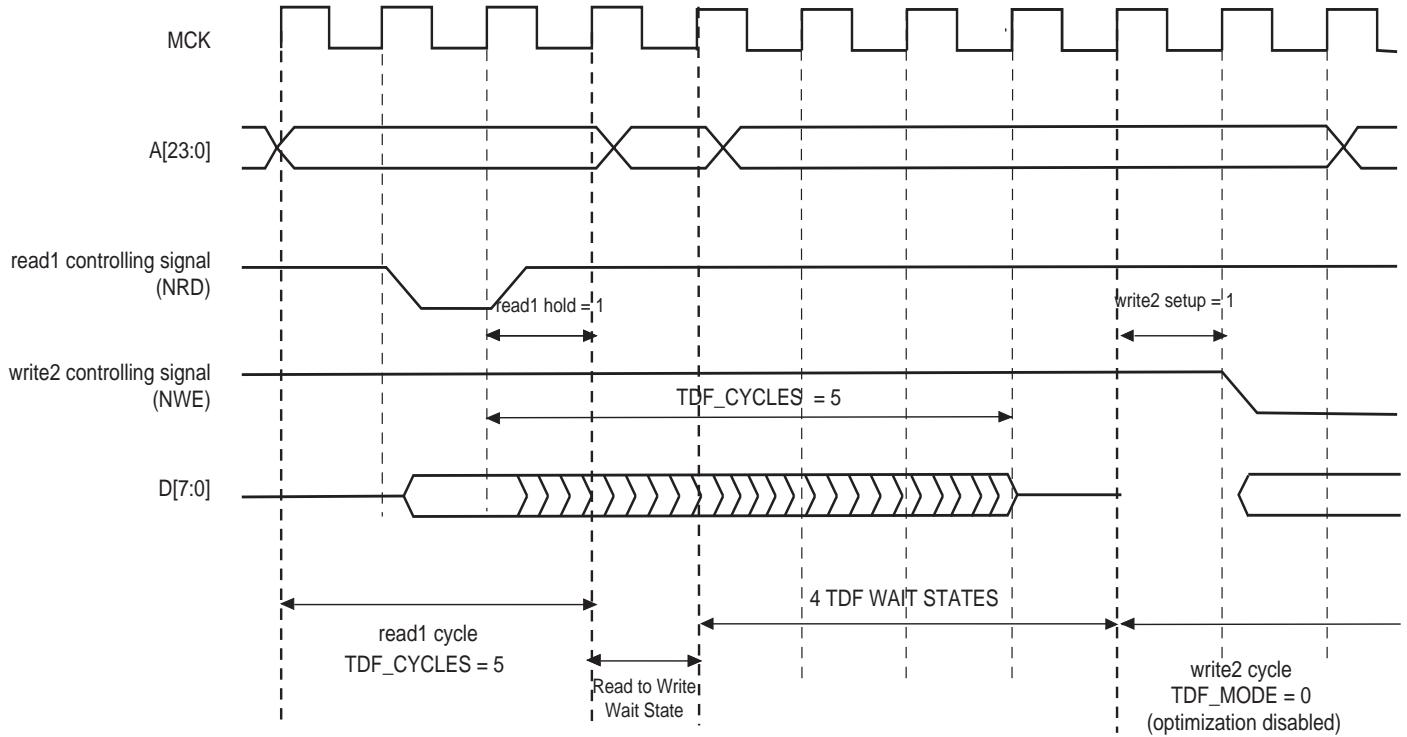


Figure 28-22. TDF Mode = 0: TDF wait states between read and write accesses on the same chip select



## 28.12 External Wait

Any access can be extended by an external device using the NWAIT input signal of the SMC. The EXNW\_MODE field of the SMC\_MODE register on the corresponding chip select must be set to either “10” (frozen mode) or “11” (ready mode). When the EXNW\_MODE is set to “00” (disabled), the NWAIT signal is simply ignored on the corresponding chip select. The NWAIT signal delays the read or write operation in regards to the read or write controlling signal, depending on the read and write modes of the corresponding chip select.

### 28.12.1 Restriction

When one of the EXNW\_MODE is enabled, it is mandatory to program at least one hold cycle for the read/write controlling signal. For that reason, the NWAIT signal cannot be used in Page Mode ([“Asynchronous Page Mode” on page 500](#)), or in Slow Clock Mode ([“Slow Clock Mode” on page 498](#)).

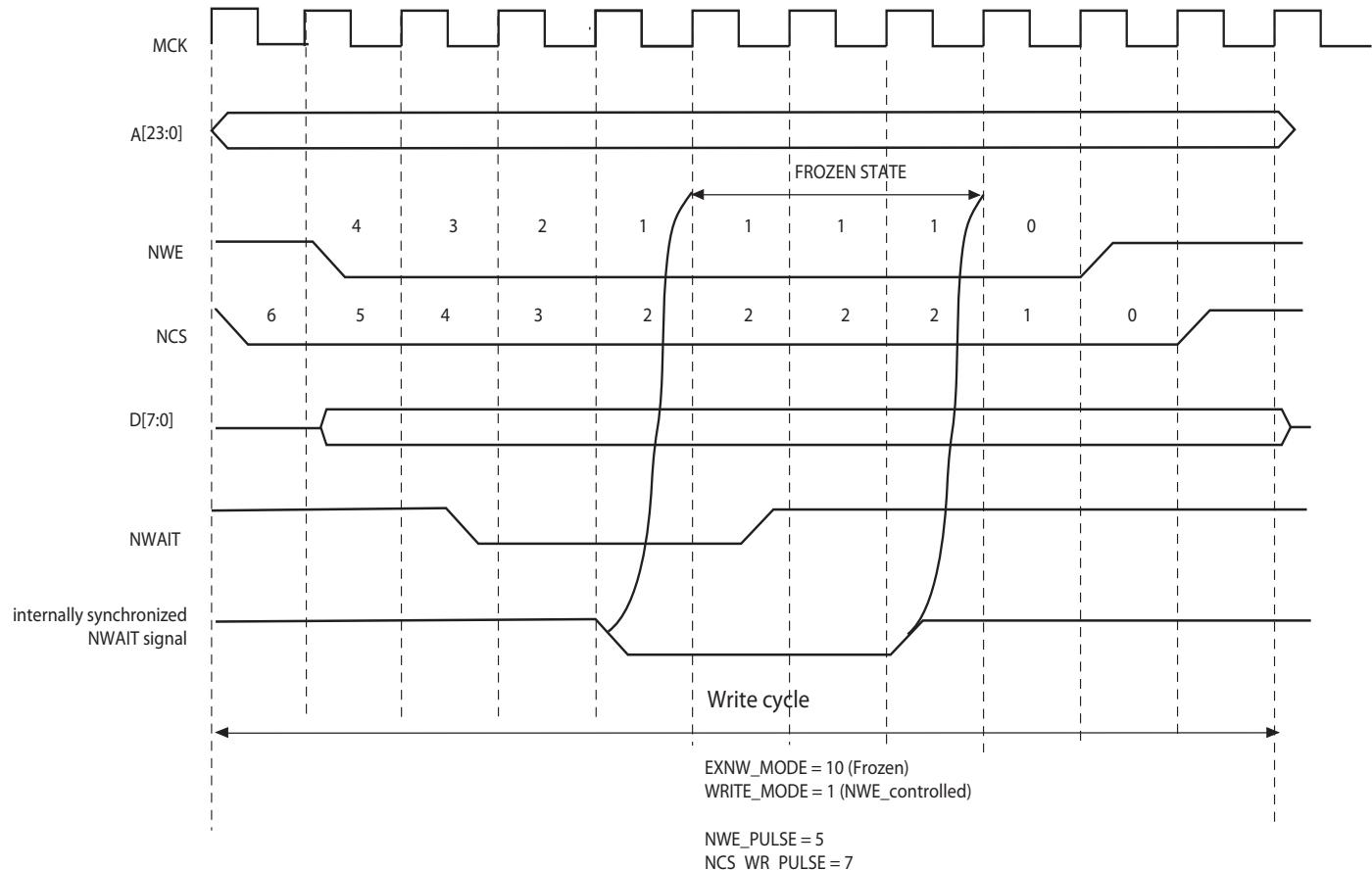
The NWAIT signal is assumed to be a response of the external device to the read/write request of the SMC. Then NWAIT is examined by the SMC only in the pulse state of the read or write controlling signal. The assertion of the NWAIT signal outside the expected period has no impact on SMC behavior.

## 28.12.2 Frozen Mode

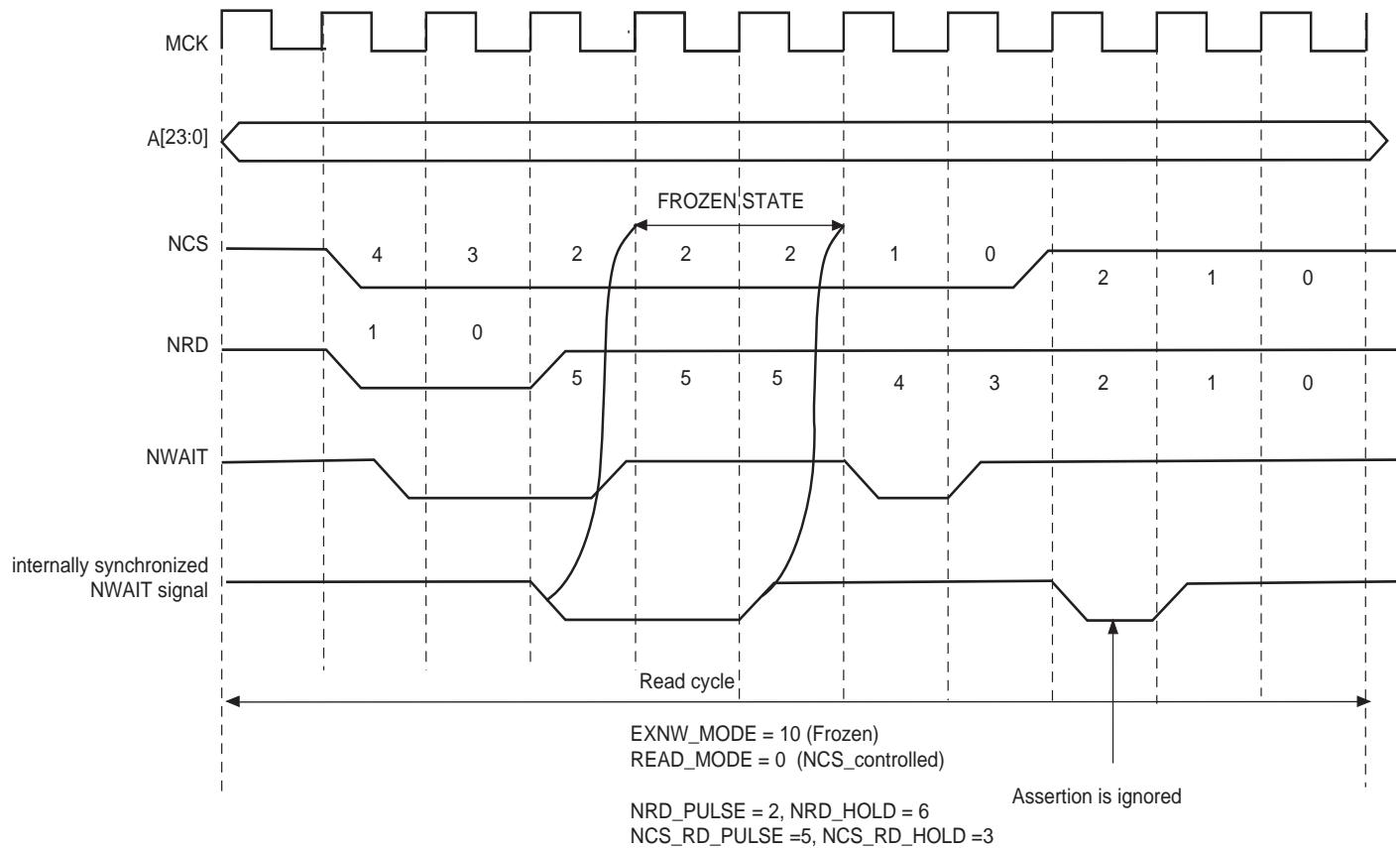
When the external device asserts the NWAIT signal (active low), and after internal synchronization of this signal, the SMC state is frozen, i.e., SMC internal counters are frozen, and all control signals remain unchanged. When the resynchronized NWAIT signal is deasserted, the SMC completes the access, resuming the access from the point where it was stopped. See [Figure 28-23](#). This mode must be selected when the external device uses the NWAIT signal to delay the access and to freeze the SMC.

The assertion of the NWAIT signal outside the expected period is ignored as illustrated in [Figure 28-24](#).

**Figure 28-23. Write Access with NWAIT Assertion in Frozen Mode (EXNW\_MODE = 10)**



**Figure 28-24. Read Access with NWAIT Assertion in Frozen Mode (EXNW\_MODE = 10)**



### 28.12.3 Ready Mode

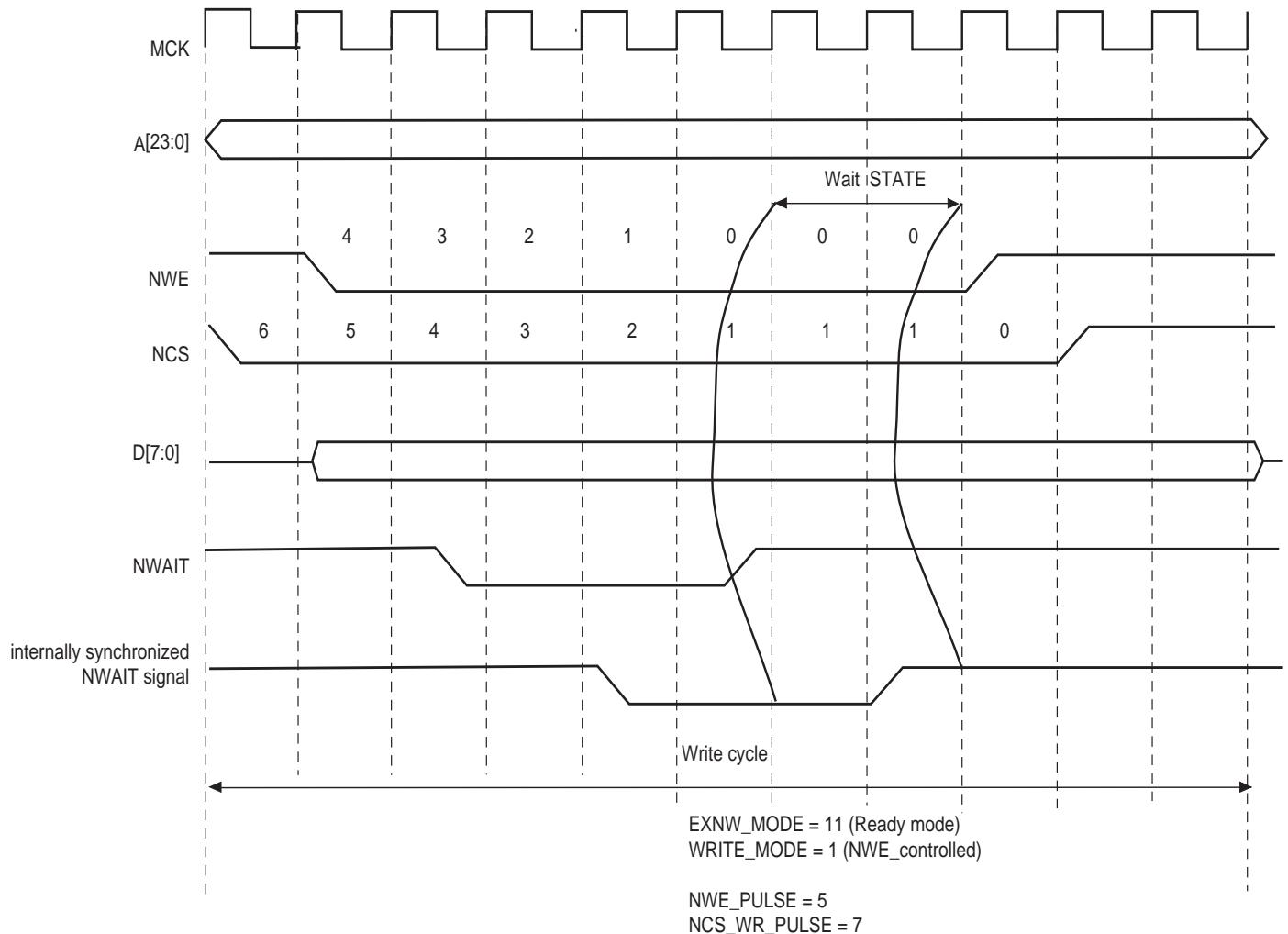
In Ready mode (`EXNW_MODE = 11`), the SMC behaves differently. Normally, the SMC begins the access by down counting the setup and pulse counters of the read/write controlling signal. In the last cycle of the pulse phase, the resynchronized NWAIT signal is examined.

If asserted, the SMC suspends the access as shown in [Figure 28-25](#) and [Figure 28-26](#). After deassertion, the access is completed: the hold step of the access is performed.

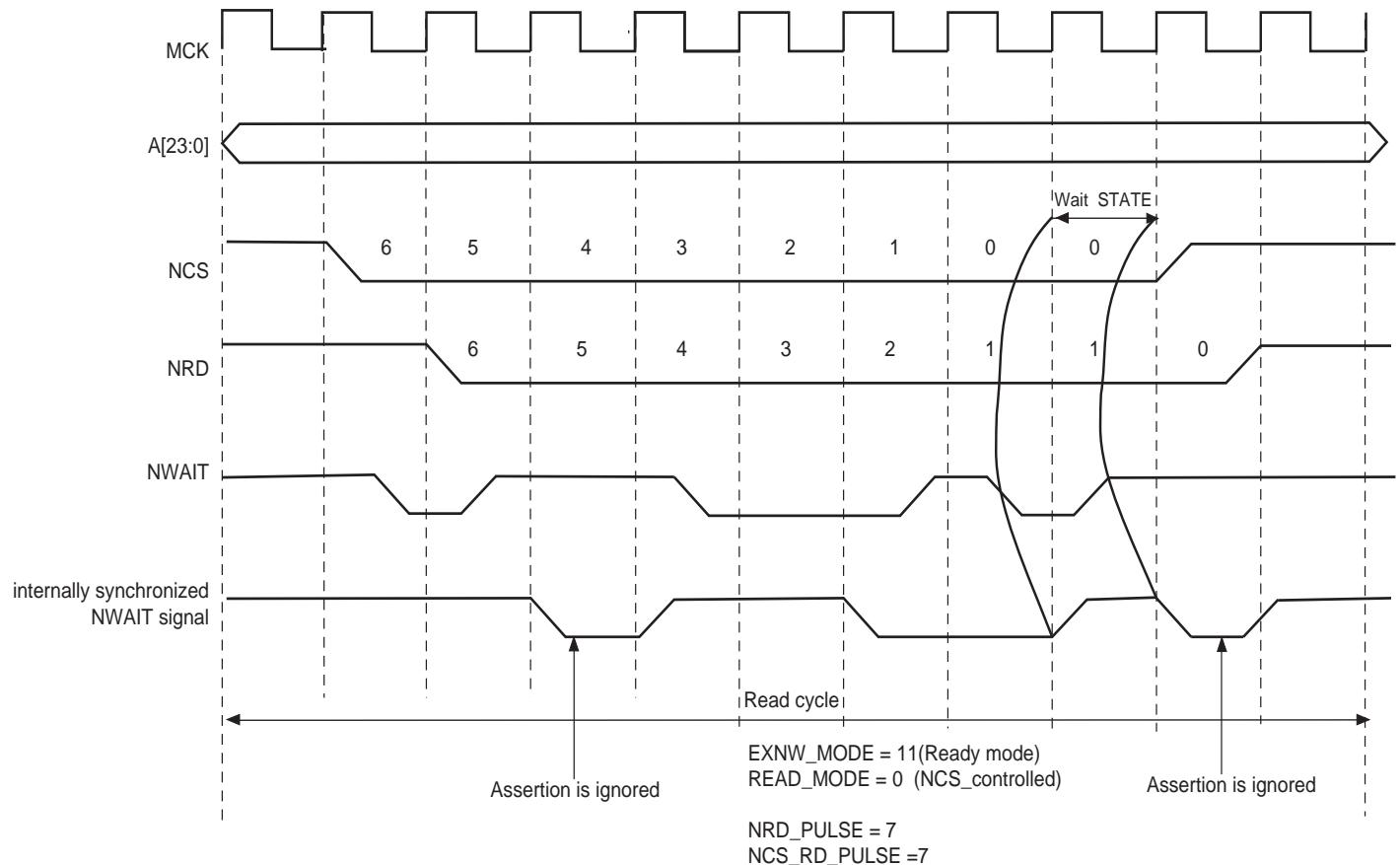
This mode must be selected when the external device uses deassertion of the NWAIT signal to indicate its ability to complete the read or write operation.

If the NWAIT signal is deasserted before the end of the pulse, or asserted after the end of the pulse of the controlling read/write signal, it has no impact on the access length as shown in [Figure 28-26](#).

**Figure 28-25. NWAIT Assertion in Write Access: Ready Mode (`EXNW_MODE = 11`)**



**Figure 28-26. NWAIT Assertion in Read Access: Ready Mode (EXNW\_MODE = 11)**



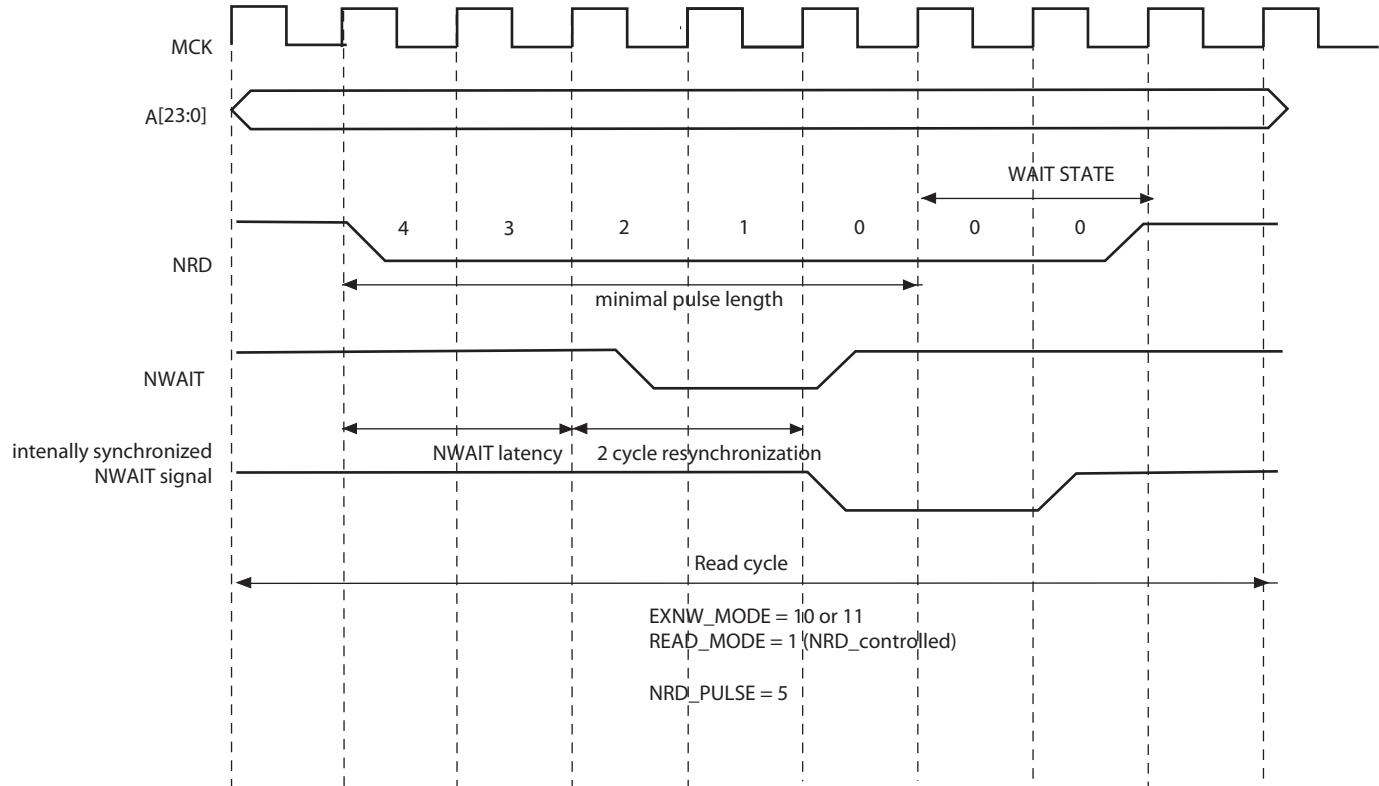
#### 28.12.4 NWAIT Latency and Read/Write Timings

There may be a latency between the assertion of the read/write controlling signal and the assertion of the NWAIT signal by the device. The programmed pulse length of the read/write controlling signal must be at least equal to this latency plus the 2 cycles of resynchronization + 1 cycle. Otherwise, the SMC may enter the hold state of the access without detecting the NWAIT signal assertion. This is true in frozen mode as well as in ready mode. This is illustrated on [Figure 28-27](#).

When EXNW\_MODE is enabled (ready or frozen), the user must program a pulse length of the read and write controlling signal of at least:

$$\text{minimal pulse length} = \text{NWAIT latency} + 2 \text{ resynchronization cycles} + 1 \text{ cycle}$$

**Figure 28-27. NWAIT Latency**



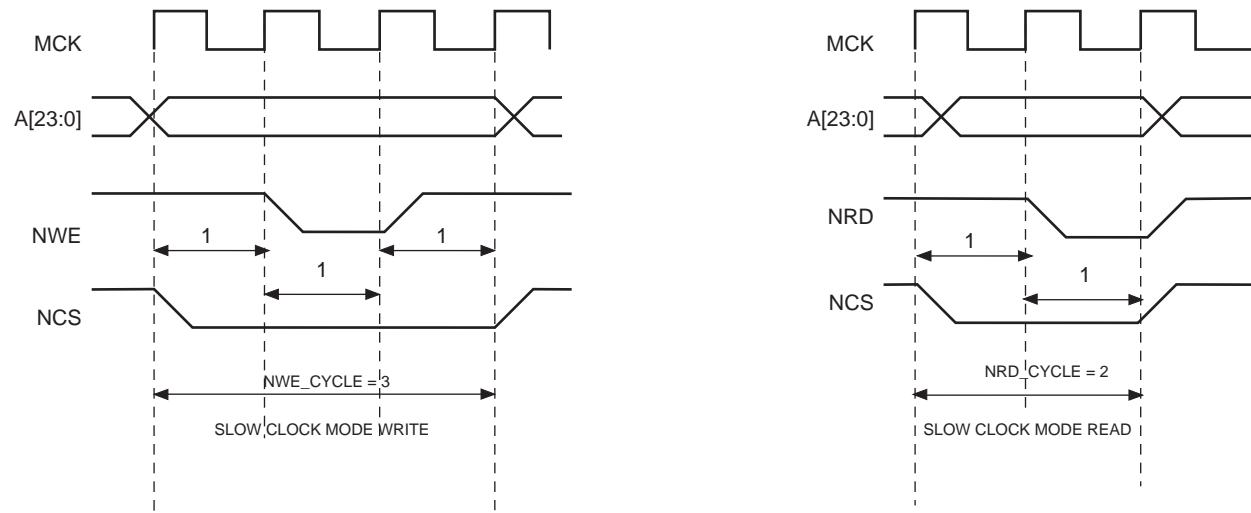
## 28.13 Slow Clock Mode

The SMC is able to automatically apply a set of “slow clock mode” read/write waveforms when an internal signal driven by the Power Management Controller is asserted because MCK has been turned to a very slow clock rate (typically 32 kHz clock rate). In this mode, the user-programmed waveforms are ignored and the slow clock mode waveforms are applied. This mode is provided so as to avoid reprogramming the User Interface with appropriate waveforms at very slow clock rate. When activated, the slow mode is active on all chip selects.

### 28.13.1 Slow Clock Mode Waveforms

[Figure 28-28](#) illustrates the read and write operations in slow clock mode. They are valid on all chip selects. [Table 28-4](#) indicates the value of read and write parameters in slow clock mode.

**Figure 28-28. Read/Write Cycles in Slow Clock Mode**



**Table 28-4. Read and Write Timing Parameters in Slow Clock Mode**

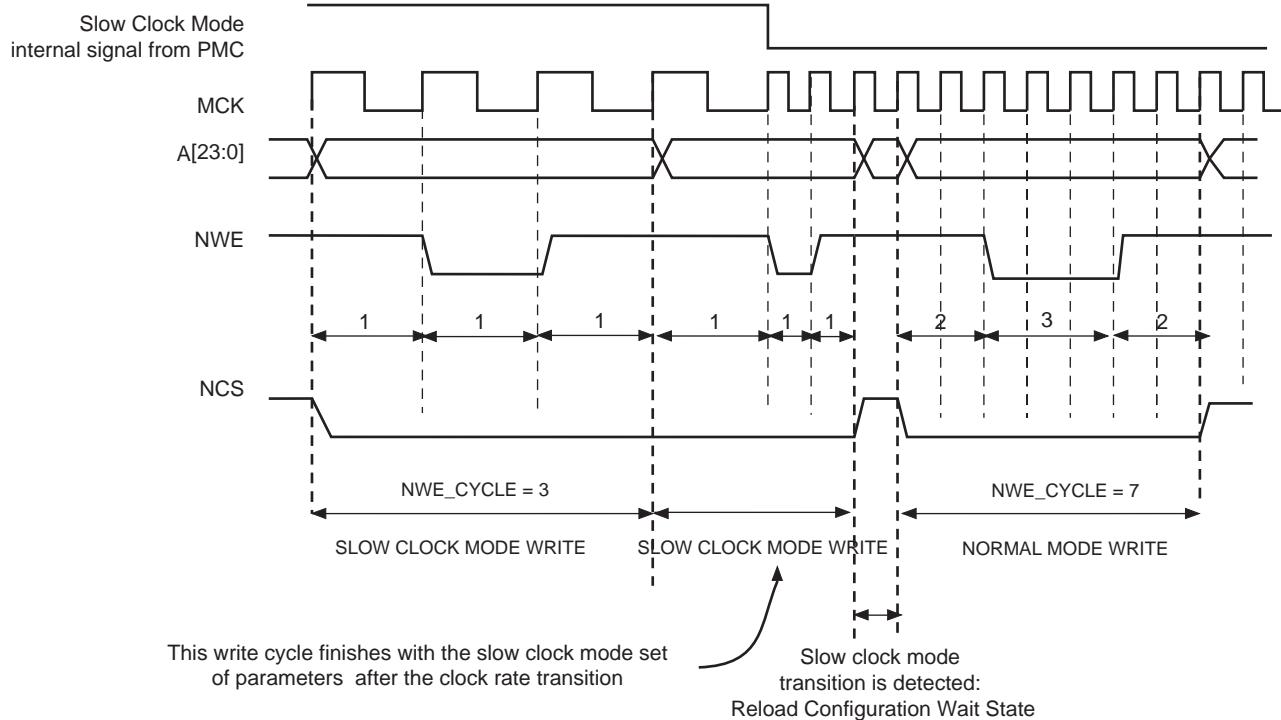
Read Parameters	Duration (cycles)	Write Parameters	Duration (cycles)
NRD_SETUP	1	NWE_SETUP	1
NRD_PULSE	1	NWE_PULSE	1
NCS_RD_SETUP	0	NCS_WR_SETUP	0
NCS_RD_PULSE	2	NCS_WR_PULSE	3
NRD_CYCLE	2	NWE_CYCLE	3

### 28.13.2 Switching from (to) Slow Clock Mode to (from) Normal Mode

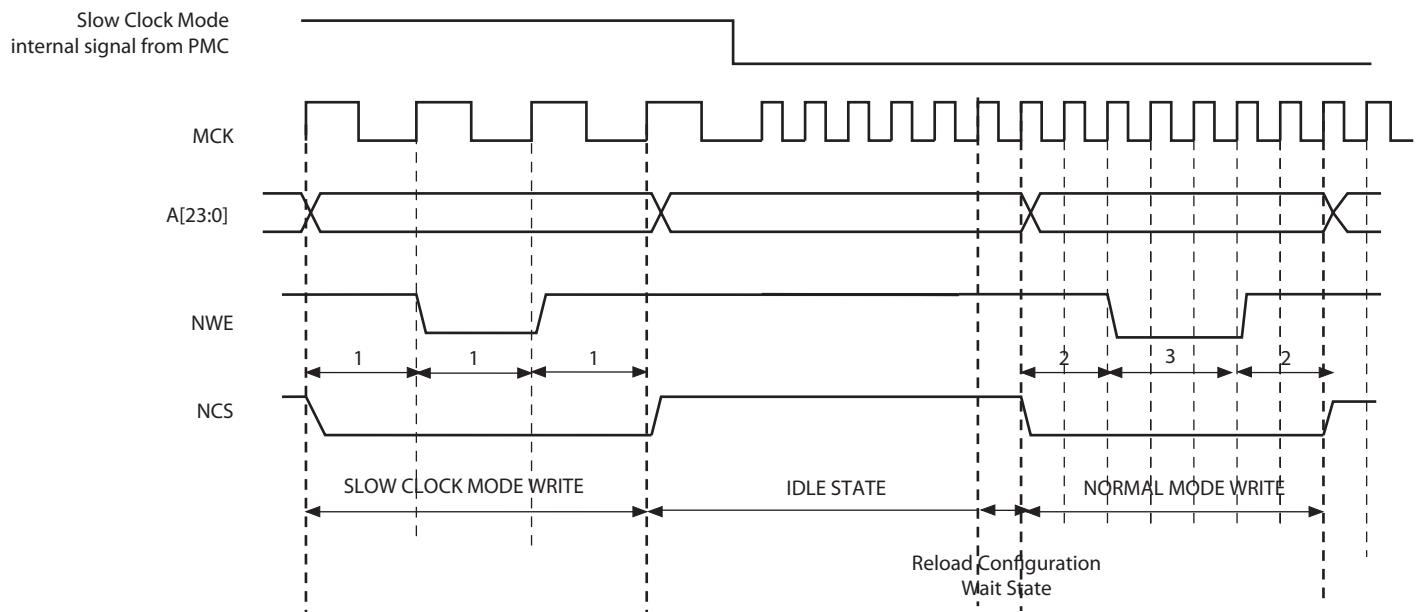
When switching from slow clock mode to the normal mode, the current slow clock mode transfer is completed at high clock rate, with the set of slow clock mode parameters. See [Figure 28-29 on page 499](#). The external device may not be fast enough to support such timings.

[Figure 28-30](#) illustrates the recommended procedure to properly switch from one mode to the other.

**Figure 28-29. Clock Rate Transition Occurs while the SMC is Performing a Write Operation**



**Figure 28-30. Recommended Procedure to Switch from Slow Clock Mode to Normal Mode or from Normal Mode to Slow Clock Mode**



## 28.14 Asynchronous Page Mode

The SMC supports asynchronous burst reads in page mode, providing that the page mode is enabled in the SMC\_MODE register (PMEN field). The page size must be configured in the SMC\_MODE register (PS field) to 4, 8, 16 or 32 bytes.

The page defines a set of consecutive bytes into memory. A 4-byte page (resp. 8-, 16-, 32-byte page) is always aligned to 4-byte boundaries (resp. 8-, 16-, 32-byte boundaries) of memory. The MSB of data address defines the address of the page in memory, the LSB of address define the address of the data in the page as detailed in [Table 28-5](#).

With page mode memory devices, the first access to one page ( $t_{pa}$ ) takes longer than the subsequent accesses to the page ( $t_{sa}$ ) as shown in [Figure 28-31](#). When in page mode, the SMC enables the user to define different read timings for the first access within one page, and next accesses within the page.

**Table 28-5. Page Address and Data Address within a Page**

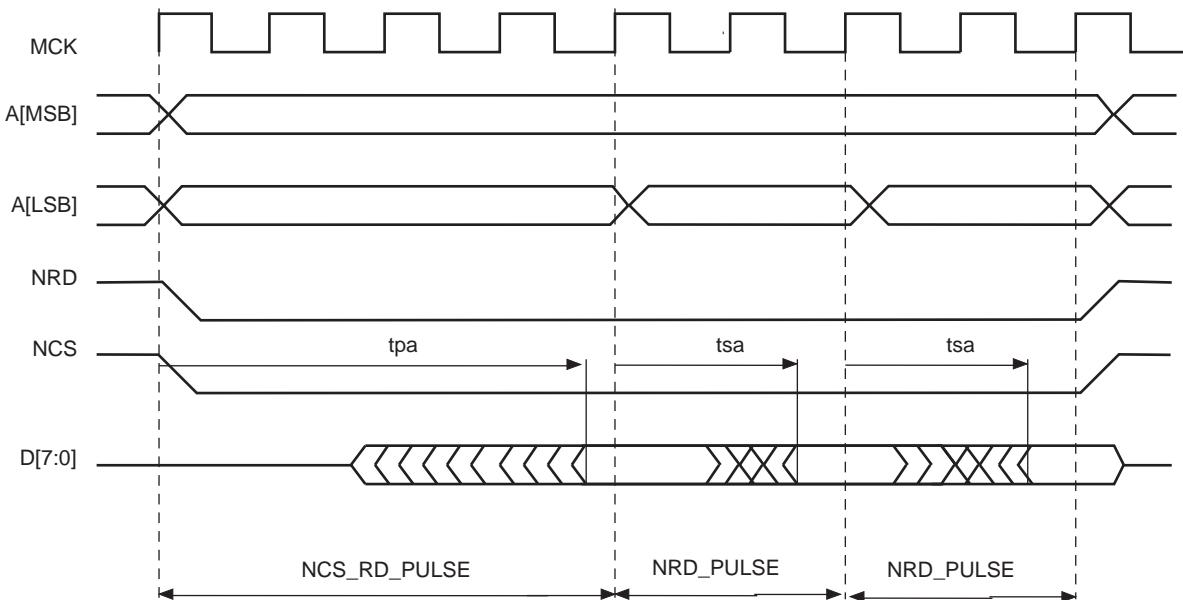
Page Size	Page Address <sup>(1)</sup>	Data Address in the Page
4 bytes	A[23:2]	A[1:0]
8 bytes	A[23:3]	A[2:0]
16 bytes	A[23:4]	A[3:0]
32 bytes	A[23:5]	A[4:0]

Note: 1. "A" denotes the address bus of the memory device.

### 28.14.1 Protocol and Timings in Page Mode

[Figure 28-31](#) shows the NRD and NCS timings in page mode access.

**Figure 28-31. Page Mode Read Protocol (Address MSB and LSB are defined in [Table 28-5](#))**



The NRD and NCS signals are held low during all read transfers, whatever the programmed values of the setup and hold timings in the User Interface may be. Moreover, the NRD and NCS timings are identical. The pulse length of the first access to the page is defined with the NCS\_RD\_PULSE field of the SMC\_PULSE register. The pulse length of subsequent accesses within the page are defined using the NRD\_PULSE parameter.

In page mode, the programming of the read timings is described in [Table 28-6](#):

**Table 28-6. Programming of Read Timings in Page Mode**

Parameter	Value	Definition
READ_MODE	'x'	No impact
NCS_RD_SETUP	'x'	No impact
NCS_RD_PULSE	$t_{pa}$	Access time of first access to the page
NRD_SETUP	'x'	No impact
NRD_PULSE	$t_{sa}$	Access time of subsequent accesses in the page
NRD_CYCLE	'x'	No impact

The SMC does not check the coherency of timings. It will always apply the NCS\_RD\_PULSE timings as page access timing ( $t_{pa}$ ) and the NRD\_PULSE for accesses to the page ( $t_{sa}$ ), even if the programmed value for  $t_{pa}$  is shorter than the programmed value for  $t_{sa}$ .

#### 28.14.2 Page Mode Restriction

The page mode is not compatible with the use of the NWAIT signal. Using the page mode and the NWAIT signal may lead to unpredictable behavior.

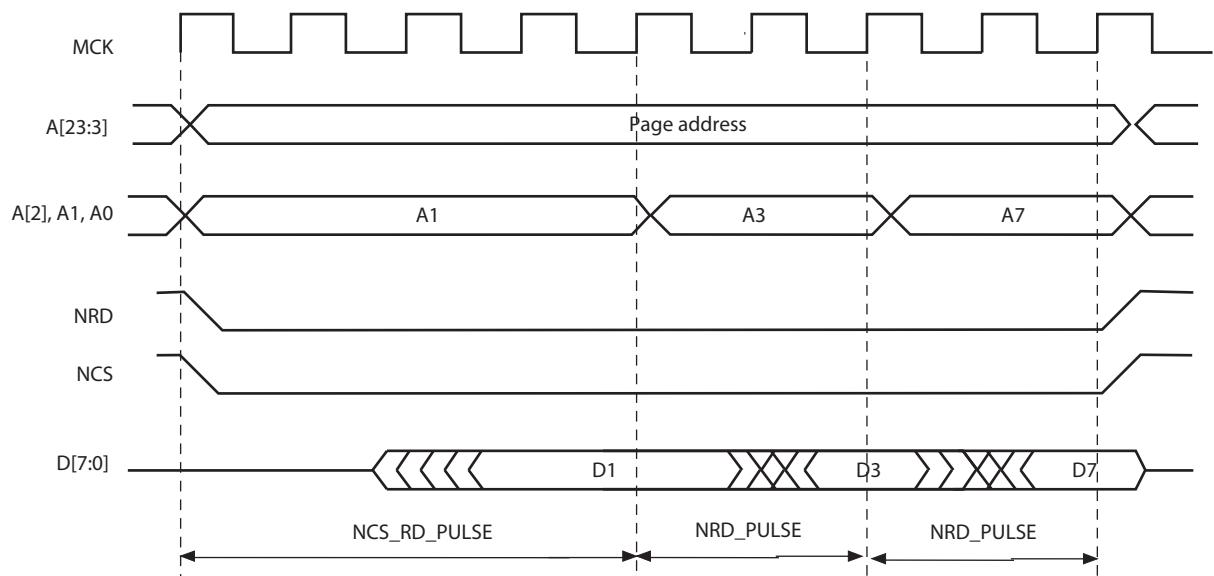
#### 28.14.3 Sequential and Non-sequential Accesses

If the chip select and the MSB of addresses as defined in [Table 28-5](#) are identical, then the current access lies in the same page as the previous one, and no page break occurs.

Using this information, all data within the same page, sequential or not sequential, are accessed with a minimum access time ( $t_{sa}$ ). [Figure 28-32](#) illustrates access to an 8-bit memory device in page mode, with 8-byte pages. Access to D1 causes a page access with a long access time ( $t_{pa}$ ). Accesses to D3 and D7, though they are not sequential accesses, only require a short access time ( $t_{sa}$ ).

If the MSB of addresses are different, the SMC performs the access of a new page. In the same way, if the chip select is different from the previous access, a page break occurs. If two sequential accesses are made to the page mode memory, but separated by an other internal or external peripheral access, a page break occurs on the second access because the chip select of the device was deasserted between both accesses.

**Figure 28-32. Access to Non-Sequential Data within the Same Page**



## 28.15 Static Memory Controller (SMC) User Interface

The SMC is programmed using the registers listed in [Table 28-7](#). For each chip select, a set of 4 registers is used to program the parameters of the external device connected on it. In [Table 28-7](#), “CS\_number” denotes the chip select number. 16 bytes (0x10) are required per chip select.

The user must complete writing the configuration by writing any one of the SMC\_MODE registers.

**Table 28-7. Register Mapping**

Offset	Register	Name	Access	Reset
0x10 x CS_number + 0x00	SMC Setup Register	SMC_SETUP	Read-write	0x01010101
0x10 x CS_number + 0x04	SMC Pulse Register	SMC_PULSE	Read-write	0x01010101
0x10 x CS_number + 0x08	SMC Cycle Register	SMC_CYCLE	Read-write	0x00030003
0x10 x CS_number + 0x0C	SMC Mode Register	SMC_MODE	Read-write	0x10000003
0x80	SMC OCMS MODE Register	SMC_OCMS	Read-write	0x00000000
0x84	SMC OCMS KEY1 Register	SMC_KEY1	Write once	0x00000000
0x88	SMC OCMS KEY2 Register	SMC_KEY2	Write once	0x00000000
0xE4	SMC Write Protect Mode Register	SMC_WPMR	Read-write	0x00000000
0xE8	SMC Write Protect Status Register	SMC_WPSR	Read-only	0x00000000
0xEC-0xFC	Reserved	-	-	-

### 28.15.1 SMC Setup Register

**Name:** SMC\_SETUP[0..3]

**Address:** 0x40060000 [0], 0x40060010 [1], 0x40060020 [2], 0x40060030 [3]

**Access:** Read-write

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

- **NWE\_SETUP: NWE Setup Length**

The NWE signal setup length is defined as:

$$\text{NWE setup length} = (128 * \text{NWE\_SETUP}[5] + \text{NWE\_SETUP}[4:0]) \text{ clock cycles}$$

- **NCS\_WR\_SETUP: NCS Setup Length in WRITE Access**

In write access, the NCS signal setup length is defined as:

$$\text{NCS setup length} = (128 * \text{NCS\_WR\_SETUP}[5] + \text{NCS\_WR\_SETUP}[4:0]) \text{ clock cycles}$$

- **NRD\_SETUP: NRD Setup Length**

The NRD signal setup length is defined in clock cycles as:

$$\text{NRD setup length} = (128 * \text{NRD\_SETUP}[5] + \text{NRD\_SETUP}[4:0]) \text{ clock cycles}$$

- **NCS\_RD\_SETUP: NCS Setup Length in READ Access**

In read access, the NCS signal setup length is defined as:

$$\text{NCS setup length} = (128 * \text{NCS\_RD\_SETUP}[5] + \text{NCS\_RD\_SETUP}[4:0]) \text{ clock cycles}$$

## 28.15.2 SMC Pulse Register

**Name:** SMC\_PULSE[0..3]

**Address:** 0x40060004 [0], 0x40060014 [1], 0x40060024 [2], 0x40060034 [3]

**Access:** Read-write

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

- **NWE\_PULSE: NWE Pulse Length**

The NWE signal pulse length is defined as:

$$\text{NWE pulse length} = (256 * \text{NWE\_PULSE}[6] + \text{NWE\_PULSE}[5:0]) \text{ clock cycles}$$

The NWE pulse length must be at least 1 clock cycle.

- **NCS\_WR\_PULSE: NCS Pulse Length in WRITE Access**

In write access, the NCS signal pulse length is defined as:

$$\text{NCS pulse length} = (256 * \text{NCS\_WR\_PULSE}[6] + \text{NCS\_WR\_PULSE}[5:0]) \text{ clock cycles}$$

The NCS pulse length must be at least 1 clock cycle.

- **NRD\_PULSE: NRD Pulse Length**

In standard read access, the NRD signal pulse length is defined in clock cycles as:

$$\text{NRD pulse length} = (256 * \text{NRD\_PULSE}[6] + \text{NRD\_PULSE}[5:0]) \text{ clock cycles}$$

The NRD pulse length must be at least 1 clock cycle.

In page mode read access, the NRD\_PULSE parameter defines the duration of the subsequent accesses in the page.

- **NCS\_RD\_PULSE: NCS Pulse Length in READ Access**

In standard read access, the NCS signal pulse length is defined as:

$$\text{NCS pulse length} = (256 * \text{NCS\_RD\_PULSE}[6] + \text{NCS\_RD\_PULSE}[5:0]) \text{ clock cycles}$$

The NCS pulse length must be at least 1 clock cycle.

In page mode read access, the NCS\_RD\_PULSE parameter defines the duration of the first access to one page.

### 28.15.3 SMC Cycle Register

**Name:** SMC\_CYCLE[0..3]

**Address:** 0x40060008 [0], 0x40060018 [1], 0x40060028 [2], 0x40060038 [3]

**Access:** Read-write

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

- **NWE\_CYCLE: Total Write Cycle Length**

The total write cycle length is the total duration in clock cycles of the write cycle. It is equal to the sum of the setup, pulse and hold steps of the NWE and NCS signals. It is defined as:

Write cycle length = (NWE\_CYCLE[8:7]\*256 + NWE\_CYCLE[6:0]) clock cycles

- **NRD\_CYCLE: Total Read Cycle Length**

The total read cycle length is the total duration in clock cycles of the read cycle. It is equal to the sum of the setup, pulse and hold steps of the NRD and NCS signals. It is defined as:

Read cycle length = (NRD\_CYCLE[8:7]\*256 + NRD\_CYCLE[6:0]) clock cycles

#### 28.15.4 SMC MODE Register

**Name:** SMC\_MODE[0..3]

**Address:** 0x4006000C [0], 0x4006001C [1], 0x4006002C [2], 0x4006003C [3]

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	PS	–	–	–	–	PMEN
23	22	21	20	19	18	17	16
–	–	–	TDF_MODE	–	–	–	TDF_CYCLES
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	EXNW_MODE	–	–	–	WRITE_MODE	READ_MODE

- **READ\_MODE:**

1: The read operation is controlled by the NRD signal.

- If TDF cycles are programmed, the external bus is marked busy after the rising edge of NRD.
- If TDF optimization is enabled (TDF\_MODE =1), TDF wait states are inserted after the setup of NRD.

0: The read operation is controlled by the NCS signal.

- If TDF cycles are programmed, the external bus is marked busy after the rising edge of NCS.
- If TDF optimization is enabled (TDF\_MODE =1), TDF wait states are inserted after the setup of NCS.

- **WRITE\_MODE**

1: The write operation is controlled by the NWE signal.

- If TDF optimization is enabled (TDF\_MODE =1), TDF wait states will be inserted after the setup of NWE.

0: The write operation is controlled by the NCS signal.

- If TDF optimization is enabled (TDF\_MODE =1), TDF wait states will be inserted after the setup of NCS.

- **EXNW\_MODE: NWAIT Mode**

The NWAIT signal is used to extend the current read or write signal. It is only taken into account during the pulse phase of the read and write controlling signal. When the use of NWAIT is enabled, at least one cycle hold duration must be programmed for the read and write controlling signal.

Value	Name	Description
0	DISABLED	Disabled
1		Reserved
2	FROZEN	Frozen Mode
3	READY	Ready Mode

- Disabled Mode: The NWAIT input signal is ignored on the corresponding Chip Select.
- Frozen Mode: If asserted, the NWAIT signal freezes the current read or write cycle. After deassertion, the read/write cycle is resumed from the point where it was stopped.
- Ready Mode: The NWAIT signal indicates the availability of the external device at the end of the pulse of the controlling read or write signal, to complete the access. If high, the access normally completes. If low, the access is extended until NWAIT returns high.

- **TDF\_CYCLES: Data Float Time**

This field gives the integer number of clock cycles required by the external device to release the data after the rising edge of the read controlling signal. The SMC always provide one full cycle of bus turnaround after the TDF\_CYCLES period. The external bus cannot be used by another chip select during TDF\_CYCLES + 1 cycles. From 0 up to 15 TDF\_CYCLES can be set.

- **TDF\_MODE: TDF Optimization**

1: TDF optimization is enabled.

- The number of TDF wait states is optimized using the setup period of the next read/write access.

0: TDF optimization is disabled.

- The number of TDF wait states is inserted before the next access begins.

- **PMEN: Page Mode Enabled**

1: Asynchronous burst read in page mode is applied on the corresponding chip select.

0: Standard read is applied.

- **PS: Page Size**

If page mode is enabled, this field indicates the size of the page in bytes.

Value	Name	Description
0	4_BYT E	4-byte page
1	8_BYT E	8-byte page
2	16_BYT E	16-byte page
3	32_BYT E	32-byte page

### 28.15.5 SMC OCMS Mode Register

Name: SMC\_OCMS

Address: 0x40060080

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	CS3SE	CS2SE	CS1SE	CS0SE
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	SMSE

- **CSxSE: Chip Select (x = 0 to 3) Scrambling Enable**

0: Disable Scrambling for CSx.

1: Enable Scrambling for CSx.

- **SMSE: Static Memory Controller Scrambling Enable**

0: Disable Scrambling for SMC access.

1: Enable Scrambling for SMC access.

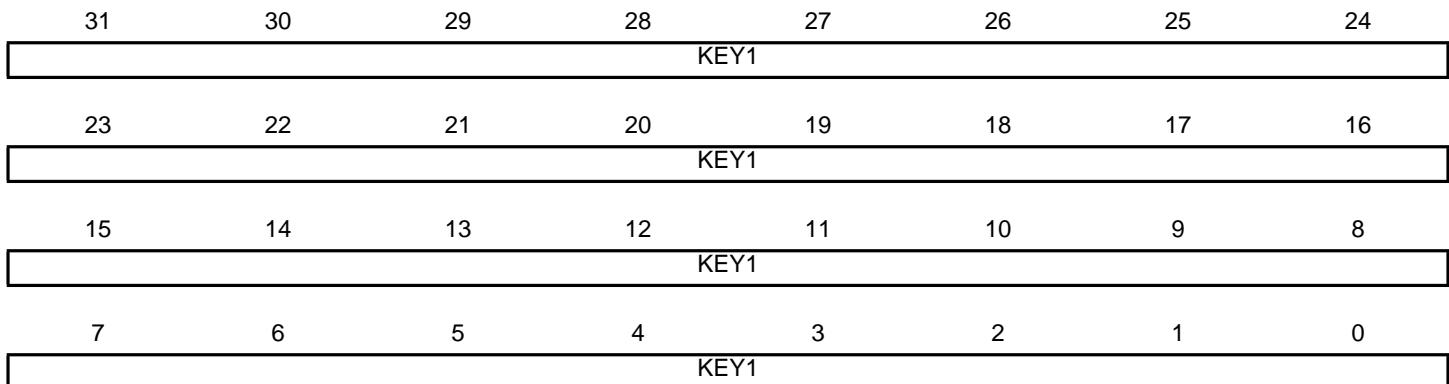
## 28.15.6 SMC OCMS Key1 Register

**Name:** SMC\_KEY1

**Address:** 0x40060084

**Access:** Write Once

**Reset:** 0x00000000



- **KEY1: Off Chip Memory Scrambling (OCMS) Key Part 1**

When Off Chip Memory Scrambling is enabled setting the SMC\_OCMS and SMC\_TIMINGS registers in accordance, the data scrambling depends on KEY1 and KEY2 values.

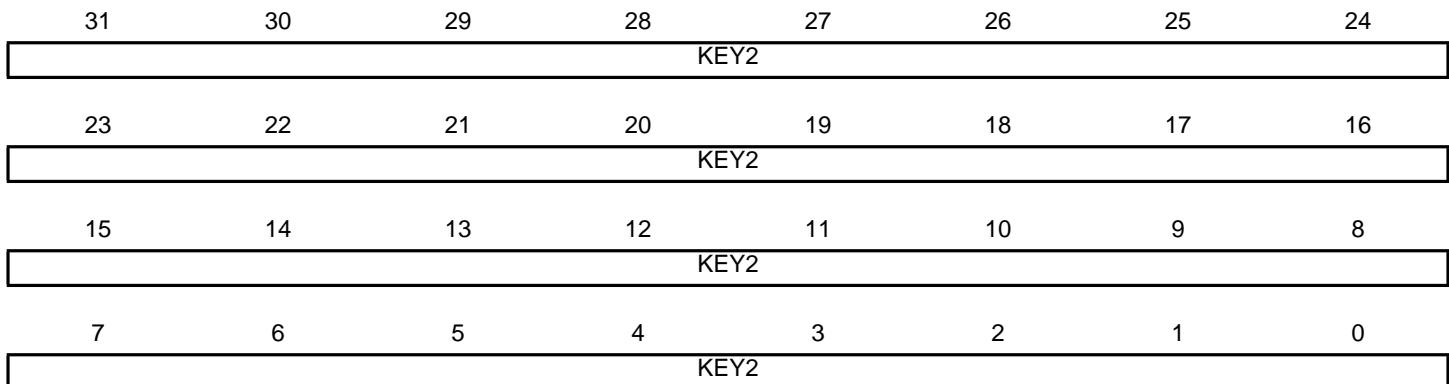
## 28.15.7 SMC OCMS Key2 Register

Name: SMC\_KEY2

Address: 0x40060088

Access: Write Once

Reset: 0x00000000



- **KEY2: Off Chip Memory Scrambling (OCMS) Key Part 2**

When Off Chip Memory Scrambling is enabled setting the SMC\_OCMS and SMC\_TIMINGS registers in accordance, the data scrambling depends on KEY2 and KEY1 values.

### 28.15.8 SMC Write Protect Mode Register

**Name:** SMC\_WPMR

**Address:** 0x400600E4

**Access:** Read-write

**Reset:** See [Table 28-7](#)

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x534D43 (“SMC” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x534D43 (“SMC” in ASCII).

Protects the registers listed below:

- [Section 28.15.1 “SMC Setup Register”](#)
- [Section 28.15.2 “SMC Pulse Register”](#)
- [Section 28.15.3 “SMC Cycle Register”](#)
- [Section 28.15.4 “SMC MODE Register”](#)

- **WPKEY: Write Protect KEY**

Should be written at value 0x534D43 (“SMC” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 28.15.9 SMC Write Protect Status Register

**Name:** SMC\_WPSR

**Address:** 0x400600E8

**Type:** Read-only

**Value:** See [Table 28-7](#)

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
WPVSRC							
WPVSRC							
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	WPVS
7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	WPVS

- **WPVS: Write Protect Enable**

0 = No Write Protect Violation has occurred since the last read of the SMC\_WPSR register.

1 = A Write Protect Violation occurred since the last read of the SMC\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Note: Reading SMC\_WPSR automatically clears all fields.



## 29. Peripheral DMA Controller (PDC)

### 29.1 Description

The Peripheral DMA Controller (PDC) transfers data between on-chip serial peripherals and the on- and/or off-chip memories. The link between the PDC and a serial peripheral is operated by the AHB to APB bridge.

The user interface of each PDC channel is integrated into the user interface of the peripheral it serves. The user interface of mono directional channels (receive only or transmit only), contains two 32-bit memory pointers and two 16-bit counters, one set (pointer, counter) for current transfer and one set (pointer, counter) for next transfer. The bi-directional channel user interface contains four 32-bit memory pointers and four 16-bit counters. Each set (pointer, counter) is used by current transmit, next transmit, current receive and next receive.

Using the PDC removes processor overhead by reducing its intervention during the transfer. This significantly reduces the number of clock cycles required for a data transfer, which improves microcontroller performance.

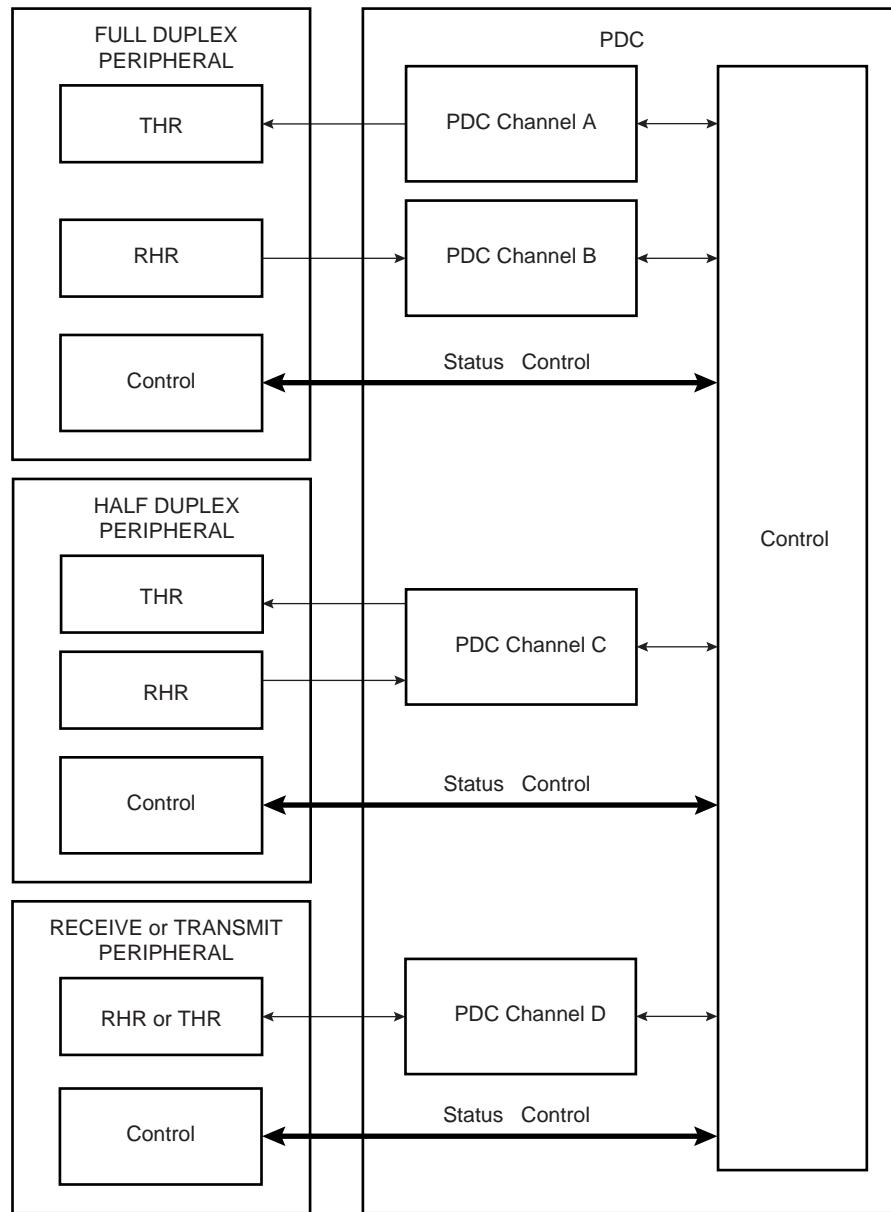
To launch a transfer, the peripheral triggers its associated PDC channels by using transmit and receive signals. When the programmed data is transferred, an end of transfer interrupt is generated by the peripheral itself.

### 29.2 Embedded Characteristics

- Performs Transfers to/from APB Communication Serial Peripherals
- Supports Half-duplex and Full-duplex Peripherals

## 29.3 Block Diagram

Figure 29-1. Block Diagram



## 29.4 Functional Description

### 29.4.1 Configuration

The PDC channel user interface enables the user to configure and control data transfers for each channel. The user interface of each PDC channel is integrated into the associated peripheral user interface.

The user interface of a serial peripheral, whether it is full or half duplex, contains four 32-bit pointers (RPR, RNPR, TPR, TNPR) and four 16-bit counter registers (RCR, RNCR, TCR, TNCR). However, the transmit and receive parts of each type are programmed differently: the transmit and receive parts of a full duplex peripheral can be programmed at the same time, whereas only one part (transmit or receive) of a half duplex peripheral can be programmed at a time.

32-bit pointers define the access location in memory for current and next transfer, whether it is for read (transmit) or write (receive). 16-bit counters define the size of current and next transfers. It is possible, at any moment, to read the number of transfers left for each channel.

The PDC has dedicated status registers which indicate if the transfer is enabled or disabled for each channel. The status for each channel is located in the associated peripheral status register. Transfers can be enabled and/or disabled by setting TXTEN/TXTDIS and RXTEN/RXTDIS in the peripheral's Transfer Control Register.

At the end of a transfer, the PDC channel sends status flags to its associated peripheral. These flags are visible in the peripheral status register (ENDRX, ENDTX, RXBUFF, and TXBUFE). Refer to [Section 29.4.3](#) and to the associated peripheral user interface.

### 29.4.2 Memory Pointers

Each full duplex peripheral is connected to the PDC by a receive channel and a transmit channel. Both channels have 32-bit memory pointers that point respectively to a receive area and to a transmit area in on- and/or off-chip memory.

Each half duplex peripheral is connected to the PDC by a bidirectional channel. This channel has two 32-bit memory pointers, one for current transfer and the other for next transfer. These pointers point to transmit or receive data depending on the operating mode of the peripheral.

Depending on the type of transfer (byte, half-word or word), the memory pointer is incremented respectively by 1, 2 or 4 bytes.

If a memory pointer address changes in the middle of a transfer, the PDC channel continues operating using the new address.

### 29.4.3 Transfer Counters

Each channel has two 16-bit counters, one for current transfer and the other one for next transfer. These counters define the size of data to be transferred by the channel. The current transfer counter is decremented first as the data addressed by current memory pointer starts to be transferred. When the current transfer counter reaches zero, the channel checks its next transfer counter. If the value of next counter is zero, the channel stops transferring data and sets the appropriate flag. But if the next counter value is greater than zero, the values of the next pointer/next counter are copied into the current pointer/current counter and the channel resumes the transfer whereas next pointer/next counter get zero/zero as values. At the end of this transfer the PDC channel sets the appropriate flags in the Peripheral Status Register.

The following list gives an overview of how status register flags behave depending on the counters' values:

- ENDRX flag is set when the PERIPH\_RCR register reaches zero.
- RXBUFF flag is set when both PERIPH\_RCR and PERIPH\_RNCR reach zero.
- ENDTX flag is set when the PERIPH\_TCR register reaches zero.
- TXBUFE flag is set when both PERIPH\_TCR and PERIPH\_TNCR reach zero.

These status flags are described in the Peripheral Status Register.

### 29.4.4 Data Transfers

The serial peripheral triggers its associated PDC channels' transfers using transmit enable (TXEN) and receive enable (RXEN) flags in the transfer control register integrated in the peripheral's user interface.

When the peripheral receives an external data, it sends a Receive Ready signal to its PDC receive channel which then requests access to the Matrix. When access is granted, the PDC receive channel starts reading the peripheral Receive Holding Register (RHR). The read data are stored in an internal buffer and then written to memory.

When the peripheral is about to send data, it sends a Transmit Ready to its PDC transmit channel which then requests access to the Matrix. When access is granted, the PDC transmit channel reads data from memory and puts them to Transmit Holding Register (THR) of its associated peripheral. The same peripheral sends data according to its mechanism.

#### 29.4.5 PDC Flags and Peripheral Status Register

Each peripheral connected to the PDC sends out receive ready and transmit ready flags and the PDC sends back flags to the peripheral. All these flags are only visible in the Peripheral Status Register.

Depending on the type of peripheral, half or full duplex, the flags belong to either one single channel or two different channels.

##### 29.4.5.1 Receive Transfer End

This flag is set when PERIPH\_RCR register reaches zero and the last data has been transferred to memory.

It is reset by writing a non zero value in PERIPH\_RCR or PERIPH\_RNCR.

##### 29.4.5.2 Transmit Transfer End

This flag is set when PERIPH\_TCR register reaches zero and the last data has been written into peripheral THR.

It is reset by writing a non zero value in PERIPH\_TCR or PERIPH\_TNCR.

##### 29.4.5.3 Receive Buffer Full

This flag is set when PERIPH\_RCR register reaches zero with PERIPH\_RNCR also set to zero and the last data has been transferred to memory.

It is reset by writing a non zero value in PERIPH\_TCR or PERIPH\_TNCR.

##### 29.4.5.4 Transmit Buffer Empty

This flag is set when PERIPH\_TCR register reaches zero with PERIPH\_TNCR also set to zero and the last data has been written into peripheral THR.

It is reset by writing a non zero value in PERIPH\_TCR or PERIPH\_TNCR.

## 29.5 Peripheral DMA Controller (PDC) User Interface

Table 29-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Receive Pointer Register	PERIPH <sup>(1)</sup> _RPR	Read-write	0
0x04	Receive Counter Register	PERIPH_RCR	Read-write	0
0x08	Transmit Pointer Register	PERIPH_TPR	Read-write	0
0x0C	Transmit Counter Register	PERIPH_TCR	Read-write	0
0x10	Receive Next Pointer Register	PERIPH_RNPR	Read-write	0
0x14	Receive Next Counter Register	PERIPH_RNCR	Read-write	0
0x18	Transmit Next Pointer Register	PERIPH_TNPR	Read-write	0
0x1C	Transmit Next Counter Register	PERIPH_TNCR	Read-write	0
0x20	Transfer Control Register	PERIPH_PTCR	Write-only	0
0x24	Transfer Status Register	PERIPH PTSR	Read-only	0

Note: 1. PERIPH: Ten registers are mapped in the peripheral memory space at the same offset. These can be defined by the user according to the function and the desired peripheral.)

### 29.5.1 Receive Pointer Register

**Name:** PERIPH\_RPR

**Access:** Read-write

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

- **RXPTR: Receive Pointer Register**

RXPTR must be set to receive buffer address.

When a half duplex peripheral is connected to the PDC, RXPTR = TXPTR.

### 29.5.2 Receive Counter Register

**Name:** PERIPH\_RCR

**Access:** Read-write

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

- **RXCTR: Receive Counter Register**

RXCTR must be set to receive buffer size.

When a half duplex peripheral is connected to the PDC, RXCTR = TXCTR.

0 = Stops peripheral data transfer to the receiver

1 - 65535 = Starts peripheral data transfer if corresponding channel is active

### 29.5.3 Transmit Pointer Register

**Name:** PERIPH\_TPR

**Access:** Read-write

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

- **TXPTR: Transmit Counter Register**

TXPTR must be set to transmit buffer address.

When a half duplex peripheral is connected to the PDC, RXPTR = TXPTR.

#### 29.5.4 Transmit Counter Register

**Name:** PERIPH\_TCR

**Access:** Read-write

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

- **TXCTR: Transmit Counter Register**

TXCTR must be set to transmit buffer size.

When a half duplex peripheral is connected to the PDC, RXCTR = TXCTR.

0 = Stops peripheral data transfer to the transmitter

1- 65535 = Starts peripheral data transfer if corresponding channel is active

### 29.5.5 Receive Next Pointer Register

**Name:** PERIPH\_RNPR

**Access:** Read-write

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

- **RXNPTR: Receive Next Pointer**

RXNPTR contains next receive buffer address.

When a half duplex peripheral is connected to the PDC, RXNPTR = TXNPTR.

### 29.5.6 Receive Next Counter Register

**Name:** PERIPH\_RNCR

**Access:** Read-write

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

- **RXNCTR: Receive Next Counter**

RXNCTR contains next receive buffer size.

When a half duplex peripheral is connected to the PDC, RXNCTR = TXNCTR.

### 29.5.7 Transmit Next Pointer Register

**Name:** PERIPH\_TNPR

**Access:** Read-write

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

- **TXNPTR: Transmit Next Pointer**

TXNPTR contains next transmit buffer address.

When a half duplex peripheral is connected to the PDC, RXNPTR = TXNPTR.

### 29.5.8 Transmit Next Counter Register

**Name:** PERIPH\_TNCR

**Access:** Read-write

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

- **TXNCTR: Transmit Counter Next**

TXNCTR contains next transmit buffer size.

When a half duplex peripheral is connected to the PDC, RXNCTR = TXNCTR.

### 29.5.9 Transfer Control Register

**Name:** PERIPH\_PTCR

**Access:** Write-only

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

- **RXTEN: Receiver Transfer Enable**

0 = No effect.

1 = Enables PDC receiver channel requests if RXTDIS is not set.

When a half duplex peripheral is connected to the PDC, enabling the receiver channel requests automatically disables the transmitter channel requests. It is forbidden to set both TXTEN and RXTEN for a half duplex peripheral.

- **RXTDIS: Receiver Transfer Disable**

0 = No effect.

1 = Disables the PDC receiver channel requests.

When a half duplex peripheral is connected to the PDC, disabling the receiver channel requests also disables the transmitter channel requests.

- **TXTEN: Transmitter Transfer Enable**

0 = No effect.

1 = Enables the PDC transmitter channel requests.

When a half duplex peripheral is connected to the PDC, it enables the transmitter channel requests only if RXTEN is not set. It is forbidden to set both TXTEN and RXTEN for a half duplex peripheral.

- **TXTDIS: Transmitter Transfer Disable**

0 = No effect.

1 = Disables the PDC transmitter channel requests.

When a half duplex peripheral is connected to the PDC, disabling the transmitter channel requests disables the receiver channel requests.

### 29.5.10 Transfer Status Register

**Name:** PERIPH\_PTSR

**Access:** Read-only

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

- **RXTEN: Receiver Transfer Enable**

0 = PDC receiver channel requests are disabled.

1 = PDC receiver channel requests are enabled.

- **TXTEN: Transmitter Transfer Enable**

0 = PDC transmitter channel requests are disabled.

1 = PDC transmitter channel requests are enabled.



## 30. Power Management Controller (PMC)

### 30.1 Clock Generator

#### 30.1.1 Description

The Clock Generator User Interface is embedded within the Power Management Controller and is described in [Section 30.2.16 "Power Management Controller \(PMC\) User Interface"](#). However, the Clock Generator registers are named CKGR\_.

#### 30.1.2 Embedded Characteristics

The Clock Generator is made up of:

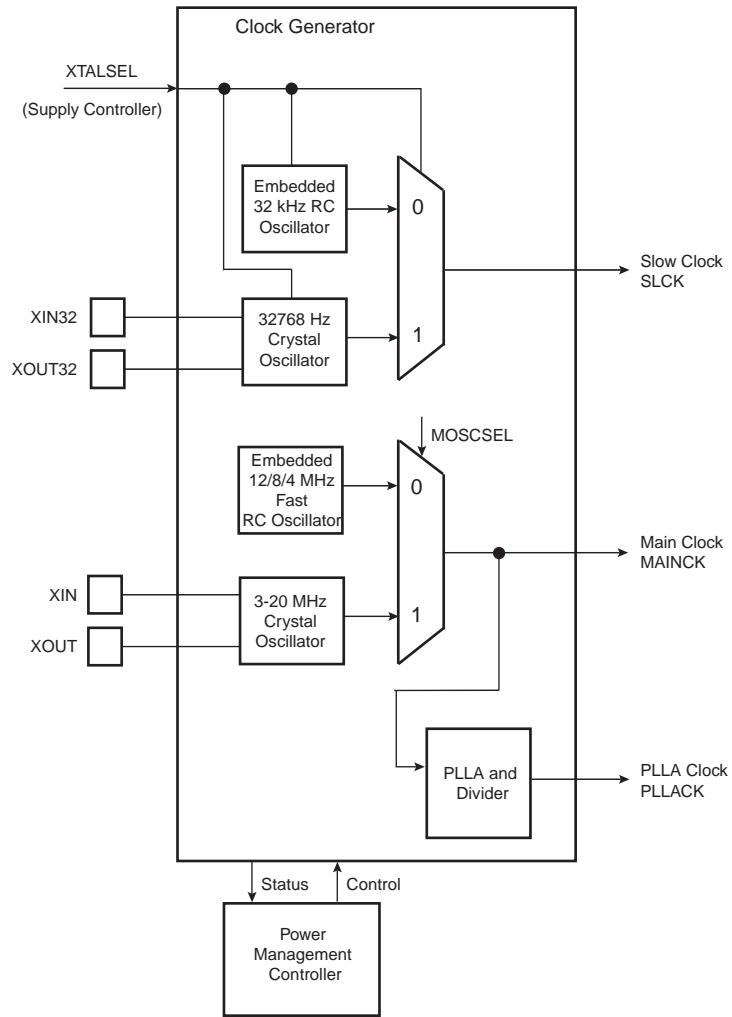
- A Low-Power 32768 Hz Slow Clock Oscillator with bypass mode.
- A Low-Power RC Oscillator
- A 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator, which can be bypassed.
- A factory programmed Fast RC Oscillator. Three output frequencies can be selected: 4/8/12 MHz. By default 4 MHz is selected.
- A 80 to 240 MHz programmable PLL (input from 3 to 32 MHz), capable of providing the clock MCK to the processor and to the peripherals.
- Write Protected Registers

It provides the following clocks:

- SLCK, the Slow Clock, which is the only permanent clock within the system.
- MAINCK is the output of the Main Clock Oscillator selection: either the Crystal or Ceramic Resonator-based Oscillator or 4/8/12 MHz Fast RC Oscillator.
- PLLACK is the output of the Divider and 80 to 240 MHz programmable PLL (PLLA).

### 30.1.3 Block Diagram

Figure 30-1. Clock Generator Block Diagram



### 30.1.4 Slow Clock

The Supply Controller embeds a slow clock generator that is supplied with the VDDIO power supply. As soon as the VDDIO is supplied, both the crystal oscillator and the embedded RC oscillator are powered up, but only the embedded RC oscillator is enabled. This allows the slow clock to be valid in a short time (about 100  $\mu$ s).

The Slow Clock is generated either by the Slow Clock Crystal Oscillator or by the Slow Clock RC Oscillator.

The selection between the RC or the crystal oscillator is made by writing the XTALSEL bit in the Supply Controller Control Register (SUPC\_CR).

#### 30.1.4.1 Slow Clock RC Oscillator

By default, the Slow Clock RC Oscillator is enabled and selected. The user has to take into account the possible drifts of the RC Oscillator. More details are given in the section “DC Characteristics” of the product datasheet.

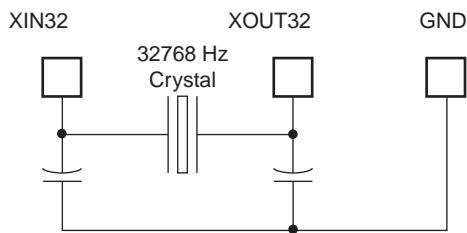
It can be disabled via the XTALSEL bit in the Supply Controller Control Register (SUPC\_CR).

#### 30.1.4.2 Slow Clock Crystal Oscillator

The Clock Generator integrates a 32768 Hz low-power oscillator. In order to use this oscillator, the XIN32 and XOUT32 pins must be connected to a 32768 Hz crystal. Two external capacitors must be wired as shown in [Figure 30-2](#). More details are given in the section “DC Characteristics” of the product datasheet.

Note that the user is not obliged to use the Slow Clock Crystal and can use the RC oscillator instead.

**Figure 30-2.** Typical Slow Clock Crystal Oscillator Connection



The user can select the crystal oscillator to be the source of the slow clock, as it provides a more accurate frequency. The command is made by writing the Supply Controller Control Register (SUPC\_CR) with the XTALSEL bit at 1. This results in a sequence which first configures the PIO lines multiplexed with XIN32 and XOUT32 to be driven by the oscillator, then enables the crystal oscillator and then disables the RC oscillator to save power. The switch of the slow clock source is glitch free. The OSCSEL bit of the Supply Controller Status Register (SUPC\_SR) or the OSCSEL bit of the PMC Status Register (PMC\_SR) tracks the oscillator frequency downstream. It must be read in order to be informed when the switch sequence, initiated when a new value is written in the XTALSEL bit of SUPC\_CR, is done.

Coming back on the RC oscillator is only possible by shutting down the VDDIO power supply. If the user does not need the crystal oscillator, the XIN32 and XOUT32 pins can be left unconnected since by default the XIN32 and XOUT32 system I/O pins are in PIO input mode with pull-up after reset.

The user can also set the crystal oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin are given in the product electrical characteristics section. In order to set the bypass mode, the OSCBYPASS bit of the Supply Controller Mode Register (SUPC\_MR) needs to be set at 1.

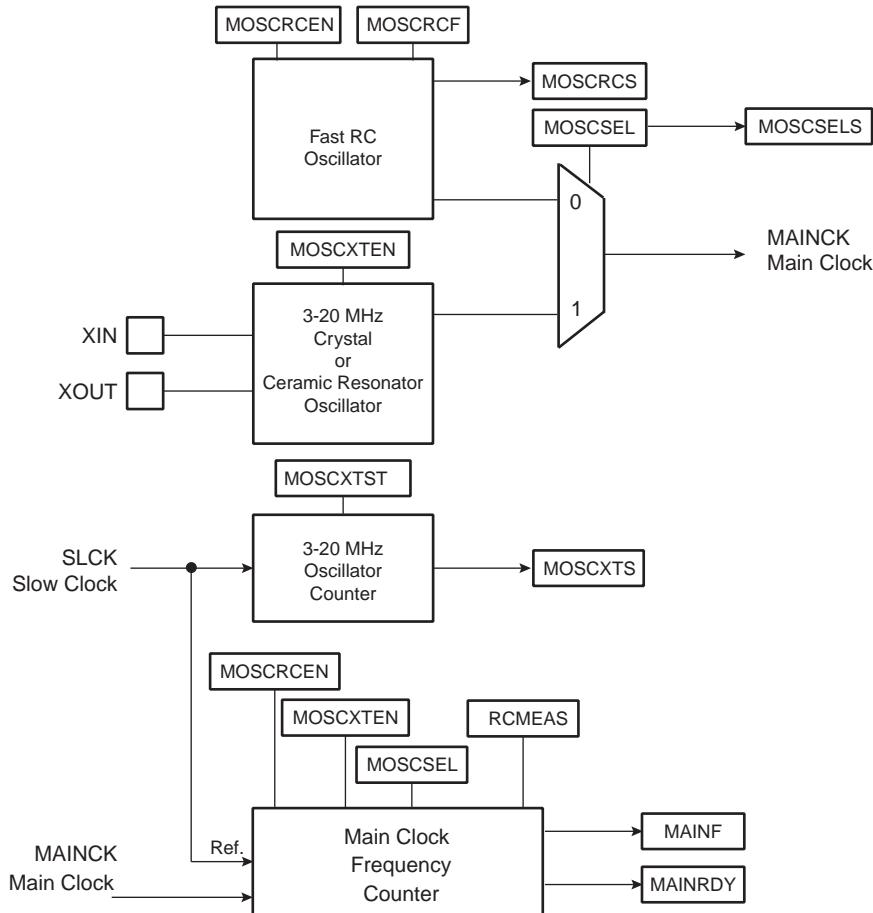
The user can set the Slow Clock Crystal Oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin under these conditions are given in the product electrical characteristics section.

The programmer has to be sure to set the OSCBYPASS bit in the Supply Controller Mode Register (SUPC\_MR) and XTALSEL bit in the Supply Controller Control Register (SUPC\_CR).

### 30.1.5 Main Clock

Figure 30-3 shows the Main Clock block diagram.

**Figure 30-3. Main Clock Block Diagram**



The Main Clock has two sources:

- 4/8/12 MHz Fast RC Oscillator which starts very quickly and is used at start-up.
- 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator which can be bypassed.

#### 30.1.5.1 Fast RC Oscillator

After reset, the 4/8/12 MHz Fast RC Oscillator is enabled with the 4 MHz frequency selected and it is selected as the source of MAINCK. MAINCK is the default clock selected to start up the system.

The Fast RC Oscillator frequencies are calibrated in production except the lowest frequency which is not calibrated.

Please refer to the “DC Characteristics” section of the product datasheet.

The software can disable or enable the 4/8/12 MHz Fast RC Oscillator with the MOSCRCEN bit in the Clock Generator Main Oscillator Register (CKGR\_MOR).

The user can also select the output frequency of the Fast RC Oscillator, either 4/8/12 MHz are available. It can be done through MOSCRCF bits in CKGR\_MOR. When changing this fre-

quency selection, the MOSCRCS bit in the Power Management Controller Status Register (PMC\_SR) is automatically cleared and MAINCK is stopped until the oscillator is stabilized. Once the oscillator is stabilized, MAINCK restarts and MOSCRCS is set.

When disabling the Main Clock by clearing the MOSCRCEN bit in CKGR\_MOR, the MOSCRCS bit in the Power Management Controller Status Register (PMC\_SR) is automatically cleared, indicating the Main Clock is off.

Setting the MOSCRCS bit in the Power Management Controller Interrupt Enable Register (PMC\_IER) can trigger an interrupt to the processor.

It is recommended to disable the Main Clock as soon as the processor no longer uses it and runs out of SLCK.

The CAL4, CAL8 and CAL12 values in the PMC Oscillator Calibration Register (PMC\_OCR) are the default values set by Atmel during production. These values are stored in a specific Flash memory area different from the main memory plane. These values cannot be modified by the user and cannot be erased by a Flash erase command or by the ERASE pin. Values written by the user's application in PMC\_OCR are reset after each power up or peripheral reset.

#### 30.1.5.2 *Fast RC Oscillator Clock Frequency Adjustment*

It is possible for the user to adjust the main RC oscillator frequency through PMC\_OCR. By default, SEL4/8/12 are low, so the RC oscillator will be driven with Flash calibration bits which are programmed during chip production.

The user can adjust the trimming of the 4/8/12 MHz Fast RC Oscillator through this register in order to obtain more accurate frequency (to compensate derating factors such as temperature and voltage).

In order to calibrate the oscillator lower frequency, SEL must be set to 1 and a good frequency value must be configured in CAL4. Likewise, SEL8/12 must be set to 1 and a trim value must be configured in CAL8/12 in order to adjust the other frequencies of the oscillator.

However, the adjustment can not be done to the frequency from which the oscillator is operating. For example, while running from the lower possible frequency, the user can adjust the other frequencies but not the lowest one.

It is possible to restart, at anytime, a measurement of the main frequency by means of the RCMEAS bit in Main Clock Frequency Register (CKGR\_MCFR). Thus, when MAINFRDY flag reads 1, another read access on Main Clock Frequency Register (CKGR\_MCFR) provides an image of the frequency of the main clock on MAINF field. The software can calculate the error with an expected frequency and correct the CAL (or CAL8/CAL12) field accordingly. This may be used to compensate frequency drift due to derating factors such as temperature and/or voltage.

#### 30.1.5.3 *3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator*

After reset, the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator is disabled and it is not selected as the source of MAINCK.

The user can select the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator to be the source of MAINCK, as it provides a more accurate frequency. The software enables or disables the main oscillator so as to reduce power consumption by clearing the MOSCXTEN bit in the Main Oscillator Register (CKGR\_MOR).

When disabling the main oscillator by clearing the MOSCXTEN bit in CKGR\_MOR, the MOSCXTS bit in PMC\_SR is automatically cleared, indicating the Main Clock is off.

When enabling the main oscillator, the user must initiate the main oscillator counter with a value corresponding to the start-up time of the oscillator. This start-up time depends on the crystal frequency connected to the oscillator.

When the MOSCXTEN bit and the MOSCXTST are written in CKGR\_MOR to enable the main oscillator, the XIN and XOUT pins are automatically switched into oscillator mode and MOSCXTS bit in the Power Management Controller Status Register (PMC\_SR) is cleared and the counter starts counting down on the slow clock divided by 8 from the MOSCXTST value. Since the MOSCXTST value is coded with 8 bits, the maximum start-up time is about 62 ms.

When the counter reaches 0, the MOSCXTS bit is set, indicating that the main clock is valid. Setting the MOSCXTS bit in PMC\_IMR can trigger an interrupt to the processor.

#### 30.1.5.4 Main Clock Oscillator Selection

The user can select either the 4/8/12 MHz Fast RC Oscillator or the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator to be the source of Main Clock.

The advantage of the 4/8/12 MHz Fast RC Oscillator is that it provides fast start-up time, this is why it is selected by default (to start up the system) and when entering Wait Mode.

The advantage of the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator is that it is very accurate.

The selection is made by writing the MOSCSEL bit in the Main Oscillator Register (CKGR\_MOR). The switch of the Main Clock source is glitch free, so there is no need to run out of SLCK, PLLACK in order to change the selection. The MOSCSELS bit of the Power Management Controller Status Register (PMC\_SR) allows knowing when the switch sequence is done.

Setting the MOSCSELS bit in PMC\_IMR can trigger an interrupt to the processor.

Enabling the Fast RC Oscillator (MOSCRCEN = 1) and changing the Fast RC Frequency (MOSCCRF) at the same time is not allowed.

The Fast RC must be enabled first and its frequency changed in a second step.

#### 30.1.5.5 Software Sequence to Detect the Presence of Fast Crystal

The frequency meter carried on the CKGR\_MCFR register is operating on the selected main clock and not on the fast crystal clock nor on the fast RC Oscillator clock.

Therefore, to check for the presence of the fast crystal clock, it is necessary to have the main clock (MAINCK) driven by the fast crystal clock (MOSCSEL=1).

The following software sequence order must be followed:

- MCK must select the slow clock (CSS=0 in the PMC\_MCKR register).
- Wait for the MCKRDY flag in the PMC\_SR register to be 1.
- The fast crystal must be enabled by programming 1 in the MOSCXTEN field in the CKGR\_MOR register with the MOSCXTST field being programmed to the appropriate value (see the Electrical Characteristics chapter).
- Wait for the MOSCXTS flag to be 1 in the PMC\_SR register to get the end of a start-up period of the fast crystal oscillator.

- Then, MOSCSEL must be programmed to 1 in the CKGR\_MOR register to select fast main crystal oscillator for the main clock.
- MOSCSEL must be read until its value equals 1.
- Then the MOSCSELS status flag must be checked in the PMC\_SR register.

At this point, 2 cases may occur (either MOSCSELS = 0 or MOSCSELS = 1).

- If MOSCSELS = 1, there is a valid crystal connected and its frequency can be determined by initiating a frequency measure by programming RCMEAS in the CKGR\_MCFR register.
- If MOSCSELS = 0, there is no fast crystal clock (either no crystal connected or a crystal clock out of specification).  
A frequency measure can reinforce this status by initiating a frequency measure by programming RCMEAS in the CKGR\_MCFR register.
- If MOSCSELS=0, the selection of the main clock must be programmed back to the main RC oscillator by writing MOSCSEL to 0 prior to disabling the fast crystal oscillator.
- If MOSCSELS=0, the crystal oscillator can be disabled (MOSCXTEN=0 in the CKGR\_MOR register).

### 30.1.5.6 Main Clock Frequency Counter

The device features a Main Clock frequency counter that provides the frequency of the Main Clock.

The Main Clock frequency counter is reset and starts incrementing at the Main Clock speed after the next rising edge of the Slow Clock in the following cases:

- When the 4/8/12 MHz Fast RC Oscillator clock is selected as the source of Main Clock and when this oscillator becomes stable (i.e., when the MOSCRCS bit is set)
- When the 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator is selected as the source of Main Clock and when this oscillator becomes stable (i.e., when the MOSCXTS bit is set)
- When the Main Clock Oscillator selection is modified
- When the RCMEAS bit of CKGR\_MCFR is written to 1.

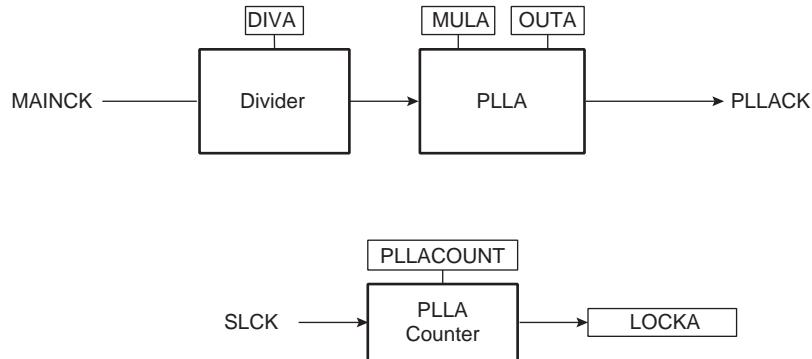
Then, at the 16th falling edge of Slow Clock, the MAINFRDY bit in the Clock Generator Main Clock Frequency Register (CKGR\_MCFR) is set and the counter stops counting. Its value can be read in the MAINF field of CKGR\_MCFR and gives the number of Main Clock cycles during 16 periods of Slow Clock, so that the frequency of the 4/8/12 MHz Fast RC Oscillator or 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator can be determined.

### 30.1.6 Divider and PLL Block

The device features one Divider/one PLL Block that permits a wide range of frequencies to be selected on either the master clock, the processor clock or the programmable clock outputs. Additionally, they provide a 48 MHz signal to the embedded USB device port regardless of the frequency of the main clock.

Figure 30-4 shows the block diagram of the dividers and PLL blocks.

**Figure 30-4.** Divider and PLL Block Diagram



### 30.1.6.1 Divider and Phase Lock Loop Programming

The divider can be set between 1 and 255 in steps of 1. When a divider field (DIV) is set to 0, the output of the corresponding divider and the PLL output is a continuous signal at level 0. On reset, each DIV field is set to 0, thus the corresponding PLL input clock is set to 0.

The PLL (PLLA) allows multiplication of the divider's outputs. The PLL clock signal has a frequency that depends on the respective source signal frequency and on the parameters DIV (DIVA) and MUL (MULA). The factor applied to the source signal frequency is  $(MUL + 1)/DIV$ . When MUL is written to 0 or DIV=0, the PLL is disabled and its power consumption is saved. Re-enabling the PLL can be performed by writing a value higher than 0 in the MUL field and DIV higher than 0.

Whenever the PLL is re-enabled or one of its parameters is changed, the LOCK (LOCKA) bit in PMC\_SR is automatically cleared. The values written in the PLLCOUNT field (PLLCOUNT) in CKGR\_PLLR (CKGR\_PLLAR) are loaded in the PLL counter. The PLL counter then decrements at the speed of the Slow Clock until it reaches 0. At this time, the LOCK bit is set in PMC\_SR and can trigger an interrupt to the processor. The user has to load the number of Slow Clock cycles required to cover the PLL transient time into the PLLCOUNT field.

The PLL clock can be divided by 2 by writing the PLLDIV2 (PLLADIV2) bit in PMC Master Clock Register (PMC\_MCKR).

It is forbidden to change 4/8/12 MHz Fast RC Oscillator, or main selection in CKGR\_MOR register while Master Clock source is PLL and PLL reference clock is the Fast RC Oscillator.

The user must:

- Switch on the Main RC oscillator by writing 1 in CSS field of PMC\_MCKR.
- Change the frequency (MOSCRCF) or oscillator selection (MOSCSEL) in CKGR\_MOR.
- Wait for MOSCRCS (if frequency changes) or MOSCSELS (if oscillator selection changes) in PMC\_SR.
- Disable and then enable the PLL (LOCK in PMC\_IDR and PMC\_IER).
- Wait for LOCK flag in PMC\_SR.
- Switch back to PLL by writing the appropriate value to CSS field of PMC\_MCKR.

## 30.2 Power Management Controller (PMC)

### 30.2.1 Description

The Power Management Controller (PMC) optimizes power consumption by controlling all system and user peripheral clocks. The PMC enables/disables the clock inputs to many of the peripherals and the Cortex-M4 Processor.

The Supply Controller selects between the 32 kHz RC oscillator or the slow crystal oscillator. The unused oscillator is disabled automatically so that power consumption is optimized.

By default, at start-up the chip runs out of the Master Clock using the Fast RC Oscillator running at 4 MHz.

The user can trim the 8 and 12 MHz RC Oscillator frequencies by software.

### 30.2.2 Embedded Characteristics

The Power Management Controller provides the following clocks:

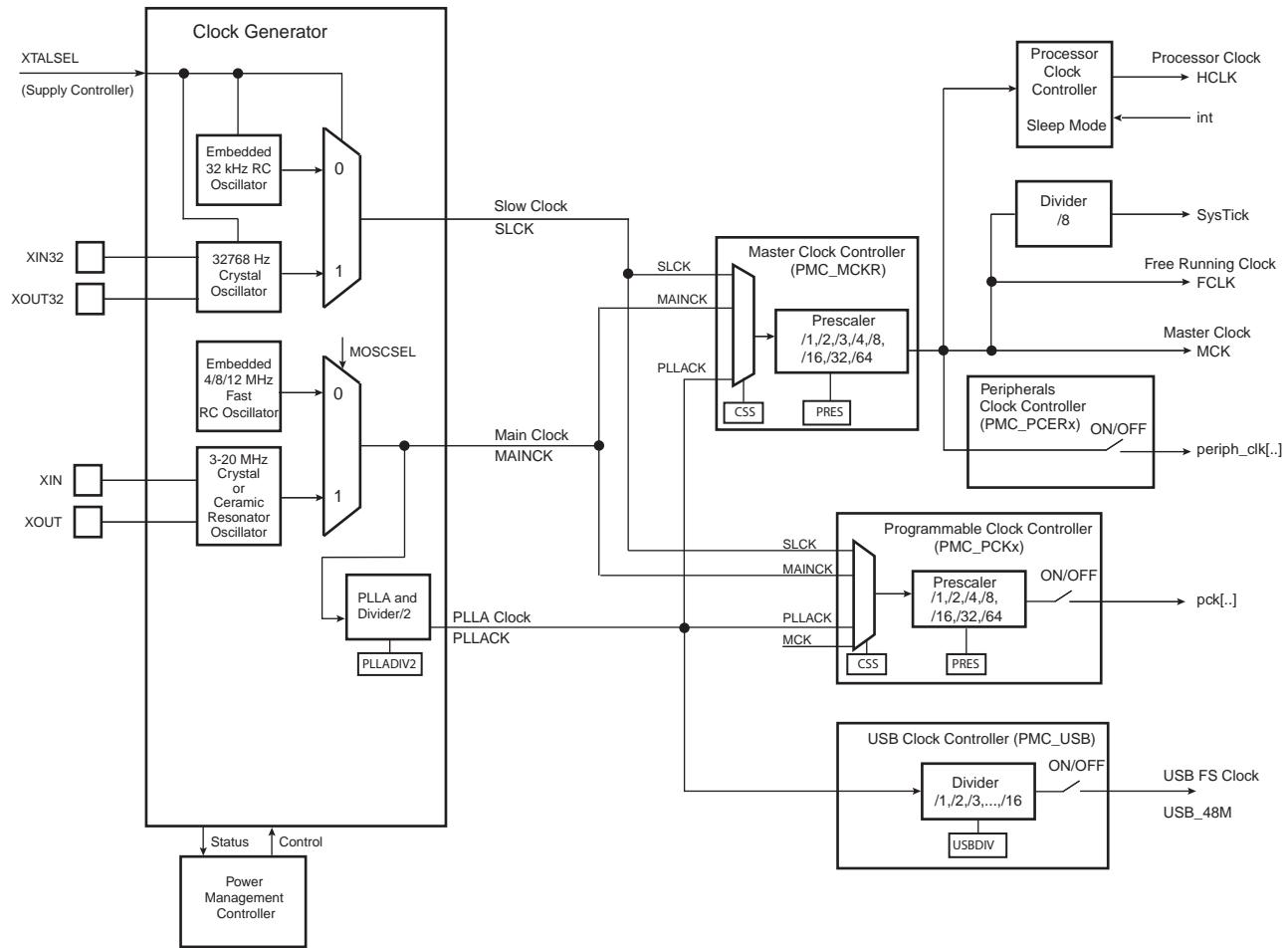
- MCK, the Master Clock, programmable from a few hundred Hz to the maximum operating frequency of the device. It is available to the modules running permanently, such as the Enhanced Embedded Flash Controller.
- Processor Clock (HCLK), automatically switched off when entering the processor in Sleep Mode.
- Free running processor Clock (FCLK)
- The Cortex-M4 SysTick external clock
- UDP Clock (UDPCK), required by USB Device Port operations.
- Peripheral Clocks, typically MCK, provided to the embedded peripherals (USART, SPI, TWI, TC, etc.) and independently controllable. In order to reduce the number of clock names in a product, the Peripheral Clocks are named MCK in the product datasheet.
- Programmable Clock Outputs can be selected from the clocks provided by the clock generator and driven on the PCKx pins.
- Write Protected Registers

The Power Management Controller also provides the following operations on clocks:

- A main crystal oscillator clock failure detector.
- A frequency counter on main clock and an adjustable main RC oscillator frequency.

### 30.2.3 Block Diagram

**Figure 30-5.** General Clock Block Diagram



### 30.2.4 Master Clock Controller

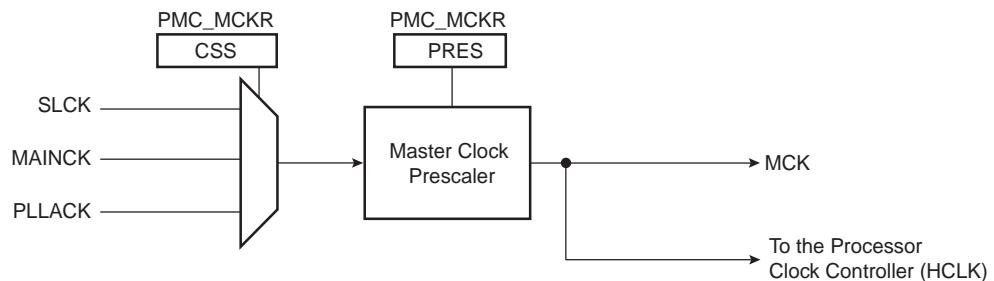
The Master Clock Controller provides selection and division of the Master Clock (MCK). MCK is the clock provided to all the peripherals. The Master Clock is selected from one of the clocks provided by the Clock Generator.

Selecting the Slow Clock provides a Slow Clock signal to the whole device. Selecting the Main Clock saves power consumption of the PLL. The Master Clock Controller is made up of a clock selector and a prescaler.

The Master Clock selection is made by writing the CSS field (Clock Source Selection) in PMC\_MCKR (Master Clock Register). The prescaler supports the division by a power of 2 of the selected clock between 1 and 64, and the division by 3. The PRES field in PMC\_MCKR programs the prescaler.

Each time PMC\_MCKR is written to define a new Master Clock, the MCKRDY bit is cleared in PMC\_SR. It reads 0 until the Master Clock is established. Then, the MCKRDY bit is set and can trigger an interrupt to the processor. This feature is useful when switching from a highspeed clock to a lower one to inform the software when the change is actually done.

**Figure 30-6.** Master Clock Controller



### 30.2.5 Processor Clock Controller

The PMC features a Processor Clock Controller (HCLK) that implements the Processor Sleep Mode. The Processor Clock can be disabled by executing the WFI (WaitForInterrupt) or the WFE (WaitForEvent) processor instruction while the LPM bit is at 0 in the PMC Fast Start-up Mode Register (PMC\_FSMR).

The Processor Clock HCLK is enabled after a reset and is automatically re-enabled by any enabled interrupt. The Processor Sleep Mode is achieved by disabling the Processor Clock, which is automatically re-enabled by any enabled fast or normal interrupt, or by the reset of the product.

When Processor Sleep Mode is entered, the current instruction is finished before the clock is stopped, but this does not prevent data transfers from other masters of the system bus.

### 30.2.6 SysTick Clock

The SysTick calibration value is fixed to 15000 which allows the generation of a time base of 1 ms with SysTick clock to the maximum frequency on MCK divided by 8.

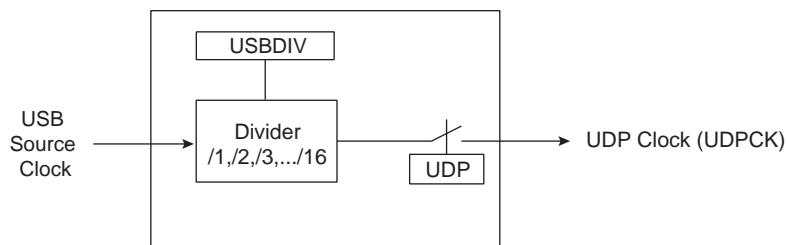
### 30.2.7 USB Clock Controller

The user can select the PLLA output as the USB Source Clock by writing the USBS bit in PMC\_USB. If using the USB, the user must program the PLL to generate an appropriate frequency depending on the USBDIV bit in PMC\_USB.

When the PLL output is stable, i.e., the LOCK bit is set:

- The USB device clock can be enabled by setting the UDP bit in PMC\_SCER. To save power on this peripheral when it is not used, the user can set the UDP bit in PMC\_SCDR. The UDP bit in PMC\_SCSR gives the activity of this clock. The USB device port requires both the 48 MHz signal and the Master Clock. The Master Clock may be controlled by means of the Master Clock Controller.

**Figure 30-7.** USB Clock Controller



### 30.2.8 Peripheral Clock Controller

The Power Management Controller controls the clocks of each embedded peripheral by means of the Peripheral Clock Controller. The user can individually enable and disable the Clock on the peripherals.

The user can also enable and disable these clocks by writing Peripheral Clock Enable 0 (PMC\_PCER0), Peripheral Clock Disable 0 (PMC\_PCDR0), Peripheral Clock Enable 1 (PMC\_PCER1) and Peripheral Clock Disable 1 (PMC\_PCDR1) registers. The status of the peripheral clock activity can be read in the Peripheral Clock Status Register (PMC\_PCSR0) and Peripheral Clock Status Register (PMC\_PCSR1).

When a peripheral clock is disabled, the clock is immediately stopped. The peripheral clocks are automatically disabled after a reset.

In order to stop a peripheral, it is recommended that the system software wait until the peripheral has executed its last programmed operation before disabling the clock. This is to avoid data corruption or erroneous behavior of the system.

The bit number within the Peripheral Clock Control registers (PMC\_PCER0-1, PMC\_PCDR0-1, and PMC\_PCSR0-1) is the Peripheral Identifier defined at the product level. The bit number corresponds to the interrupt source number assigned to the peripheral.

### 30.2.9 Free Running Processor Clock

The Free Running Processor Clock (FCLK) used for sampling interrupts and clocking debug blocks ensures that interrupts can be sampled, and sleep events can be traced, while the processor is sleeping. It is connected to Master Clock (MCK).

### 30.2.10 Programmable Clock Output Controller

The PMC controls 3 signals to be output on external pins, PCKx. Each signal can be independently programmed via the Programmable Clock Registers (PMC\_PCKx).

PCKx can be independently selected between the Slow Clock (SLCK), the Main Clock (MAINCK), the PLLA Clock (PLLACK), and the Master Clock (MCK) by writing the CSS field in PMC\_PCKx. Each output signal can also be divided by a power of 2 between 1 and 64 by writing the PRES (Prescaler) field in PMC\_PCKx.

Each output signal can be enabled and disabled by writing 1 in the corresponding bit, PCKx of PMC\_SCER and PMC\_SCDR, respectively. Status of the active programmable output clocks are given in the PCKx bits of PMC\_SCSR (System Clock Status Register).

Moreover, like the PCK, a status bit in PMC\_SR indicates that the Programmable Clock is actually what has been programmed in the Programmable Clock registers.

As the Programmable Clock Controller does not manage with glitch prevention when switching clocks, it is strongly recommended to disable the Programmable Clock before any configuration change and to re-enable it after the change is actually performed.

### 30.2.11 Fast Start-up

The device allows the processor to restart in less than 10 microseconds while the device is in Wait Mode.

The system enters Wait Mode either by writing the WAITMODE bit at 1 in the PMC Clock Generator Main Oscillator Register (CKGR\_MOR), or by executing the WaitForEvent (WFE) instruction of the processor while the LPM bit is at 1 in the PMC Fast Start-up Mode Register

(PMC\_FSMR). Waiting for the MOSCRCEN bit to be cleared is strongly recommended to ensure that the core will not execute undesired instructions.

**Important:** Prior to instructing the system to enter the Wait Mode, the internal sources of wake-up must be cleared. It must be verified that none of the enabled external wake-up inputs (WKUP) hold an active polarity.

A Fast Start-up is enabled upon the detection of a programmed level on one of the 16 wake-up inputs (WKUP) or upon an active alarm from the RTC, RTT and USB Controller. The polarity of the 16 wake-up inputs is programmable by writing the PMC Fast Start-up Polarity Register (PMC\_FSPR).

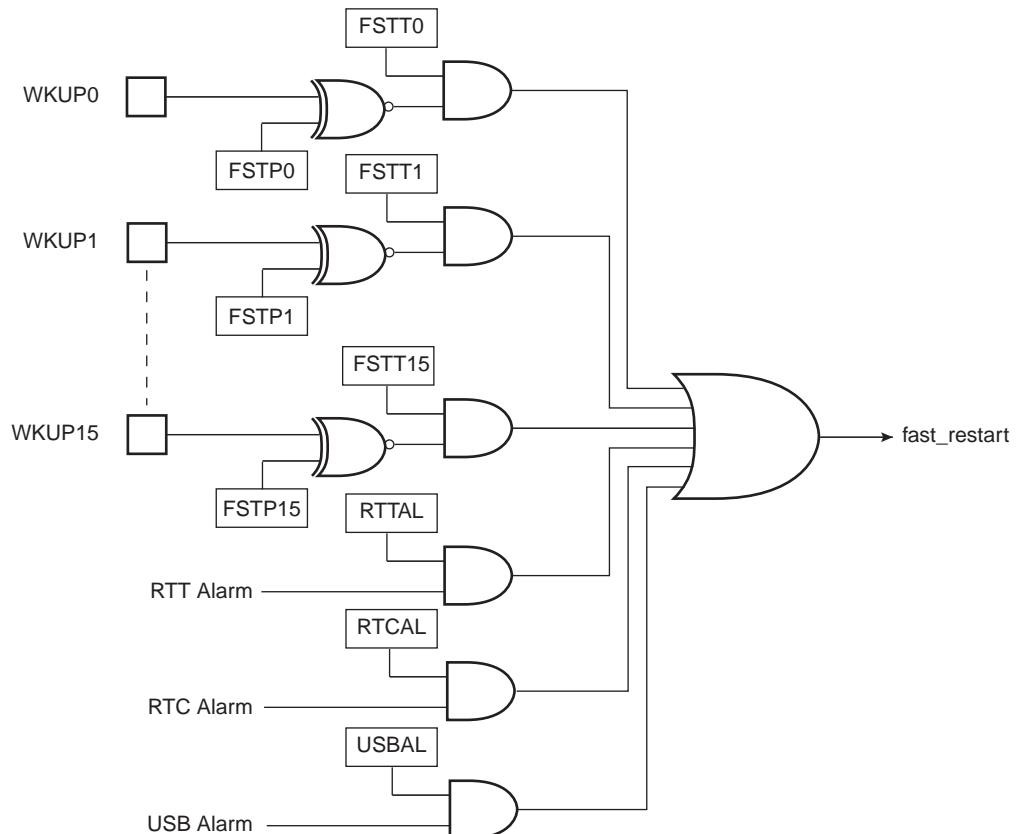
The Fast Restart circuitry, as shown in [Figure 30-8](#), is fully asynchronous and provides a fast start-up signal to the Power Management Controller. As soon as the fast start-up signal is asserted, the embedded 4/8/12 MHz Fast RC Oscillator restarts automatically.

When entering the Wait Mode, the embedded flash can be placed in low power mode depending on the configuration of the FLPM in PMC\_FSMR register. This bitfield can be programmed anytime and will be taken into account at the next time the system enters Wait Mode.

The power consumption reduction is optimal when configuring 1 (deep power down mode) in FLPM. If 0 is programmed (standby mode), the power consumption is slightly higher as compared to the deep power down mode.

When programming 2 in FLPM, the Wait Mode flash power consumption is equivalent to the active mode when there is no read access on the flash.

**Figure 30-8.** Fast Start-up Circuitry



Each wake-up input pin and alarm can be enabled to generate a Fast Start-up event by writing 1 to the corresponding bit in the Fast Start-up Mode Register (PMC\_FSMR).

The user interface does not provide any status for Fast Start-up, but the user can easily recover this information by reading the PIO Controller and the status registers of the RTC, RTT and USB Controller.

### 30.2.12 Main Crystal Clock Failure Detector

The clock failure detector monitors the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator to identify an eventual defect of this oscillator (for example, if the crystal is unconnected).

The clock failure detector can be enabled or disabled by means of the CFDEN bit in the PMC Clock Generator Main Oscillator Register (CKGR\_MOR). After reset, the detector is disabled. However, if the 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator is disabled, the clock failure detector is disabled too.

The slow RC oscillator must be enabled. The clock failure detection must be enabled only when system clock MCK selects the fast RC Oscillator. Then the status register must be read 2 slow clock cycles after enabling.

A failure is detected by means of a counter incrementing on the 3 to 20 MHz Crystal oscillator or Ceramic Resonator-based oscillator clock edge and timing logic clocked on the slow clock RC oscillator controlling the counter. The counter is cleared when the slow clock RC oscillator signal is low and enabled when the slow clock RC oscillator is high. Thus the failure detection time is 1 slow clock RC oscillator clock period. If, during the high level period of the slow clock RC oscillator, less than 8 fast crystal oscillator clock periods have been counted, then a failure is declared.

If a failure of the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator clock is detected, the CFDEV flag is set in the PMC Status Register (PMC\_SR), and generates an interrupt if it is not masked. The interrupt remains active until a read operation in the PMC\_SR register. The user can know the status of the clock failure detector at any time by reading the CFDS bit in the PMC\_SR register.

If the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator clock is selected as the source clock of MAINCK (MOSCSEL = 1), and if the Master Clock Source is PLLACK (CSS = 2), a clock failure detection automatically forces MAINCK to be the source clock for the master clock (MCK). Then, regardless of the PMC configuration, a clock failure detection automatically forces the 4/8/12 MHz Fast RC Oscillator to be the source clock for MAINCK. If the Fast RC Oscillator is disabled when a clock failure detection occurs, it is automatically re-enabled by the clock failure detection mechanism.

It takes 2 slow clock RC oscillator cycles to detect and switch from the 3 to 20 MHz Crystal, or Ceramic Resonator-based Oscillator, to the 4/8/12 MHz Fast RC Oscillator if the Master Clock source is Main Clock, or 3 slow clock RC oscillator cycles if the Master Clock source is PLLACK.

A clock failure detection activates a fault output that is connected to the Pulse Width Modulator (PWM) Controller. With this connection, the PWM controller is able to force its outputs and to protect the driven device, if a clock failure is detected.

The user can know the status of the clock failure detector at any time by reading the FOS bit in the PMC\_SR register.

This fault output remains active until the defect is detected and until it is cleared by the bit FOCLR in the PMC Fault Output Clear Register (PMC\_FOCR).

### 30.2.13 Programming Sequence

#### 1. Enabling the Main Oscillator:

The main oscillator is enabled by setting the MOSCXTEN field in the Main Oscillator Register (CKGR\_MOR). The user can define a start-up time. This can be achieved by writing a value in the MOSCXTST field in CKGR\_MOR. Once this register has been correctly configured, the user must wait for MOSCXTS field in the PMC\_SR register to be set. This can be done either by polling the status register, or by waiting the interrupt line to be raised if the associated interrupt to MOSCXTS has been enabled in the PMC\_IER register.

#### 2. Checking the Main Oscillator Frequency (Optional):

In some situations the user may need an accurate measure of the main clock frequency. This measure can be accomplished via the Main Clock Frequency Register (CKGR\_MCFR).

Once the MAINFRDY field is set in CKGR\_MCFR, the user may read the MAINF field in CKGR\_MCFR by performing another CKGR\_MCFR read access. This provides the number of main clock cycles within sixteen slow clock cycles.

#### 3. Setting PLL and Divider:

All parameters needed to configure PLLA and the divider are located in CKGR\_PLLAR.

The DIV field is used to control the divider itself. It must be set to 1 when PLL is used. By default, DIV parameter is set to 0 which means that the divider is turned off.

The MUL field is the PLL multiplier factor. This parameter can be programmed between 0 and 80. If MUL is set to 0, PLL will be turned off, otherwise the PLL output frequency is PLL input frequency multiplied by (MUL + 1).

The PLLCOUNT field specifies the number of slow clock cycles before the LOCK bit is set in PMC\_SR, after CKGR\_PLLAR has been written.

Once the CKGR\_PLL register has been written, the user must wait for the LOCK bit to be set in the PMC\_SR. This can be done either by polling the status register or by waiting the interrupt line to be raised if the associated interrupt to LOCK has been enabled in PMC\_IER. All parameters in CKGR\_PLLAR can be programmed in a single write operation. If at some stage one of the following parameters, MUL or DIV is modified, the LOCK bit will go low to indicate that PLL is not ready yet. When PLL is locked, LOCK will be set again. The user is constrained to wait for LOCK bit to be set before using the PLL output clock.

#### 4. Selection of Master Clock and Processor Clock

The Master Clock and the Processor Clock are configurable via the Master Clock Register (PMC\_MCKR).

The CSS field is used to select the Master Clock divider source. By default, the selected clock source is main clock.

The PRES field is used to control the Master Clock prescaler. The user can choose between different values (1, 2, 3, 4, 8, 16, 32, 64). Master Clock output is prescaler input divided by PRES parameter. By default, PRES parameter is set to 1 which means that master clock is equal to main clock.

Once PMC\_MCKR has been written, the user must wait for the MCKRDY bit to be set in PMC\_SR. This can be done either by polling the status register or by waiting for the interrupt line to be raised if the associated interrupt to MCKRDY has been enabled in the PMC\_IER register.

The PMC\_MCKR must not be programmed in a single write operation. The preferred programming sequence for PMC\_MCKR is as follows:

- If a new value for CSS field corresponds to PLL Clock,
  - Program the PRES field in PMC\_MCKR.
  - Wait for the MCKRDY bit to be set in PMC\_SR.
  - Program the CSS field in PMC\_MCKR.
  - Wait for the MCKRDY bit to be set in PMC\_SR.
- If a new value for CSS field corresponds to Main Clock or Slow Clock,
  - Program the CSS field in PMC\_MCKR.
  - Wait for the MCKRDY bit to be set in the PMC\_SR.
  - Program the PRES field in PMC\_MCKR.
  - Wait for the MCKRDY bit to be set in PMC\_SR.

If at some stage one of the following parameters, CSS or PRES is modified, the MCKRDY bit will go low to indicate that the Master Clock and the Processor Clock are not ready yet. The user must wait for MCKRDY bit to be set again before using the Master and Processor Clocks.

Note: IF PLLx clock was selected as the Master Clock and the user decides to modify it by writing in CKGR\_PLLR, the MCKRDY flag will go low while PLL is unlocked. Once PLL is locked again, LOCK goes high and MCKRDY is set.  
While PLL is unlocked, the Master Clock selection is automatically changed to Slow Clock. For further information, see [Section 30.2.14.2 "Clock Switching Waveforms" on page 548](#).

Code Example:

```
write_register(PMC_MCKR, 0x00000001)
wait (MCKRDY=1)
write_register(PMC_MCKR, 0x00000011)
wait (MCKRDY=1)
```

The Master Clock is main clock divided by 2.

The Processor Clock is the Master Clock.

## 5. Selection of Programmable Clocks

Programmable clocks are controlled via registers, PMC\_SCER, PMC\_SCDR and PMC\_SCSR.

Programmable clocks can be enabled and/or disabled via PMC\_SCER and PMC\_SCDR. 3 Programmable clocks can be enabled or disabled. The PMC\_SCSR provides a clear indication as to which Programmable clock is enabled. By default all Programmable clocks are disabled.

Programmable Clock Registers (PMC\_PCKx) are used to configure Programmable clocks.

The CSS field is used to select the Programmable clock divider source. Four clock options are available: main clock, slow clock, PLLACK, . By default, the clock source selected is slow clock.

The PRES field is used to control the Programmable clock prescaler. It is possible to choose between different values (1, 2, 4, 8, 16, 32, 64). Programmable clock output is prescaler input divided by PRES parameter. By default, the PRES parameter is set to 0 which means that master clock is equal to slow clock.

Once PMC\_PCKx has been programmed, The corresponding Programmable clock must be enabled and the user is constrained to wait for the PCKRDYx bit to be set in PMC\_SR. This can be done either by polling the status register or by waiting the interrupt line to be raised, if the associated interrupt to PCKRDYx has been enabled in the PMC\_IER register. All parameters in PMC\_PCKx can be programmed in a single write operation.

If the CSS and PRES parameters are to be modified, the corresponding Programmable clock must be disabled first. The parameters can then be modified. Once this has been done, the user must re-enable the Programmable clock and wait for the PCKRDYx bit to be set.

## 6. Enabling Peripheral Clocks

Once all of the previous steps have been completed, the peripheral clocks can be enabled and/or disabled via registers PMC\_PCER0, PMC\_PCER, PMC\_PCDR0 and PMC\_PCDR.

### 30.2.14 Clock Switching Details

#### 30.2.14.1 Master Clock Switching Timings

**Table 30-1** and give the worst case timings required for the Master Clock to switch from one selected clock to another one. This is in the event that the prescaler is de-activated. When the prescaler is activated, an additional time of 64 clock cycles of the newly selected clock has to be added.

**Table 30-1.** Clock Switching Timings (Worst Case)

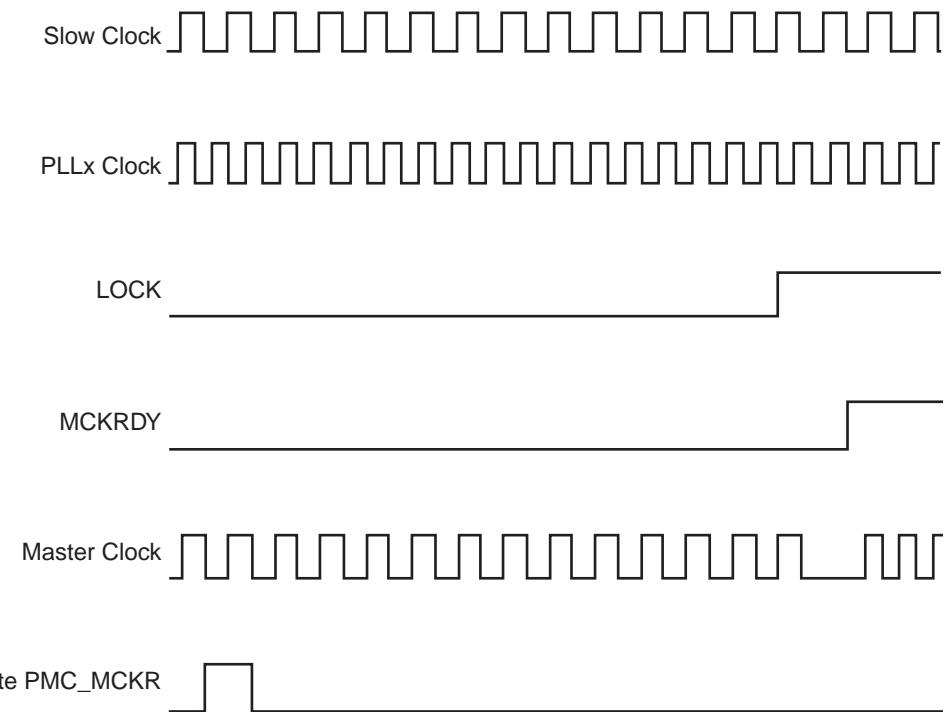
From To	Main Clock	SLCK	PLL Clock
Main Clock	–	4 x SLCK + 2.5 x Main Clock	3 x PLL Clock + 4 x SLCK + 1 x Main Clock
SLCK	0.5 x Main Clock + 4.5 x SLCK	–	3 x PLL Clock + 5 x SLCK
PLL Clock	0.5 x Main Clock + 4 x SLCK + PLLCOUNT x SLCK + 2.5 x PLLx Clock	2.5 x PLL Clock + 5 x SLCK + PLLCOUNT x SLCK	2.5 x PLL Clock + 4 x SLCK + PLLCOUNT x SLCK

Notes:

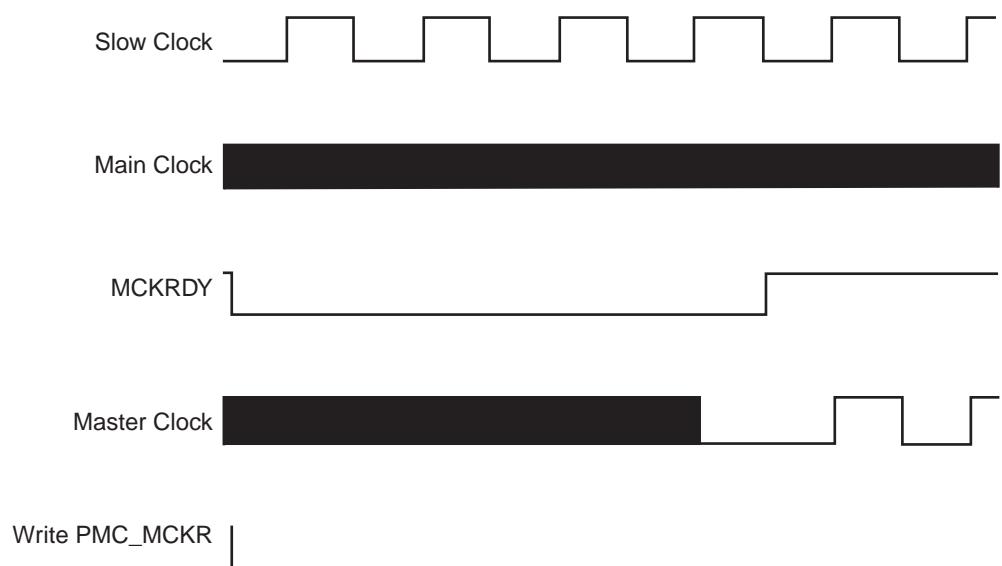
1. PLL designates the PLLA .
2. PLLCOUNT designates PLLACOUNT .

### 30.2.14.2 Clock Switching Waveforms

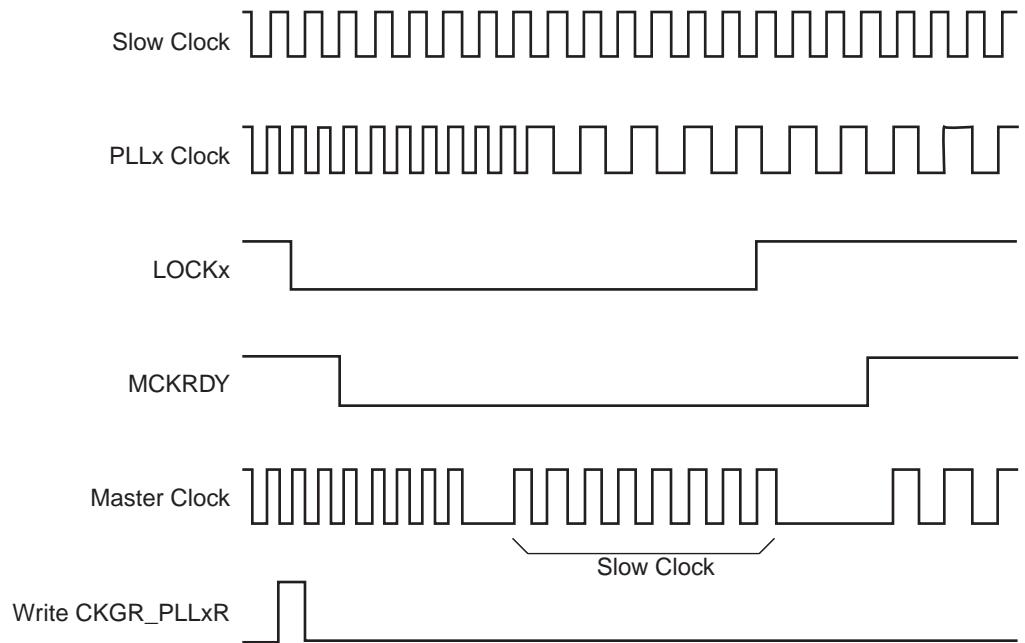
**Figure 30-9.** Switch Master Clock from Slow Clock to PLLx Clock



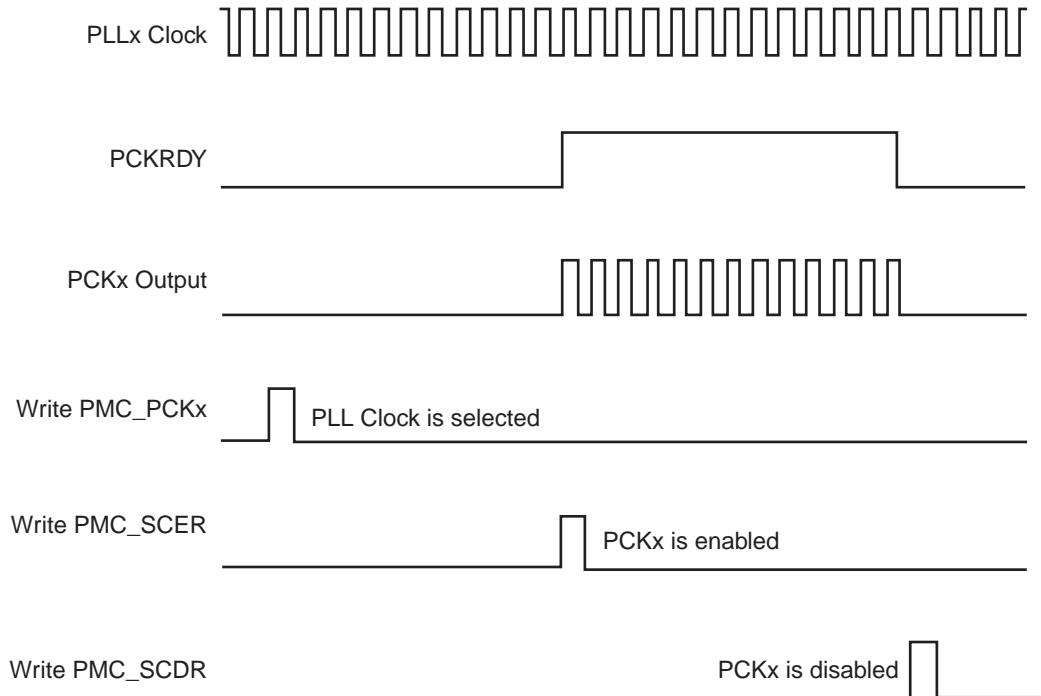
**Figure 30-10.** Switch Master Clock from Main Clock to Slow Clock



**Figure 30-11.** Change PLLx Programming



**Figure 30-12.** Programmable Clock Output Programming



### 30.2.15 Write Protection Registers

To prevent any single software error that may corrupt PMC behavior, certain address spaces can be write protected by setting the WPEN bit in the “PMC Write Protect Mode Register” (PMC\_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the PMC Write Protect Status Register (PMC\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the PMC Write Protect Mode Register (PMC\_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- “PMC System Clock Enable Register”
- “PMC System Clock Disable Register”
- “PMC Peripheral Clock Enable Register 0”
- “PMC Peripheral Clock Disable Register 0”
- “PMC Clock Generator Main Oscillator Register”
- “PMC Clock Generator PLLA Register”
- “PMC Master Clock Register”
- “PMC USB Clock Register”
- “PMC Programmable Clock Register”
- “PMC Fast Start-up Mode Register”
- “PMC Fast Start-up Polarity Register”
- “PMC Peripheral Clock Enable Register 1”
- “PMC Peripheral Clock Disable Register 1”
- “PMC Oscillator Calibration Register”

### 30.2.16 Power Management Controller (PMC) User Interface

**Table 30-2.** Register Mapping

Offset	Register	Name	Access	Reset
0x0000	System Clock Enable Register	PMC_SCER	Write-only	—
0x0004	System Clock Disable Register	PMC_SCDR	Write-only	—
0x0008	System Clock Status Register	PMC_SCSR	Read-only	0x0000_0001
0x000C	Reserved	—	—	—
0x0010	Peripheral Clock Enable Register 0	PMC_PCER0	Write-only	—
0x0014	Peripheral Clock Disable Register 0	PMC_PCDR0	Write-only	—
0x0018	Peripheral Clock Status Register 0	PMC_PCSR0	Read-only	0x0000_0000
0x001C	Reserved	—	—	—
0x0020	Main Oscillator Register	CKGR_MOR	Read-write	0x0000_0008
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read-write	0x0000_0000
0x0028	PLLA Register	CKGR_PLLAR	Read-write	0x0000_3F00
0x002C	Reserved	—	—	—
0x0030	Master Clock Register	PMC_MCKR	Read-write	0x0000_0001
0x0034	Reserved	—	—	—
0x0038	USB Clock Register	PMC_USB	Read/Write	0x0000_0000
0x003C	Reserved	—	—	—
0x0040	Programmable Clock 0 Register	PMC_PCK0	Read-write	0x0000_0000
0x0044	Programmable Clock 1 Register	PMC_PCK1	Read-write	0x0000_0000
0x0048	Programmable Clock 2 Register	PMC_PCK2	Read-write	0x0000_0000
0x004C - 0x005C	Reserved	—	—	—
0x0060	Interrupt Enable Register	PMC_IER	Write-only	—
0x0064	Interrupt Disable Register	PMC_IDR	Write-only	—
0x0068	Status Register	PMC_SR	Read-only	0x0001_0008
0x006C	Interrupt Mask Register	PMC_IMR	Read-only	0x0000_0000
0x0070	Fast Start-up Mode Register	PMC_FSMR	Read-write	0x0000_0000
0x0074	Fast Start-up Polarity Register	PMC_FSPR	Read-write	0x0000_0000
0x0078	Fault Output Clear Register	PMC_FOCR	Write-only	—
0x007C- 0x00E0	Reserved	—	—	—
0x00E4	Write Protect Mode Register	PMC_WPMR	Read-write	0x0
0x00E8	Write Protect Status Register	PMC_WPSR	Read-only	0x0
0x00EC-0x00FC	Reserved	—	—	—
0x0100	Peripheral Clock Enable Register 1	PMC_PCER1	Write-only	—
0x0104	Peripheral Clock Disable Register 1	PMC_PCDR1	Write-only	—
0x0108	Peripheral Clock Status Register 1	PMC_PCSR1	Read-only	0x0000_0000

**Table 30-2.** Register Mapping

Offset	Register	Name	Access	Reset
0x010C	Reserved	—	—	—
0x0110	Oscillator Calibration Register	PMC_OCR	Read-write	0x0040_4040
0x130	PLL Maximum Multiplier Value Register	PMC_PMMR	Read-Write	0x07FF07FF

Note: If an offset is not listed in the table it must be considered as “reserved”.

### 30.2.16.1 PMC System Clock Enable Register

**Name:** PMC\_SCER

**Address:** 0x400E0400

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	—	—	—	—	—	—	—

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **UDP: USB Device Port Clock Enable**

0 = No effect.

1 = Enables the 48 MHz clock (UDPCK) of the USB Device Port.

- **PCKx: Programmable Clock x Output Enable**

0 = No effect.

1 = Enables the corresponding Programmable Clock output.

**30.2.16.2 PMC System Clock Disable Register**

**Name:** PMC\_SCDR

**Address:** 0x400E0404

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	—	—	—	—	—	—	—

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **UDP: USB Device Port Clock Disable**

0 = No effect.

1 = Disables the 48 MHz clock (UDPCK) of the USB Device Port.

- **PCKx: Programmable Clock x Output Disable**

0 = No effect.

1 = Disables the corresponding Programmable Clock output.

### 30.2.16.3 PMC System Clock Status Register

**Name:** PMC\_SCSR

**Address:** 0x400E0408

**Access:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	-	-	-	-	-	-	-

- **UDP: USB Device Port Clock Status**

0 = The 48 MHz clock (UDPCK) of the USB Device Port is disabled.

1 = The 48 MHz clock (UDPCK) of the USB Device Port is enabled.

- **PCKx: Programmable Clock x Output Status**

0 = The corresponding Programmable Clock output is disabled.

1 = The corresponding Programmable Clock output is enabled.

### 30.2.16.4 PMC Peripheral Clock Enable Register 0

**Name:** PMC\_PCER0

**Address:** 0x400E0410

**Access:** Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	-	-	-	-	-	-	-

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **PIDx: Peripheral Clock x Enable**

0 = No effect.

1 = Enables the corresponding peripheral clock.

Note: To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet. Other peripherals can be enabled in PMC\_PCER1 ([Section 30.2.16.22 “PMC Peripheral Clock Enable Register 1”](#)).

Note: Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

**30.2.16.5 PMC Peripheral Clock Disable Register 0**

**Name:** PMC\_PCDR0

**Address:** 0x400E0414

**Access:** Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	-	-	-	-	-	-	-

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **PIDx: Peripheral Clock x Disable**

0 = No effect.

1 = Disables the corresponding peripheral clock.

Note: To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet. Other peripherals can be disabled in PMC\_PCDR1 ([Section 30.2.16.23 "PMC Peripheral Clock Disable Register 1"](#)).

**30.2.16.6 PMC Peripheral Clock Status Register 0**

**Name:** PMC\_PCSR0

**Address:** 0x400E0418

**Access:** Read-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	-	-	-	-	-	-	-

- **PIDx: Peripheral Clock x Status**

0 = The corresponding peripheral clock is disabled.

1 = The corresponding peripheral clock is enabled.

Note: To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet. Other peripherals status can be read in PMC\_PCSR1 ([Section 30.2.16.24 "PMC Peripheral Clock Status Register 1"](#)).

### 30.2.16.7 PMC Clock Generator Main Oscillator Register

**Name:** CKGR\_MOR

**Address:** 0x400E0420

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	CFDEN	MOSCSEL
23	22	21	20	19	18	17	16
KEY							
15	14	13	12	11	10	9	8
MOSCXTST							
7	6	5	4	3	2	1	0
-	MOSCRCF			MOSCRCEN	WAITMODE	MOSCXTBY	MOSCXTEN

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **MOSCXTEN: Main Crystal Oscillator Enable**

A crystal must be connected between XIN and XOUT.

0 = The Main Crystal Oscillator is disabled.

1 = The Main Crystal Oscillator is enabled. MOSCXTBY must be set to 0.

When MOSCXTEN is set, the MOSCXTS flag is set once the Main Crystal Oscillator start-up time is achieved.

- **MOSCXTBY: Main Crystal Oscillator Bypass**

0 = No effect.

1 = The Main Crystal Oscillator is bypassed. MOSCXTEN must be set to 0. An external clock must be connected on XIN.

When MOSCXTBY is set, the MOSCXTS flag in PMC\_SR is automatically set.

Clearing MOSCXTEN and MOSCXTBY bits allows resetting the MOSCXTS flag.

- **WAITMODE: Wait Mode Command**

0 = No effect.

1 = Enters the device in Wait Mode.

Note: The WAITMODE bit is write-only.

- **MOSCRCEN: Main On-Chip RC Oscillator Enable**

0 = The Main On-Chip RC Oscillator is disabled.

1 = The Main On-Chip RC Oscillator is enabled.

When MOSCRCEN is set, the MOSCRCS flag is set once the Main On-Chip RC Oscillator start-up time is achieved.

- **MOSCRCF: Main On-Chip RC Oscillator Frequency Selection**

At start-up, the Main On-Chip RC Oscillator frequency is **4** MHz.

Value	Name	Description
0x0	4_MHz	The Fast RC Oscillator Frequency is at 4 MHz (default)
0x1	8_MHz	The Fast RC Oscillator Frequency is at 8 MHz
0x2	12_MHz	The Fast RC Oscillator Frequency is at 12 MHz

Note: MOSCRCF must be changed only if MOSCRCS is set in the PMC\_SR register. Therefore MOSCRCF and MOSCRCEN cannot be changed at the same time.

- **MOSCXTST: Main Crystal Oscillator Start-up Time**

Specifies the number of Slow Clock cycles multiplied by 8 for the Main Crystal Oscillator start-up time.

- **KEY: Write Access Password**

Value	Name	Description
0x37	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.

- **MOSCSEL: Main Oscillator Selection**

0 = The Main On-Chip RC Oscillator is selected.

1 = The Main Crystal Oscillator is selected.

- **CFDEN: Clock Failure Detector Enable**

0 = The Clock Failure Detector is disabled.

1 = The Clock Failure Detector is enabled.

Note: 1. The slow RC oscillator must be enabled when the CFDEN is enabled.

2. The clock failure detection must be enabled only when system clock MCK selects the fast RC Oscillator.

3. Then the status register must be read 2 slow clock cycles after enabling.

### 30.2.16.8 PMC Clock Generator Main Clock Frequency Register

**Name:** CKGR\_MCFR

**Address:** 0x400E0424

**Access:** Read-Write

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

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **MAINF: Main Clock Frequency**

Gives the number of Main Clock cycles within 16 Slow Clock periods.

- **MAINFRDY: Main Clock Ready**

0 = MAINF value is not valid or the Main Oscillator is disabled or a measure has just been started by means of RCMEAS.

1 = The Main Oscillator has been enabled previously and MAINF value is available.

Note: To ensure that a correct value is read on the MAINF bitfield, the MAINFRDY flag must be read at 1 then another read access must be performed on the register to get a stable value on the MAINF bitfield.

- **RCMEAS: RC Oscillator Frequency Measure (write-only)**

0 = No effect.

1 = Restarts measuring of the main RC frequency. MAINF will carry the new frequency as soon as a low to high transition occurs on the MAINFRDY flag.

The measure is performed on the main frequency (i.e. not limited to RC oscillator only), but if the main clock frequency source is the fast crystal oscillator, the restart of measuring is not needed because of the well known stability of crystal oscillators.

### 30.2.16.9 PMC Clock Generator PLLA Register

**Name:** CKGR\_PLLAR

**Address:** 0x400E0428

**Access:** Read-write

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

Possible limitations on PLLA input frequencies and multiplier factors should be checked before using the PMC.

**Warning:** Bit 29 must always be set to 1 when programming the CKGR\_PLLAR register.

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **DIVA: PLLA Front\_End Divider**

0 = Divider output is stuck at 0 and PLLA is disabled.

1 = Divider is bypassed (divide by 1) PLLA is enabled

2 up to 255 = clock is divided by DIVA

- **PLLACOUNT: PLLA Counter**

Specifies the number of Slow Clock cycles before the LOCKA bit is set in PMC\_SR after CKGR\_PLLAR is written.

- **MULA: PLLA Multiplier**

0 = The PLLA is deactivated (PLLA also disabled if DIVA = 0).

1 up to 80 = The PLLA Clock frequency is the PLLA input frequency multiplied by MULA + 1.

- **ONE: Must Be Set to 1**

Bit 29 must always be set to 1 when programming the CKGR\_PLLAR register.

### 30.2.16.10 PMC Master Clock Register

**Name:** PMC\_MCKR

**Address:** 0x400E0430

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	PLLADIV2	—	—	—	—
7	6	5	4	3	2	1	0
—	PRES			—	—	CSS	

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **CSS: Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected

- **PRES: Processor Clock Prescaler**

Value	Name	Description
0	CLK_1	Selected clock
1	CLK_2	Selected clock divided by 2
2	CLK_4	Selected clock divided by 4
3	CLK_8	Selected clock divided by 8
4	CLK_16	Selected clock divided by 16
5	CLK_32	Selected clock divided by 32
6	CLK_64	Selected clock divided by 64
7	CLK_3	Selected clock divided by 3

- **PLLADIV2: PLLA Divisor by 2**

PLLADIV2	PLLA Clock Division
0	PLLA clock frequency is divided by 1.
1	PLLA clock frequency is divided by 2.

### 30.2.16.11 PMC USB Clock Register

**Name:** PMC\_USB

**Address:** 0x400E0438

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **USBDIV: Divider for USB Clock**

USB Clock is Input clock divided by USBDIV+1.

### 30.2.16.12 PMC Programmable Clock Register

**Name:** PMC\_PCKx

**Address:** 0x400E0440

**Access:** Read-write

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
—	PRES			—	CSS		

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **CSS: Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected
4	MCK	Master Clock is selected

- **PRES: Programmable Clock Prescaler**

Value	Name	Description
0	CLK_1	Selected clock
1	CLK_2	Selected clock divided by 2
2	CLK_4	Selected clock divided by 4
3	CLK_8	Selected clock divided by 8
4	CLK_16	Selected clock divided by 16
5	CLK_32	Selected clock divided by 32
6	CLK_64	Selected clock divided by 64

### 30.2.16.13 PMC Interrupt Enable Register

**Name:** PMC\_IER

**Address:** 0x400E0460

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
—	—	—	—	—	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
—	—	—	—	MCKRDY	—	LOCKA	MOSCXTS

- **MOSCXTS:** Main Crystal Oscillator Status Interrupt Enable
- **LOCKA:** PLLA Lock Interrupt Enable
- **MCKRDY:** Master Clock Ready Interrupt Enable
- **PCKRDYx:** Programmable Clock Ready x Interrupt Enable
- **MOSCSELS:** Main Oscillator Selection Status Interrupt Enable
- **MOSCRCs:** Main On-Chip RC Status Interrupt Enable
- **CFDEV:** Clock Failure Detector Event Interrupt Enable

### 30.2.16.14 PMC Interrupt Disable Register

**Name:** PMC\_IDR

**Address:** 0x400E0464

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
—	—	—	—	—	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
—	—	—	—	MCKRDY	—	LOCKA	MOSCXTS

- **MOSCXTS:** Main Crystal Oscillator Status Interrupt Disable
- **LOCKA:** PLLA Lock Interrupt Disable
- **MCKRDY:** Master Clock Ready Interrupt Disable
- **PCKRDYx:** Programmable Clock Ready x Interrupt Disable
- **MOSCSELS:** Main Oscillator Selection Status Interrupt Disable
- **MOSCRCs:** Main On-Chip RC Status Interrupt Disable
- **CFDEV:** Clock Failure Detector Event Interrupt Disable

### 30.2.16.15 PMC Status Register

**Name:** PMC\_SR

**Address:** 0x400E0468

**Access:** Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	FOS	CFDS	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
—	—	—	—	—	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
OSCSELS	—	—	—	MCKRDY	—	LOCKA	MOSCXTS

- **MOSCXTS: Main XTAL Oscillator Status**

0 = Main XTAL oscillator is not stabilized.

1 = Main XTAL oscillator is stabilized.

- **LOCKA: PLLA Lock Status**

0 = PLLA is not locked

1 = PLLA is locked.

- **MCKRDY: Master Clock Status**

0 = Master Clock is not ready.

1 = Master Clock is ready.

- **OSCSELS: Slow Clock Oscillator Selection**

0 = Internal slow clock RC oscillator is selected.

1 = External slow clock 32 kHz oscillator is selected.

- **PCKRDYx: Programmable Clock Ready Status**

0 = Programmable Clock x is not ready.

1 = Programmable Clock x is ready.

- **MOSCSELS: Main Oscillator Selection Status**

0 = Selection is in progress.

1 = Selection is done.

- **MOSCRCS: Main On-Chip RC Oscillator Status**

0 = Main on-chip RC oscillator is not stabilized.

1 = Main on-chip RC oscillator is stabilized.

- **CFDEV: Clock Failure Detector Event**

0 = No clock failure detection of the fast crystal oscillator clock has occurred since the last read of PMC\_SR.  
1 = At least one clock failure detection of the fast crystal oscillator clock has occurred since the last read of PMC\_SR.

- **CFDS: Clock Failure Detector Status**

0 = A clock failure of the fast crystal oscillator clock is not detected.

1 = A clock failure of the fast crystal oscillator clock is detected.

- **FOS: Clock Failure Detector Fault Output Status**

0 = The fault output of the clock failure detector is inactive.

1 = The fault output of the clock failure detector is active.

### 30.2.16.16 PMC Interrupt Mask Register

**Name:** PMC\_IMR

**Address:** 0x400E046C

**Access:** Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
—	—	—	—	—	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
—	—	—	—	MCKRDY	—	LOCKA	MOSCXTS

- **MOSCXTS:** Main Crystal Oscillator Status Interrupt Mask
- **LOCKA:** PLLA Lock Interrupt Mask
- **MCKRDY:** Master Clock Ready Interrupt Mask
- **PCKRDYx:** Programmable Clock Ready x Interrupt Mask
- **MOSCSELS:** Main Oscillator Selection Status Interrupt Mask
- **MOSCRCs:** Main On-Chip RC Status Interrupt Mask
- **CFDEV:** Clock Failure Detector Event Interrupt Mask

### 30.2.16.17 PMC Fast Start-up Mode Register

**Name:** PMC\_FSMR

**Address:** 0x400E0470

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	FLPM	LPM	-	USBAL	RTCAL	RTTAL	
15	14	13	12	11	10	9	8
FSTT15	FSTT14	FSTT13	FSTT12	FSTT11	FSTT10	FSTT9	FSTT8
7	6	5	4	3	2	1	0
FSTT7	FSTT6	FSTT5	FSTT4	FSTT3	FSTT2	FSTT1	FSTT0

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register”.

- **FSTT0 - FSTT15: Fast Start-up Input Enable 0 to 15**

0 = The corresponding wake-up input has no effect on the Power Management Controller.

1 = The corresponding wake-up input enables a fast restart signal to the Power Management Controller.

- **RTTAL: RTT Alarm Enable**

0 = The RTT alarm has no effect on the Power Management Controller.

1 = The RTT alarm enables a fast restart signal to the Power Management Controller.

- **RTCAL: RTC Alarm Enable**

0 = The RTC alarm has no effect on the Power Management Controller.

1 = The RTC alarm enables a fast restart signal to the Power Management Controller.

- **USBAL: USB Alarm Enable**

0 = The USB alarm has no effect on the Power Management Controller.

1 = The USB alarm enables a fast restart signal to the Power Management Controller.

- **LPM: Low Power Mode**

0 = The WaitForInterrupt (WFI) or the WaitForEvent (WFE) instruction of the processor makes the processor enter Sleep Mode.

1 = The WaitForEvent (WFE) instruction of the processor makes the system to enter in Wait Mode.

- **FLPM: Flash Low Power Mode**

Value	Name	Description
0	FLASH_STANDBY	Flash is in Standby Mode when system enters Wait Mode
1	FLASH_DEEP_POWERDOWN	Flash is in deep power down mode when system enters Wait Mode
2	FLASH_IDLE	idle mode

### 30.2.16.18 PMC Fast Start-up Polarity Register

**Name:** PMC\_FSPR

**Address:** 0x400E0474

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
FSTP15	FSTP14	FSTP13	FSTP12	FSTP11	FSTP10	FSTP9	FSTP8
7	6	5	4	3	2	1	0
FSTP7	FSTP6	FSTP5	FSTP4	FSTP3	FSTP2	FSTP1	FSTP0

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **FSTPx: Fast Start-up Input Polarity**

Defines the active polarity of the corresponding wake-up input. If the corresponding wake-up input is enabled and at the FSTP level, it enables a fast restart signal.

### 30.2.16.19 PMC Fault Output Clear Register

**Name:** PMC\_FOCR

**Address:** 0x400E0478

**Access:** Write-only

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
-	-	-	-	-	-	-	FOCLR

- **FOCLR: Fault Output Clear**

Clears the clock failure detector fault output.

### 30.2.16.20 PMC Write Protect Mode Register

**Name:** PMC\_WPMR

**Address:** 0x400E04E4

**Access:** Read-write

**Reset:** See Table 30-2

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x504D43 ("PMC" in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x504D43 ("PMC" in ASCII).

Protects the registers:

- "PMC System Clock Enable Register"
- "PMC System Clock Disable Register"
- "PMC Peripheral Clock Enable Register 0"
- "PMC Peripheral Clock Disable Register 0"
- "PMC Clock Generator Main Oscillator Register"
- "PMC Clock Generator PLLA Register"
- "PMC Master Clock Register"
- "PMC USB Clock Register"
- "PMC Programmable Clock Register"
- "PMC Fast Start-up Mode Register"
- "PMC Fast Start-up Polarity Register"
- "PMC Peripheral Clock Enable Register 1"
- "PMC Peripheral Clock Disable Register 1"
- "PMC Oscillator Calibration Register"

- **WPKEY: Write Protect KEY**

Value	Name	Description
0x504D43	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

**30.2.16.21 PMC Write Protect Status Register**

**Name:** PMC\_WPSR

**Address:** 0x400E04E8

**Access:** Read-only

**Reset:** See [Table 30-2](#)

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

- **WPVS: Write Protect Violation Status**

0 = No Write Protect Violation has occurred since the last read of the PMC\_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the PMC\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Reading PMC\_WPSR automatically clears all fields.

### 30.2.16.22 PMC Peripheral Clock Enable Register 1

**Name:** PMC\_PCER1

**Address:** 0x400E0500

**Access:** Write-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
PID47	PID46	PID45	PID44	PID43	PID42	PID41	PID40
7	6	5	4	3	2	1	0
PID39	PID38	PID37	PID36	PID35	PID34	PID33	PID32

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **PIDx: Peripheral Clock x Enable**

0 = No effect.

1 = Enables the corresponding peripheral clock.

Notes: 1. To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.  
2. Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

### 30.2.16.23 PMC Peripheral Clock Disable Register 1

**Name:** PMC\_PCDR1

**Address:** 0x400E0504

**Access:** Write-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
PID47	PID46	PID45	PID44	PID43	PID42	PID41	PID40
7	6	5	4	3	2	1	0
PID39	PID38	PID37	PID36	PID35	PID34	PID33	PID32

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **PIDx: Peripheral Clock x Disable**

0 = No effect.

1 = Disables the corresponding peripheral clock.

Note: To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

### 30.2.16.24 PMC Peripheral Clock Status Register 1

**Name:** PMC\_PCSR1

**Address:** 0x400E0508

**Access:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
PID47	PID46	PID45	PID44	PID43	PID42	PID41	PID40
7	6	5	4	3	2	1	0
PID39	PID38	PID37	PID36	PID35	PID34	PID33	PID32

- **PIDx: Peripheral Clock x Status**

0 = The corresponding peripheral clock is disabled.

1 = The corresponding peripheral clock is enabled.

Note: To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

### 30.2.16.25 PMC Oscillator Calibration Register

**Name:** PMC\_OCR

**Address:** 0x400E0510

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
SEL12				CAL12			
15	14	13	12	11	10	9	8
SEL8				CAL8			
7	6	5	4	3	2	1	0
SEL4				CAL4			

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **CAL4: RC Oscillator Calibration bits for 4 MHz**

Calibration bits applied to the RC Oscillator when SEL4 is set.

- **SEL4: Selection of RC Oscillator Calibration bits for 4 MHz**

0 = Default value stored in Flash memory.

1 = Value written by user in CAL4 field of this register.

- **CAL8: RC Oscillator Calibration bits for 8 MHz**

Calibration bits applied to the RC Oscillator when SEL8 is set.

- **SEL8: Selection of RC Oscillator Calibration bits for 8 MHz**

0 = Factory determined value stored in Flash memory.

1 = Value written by user in CAL8 field of this register.

- **CAL12: RC Oscillator Calibration bits for 12 MHz**

Calibration bits applied to the RC Oscillator when SEL12 is set.

- **SEL12: Selection of RC Oscillator Calibration bits for 12 MHz**

0 = Factory determined value stored in Flash memory.

1 = Value written by user in CAL12 field of this register.

### 30.2.16.26 PLL Maximum Multiplier Value Register

**Name:** PMC\_PMMR

**Address:** 0x400E0530

**Access:** Read-Write

**Reset:** 0x00000000

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

This register can only be written if the WPEN bit is cleared in “[PMC Write Protect Mode Register](#)” .

- **PLLA\_MMAX: PLLA Maximum Allowed Multiplier Value**

Defines the maximum value of multiplication factor that can be sent to PLLA. Any value of the MULA bitfield (see “[PMC Clock Generator PLLA Register](#)” ) above PLLA\_MMAX is saturated to PLLA\_MMAX. PLLA\_MMAX write operation is cancelled in the following cases:

- The value of MULA is currently saturated by PLLA\_MMAX
- The user is trying to write a value of PLLA\_MMAX that is smaller than the current value of MULA

# 31. Advanced Encryption Standard (AES)

## 31.1 Description

The Advanced Encryption Standard (AES) is compliant with the American *FIPS (Federal Information Processing Standard) Publication 197* specification.

The AES supports all five confidentiality modes of operation for symmetrical key block cipher algorithms (ECB, CBC, OFB, CFB and CTR), as specified in the *NIST Special Publication 800-38A Recommendation*. It is compatible with all these modes via Peripheral DMA Controller channels, minimizing processor intervention for large buffer transfers.

The 128-bit/192-bit/256-bit key is stored in four/six/eight 32-bit registers (AES\_KEYWRx) which are all write-only.

The 128-bit input data and initialization vector (for some modes) are each stored in four 32-bit registers (AES\_IDATARx and AES\_IVRx) which are all write-only.

As soon as the initialization vector, the input data and the key are configured, the encryption/decryption process may be started. Then the encrypted/decrypted data are ready to be read out on the four 32-bit output data registers (AES\_ODATARx) or through the DMA channels.

## 31.2 Embedded Characteristics

- Compliant with *FIPS Publication 197, Advanced Encryption Standard (AES)*
- 128-bit/192-bit/256-bit Cryptographic Key
- 12/14/16 Clock Cycles Encryption/Decryption Processing Time with a 128-bit/192-bit/256-bit Cryptographic Key
- Double Input Buffer Optimizes Runtime
- Support of the Five Standard Modes of Operation Specified in the *NIST Special Publication 800-38A, Recommendation for Block Cipher Modes of Operation - Methods and Techniques*:
  - Electronic Code Book (ECB)
  - Cipher Block Chaining (CBC) including CBC-MAC
  - Cipher Feedback (CFB)
  - Output Feedback (OFB)
  - Counter (CTR)
- 8-bit, 16-bit, 32-bit, 64-bit and 128-bit Data Sizes Possible in CFB Mode
- Last Output Data Mode Allows Optimized Message Authentication Code (MAC) Generation
- Connection to DMA Optimizes Data Transfers for all Operating Modes

## 31.3 Product Dependencies

### 31.3.1 Power Management

The AES may be clocked through the Power Management Controller (PMC), so the programmer must first configure the PMC to enable the AES clock.

### 31.3.2 Interrupt

The AES interface has an interrupt line connected to the Interrupt Controller.

Handling the AES interrupt requires programming the Interrupt Controller before configuring the AES.

**Table 31-1. Peripheral IDs**

Instance	ID
AES	39

## 31.4 Functional Description

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information.

Encryption converts data to an unintelligible form called ciphertext. Decrypting the ciphertext converts the data back into its original form, called plaintext. The CIPHER bit in the AES Mode Register (AES\_MR) allows selection between the encryption and the decryption processes.

The AES is capable of using cryptographic keys of 128/192/256 bits to encrypt and decrypt data in blocks of 128 bits. This 128-bit/192-bit/256-bit key is defined in the Key Registers (AES\_KEYWRx).

The input to the encryption processes of the CBC, CFB, and OFB modes includes, in addition to the plaintext, a 128-bit data block called the initialization vector (IV), which must be set in the Initialization Vector Registers (AES\_IVRx). The initialization vector is used in an initial step in the encryption of a message and in the corresponding decryption of the message. The Initialization Vector Registers are also used by the CTR mode to set the counter value.

### 31.4.1 Operation Modes

The AES supports the following modes of operation:

- ECB: Electronic Code Book
- CBC: Cipher Block Chaining
- OFB: Output Feedback
- CFB: Cipher Feedback
  - CFB8 (CFB where the length of the data segment is 8 bits)
  - CFB16 (CFB where the length of the data segment is 16 bits)
  - CFB32 (CFB where the length of the data segment is 32 bits)
  - CFB64 (CFB where the length of the data segment is 64 bits)
  - CFB128 (CFB where the length of the data segment is 128 bits)
- CTR: Counter

The data pre-processing, post-processing and data chaining for the concerned modes are automatically performed. Refer to the *NIST Special Publication 800-38A Recommendation* for more complete information.

These modes are selected by setting the OPMOD field in the AES Mode Register (AES\_MR).

In CFB mode, five data sizes are possible (8, 16, 32, 64 or 128 bits), configurable by means of the CFBS field in the mode register. ([Section 31.6.2 “AES Mode Register” on page 590](#)).

In CTR mode, the size of the block counter embedded in the module is 16 bits. Therefore, there is a rollover after processing 1 megabyte of data. If the file to be processed is greater than 1 megabyte, this file must be split into fragments of 1 megabyte. Prior to loading the first fragment into AES\_IDATARx registers, the AES\_IVRx registers must be cleared. For any fragment, after the transfer is completed and prior to transferring the next fragment, the AES\_IVR0 register must be programmed so that the fragment number (0 for the first fragment, 1 for the second one, and so on) is written in the 16 MSB of the AES\_IVR0 register.

### 31.4.2 Double Input Buffer

The input data register can be double-buffered to reduce the runtime of large files.

This mode allows writing a new message block when the previous message block is being processed. This is only possible when DMA accesses are performed (SMOD = 0x2).

The DUALBUFF field in AES\_MR register must be set to 1 to get access to double buffer.

### 31.4.3 Start Modes

The SMOD field in the AES Mode Register (AES\_MR) allows selection of the encryption (or decryption) start mode.

#### 31.4.3.1 Manual Mode

The sequence order is as follows:

- Write the Mode Register (AES\_MR) with all required fields, including but not limited to SMOD and OPMOD.
- Write the 128-bit/192-bit/256-bit key in the Key Registers (AES\_KEYWRx).
- Write the initialization vector (or counter) in the Initialization Vector Registers (AES\_IVRx).

Note: The Initialization Vector Registers concern all modes except ECB.

- Set the bit DATRDY (Data Ready) in the AES Interrupt Enable register (AES\_IER), depending on whether an interrupt is required or not at the end of processing.
- Write the data to be encrypted/decrypted in the authorized Input Data Registers (See [Table 31-2](#)).

**Table 31-2. Authorized Input Data Registers**

Operation Mode	Input Data Registers to Write
ECB	All
CBC	All
OFB	All
128-bit CFB	All
64-bit CFB	AES_IDATAR0 and AES_IDATAR1
32-bit CFB	AES_IDATAR0
16-bit CFB	AES_IDATAR0
8-bit CFB	AES_IDATAR0
CTR	All

Note: In 64-bit CFB mode, writing to AES\_IDATAR2 and AES\_IDATAR3 registers is not allowed and may lead to errors in processing.

Note: In 32-, 16- and 8-bit CFB modes, writing to AES\_IDATAR1, AES\_IDATAR2 and AES\_IDATAR3 registers is not allowed and may lead to errors in processing.

- Set the START bit in the AES Control register AES\_CR to begin the encryption or the decryption process.
- When processing completes, the DATRDY bit in the AES Interrupt Status Register (AES\_ISR) raises. If an interrupt has been enabled by setting the DATRDY bit in AES\_IER, the interrupt line of the AES is activated.
- When the software reads one of the Output Data Registers (AES\_ODATARx), the DATRDY bit is automatically cleared.

### 31.4.3.2 Auto Mode

The Auto Mode is similar to the manual one, except that in this mode, as soon as the correct number of Input Data registers is written, processing is automatically started without any action in the Control Register.

### 31.4.4 DMA Mode

The DMA Controller can be used in association with the AES to perform an encryption/decryption of a buffer without any action by the software during processing.

The SMOD field of the AES\_MR must be set to 0x2 and the DMA must be configured with non incremental addresses.

The start address of any transfer descriptor must be set to AES\_IDATAR0 register.

The DMA chunk size configuration depends on the AES mode of operation and is listed in [Table 31-3 "DMA Data Transfer Type for the Different Operation Modes"](#).

When writing data to AES with a first DMA channel, data are first fetched from a memory buffer (source data). It is recommended to configure the size of source data to "words" even for CFB modes. On the contrary, the destination data size depends on the mode of operation. When reading data from the AES with the second DMA channel, the source data is the data read from AES and data destination is the memory buffer. In this case, source data size depends on the AES mode of operation and is listed in [Table 31-3..](#)

**Table 31-3. DMA Data Transfer Type for the Different Operation Modes**

Operation Mode	Chunk Size	Destination/Source Data Transfer Type
ECB	4	Word
CBC	4	Word
OFB	4	Word
CFB 128-bit	4	Word
CFB 64-bit	1	Word
CFB 32-bit	1	Word
CFB 16-bit	1	Half-word
CFB 8-bit	1	Byte
CTR	4	Word

### 31.4.5 Last Output Data Mode

This mode is used to generate cryptographic checksums on data (MAC) by means of cipher block chaining encryption algorithm (CBC-MAC algorithm for example).

After each end of encryption/decryption, the output data are available either on the output data registers for Manual and Auto mode or at the address specified in the receive buffer pointer for DMA mode (See [Table 31-4 "Last Output Data Mode Behavior versus Start Modes"](#)).

The Last Output Data bit (LOD) in the AES Mode Register (AES\_MR) allows retrieval of only the last data of several encryption/decryption processes.

Therefore, there is no need to define a read buffer in DMA mode.

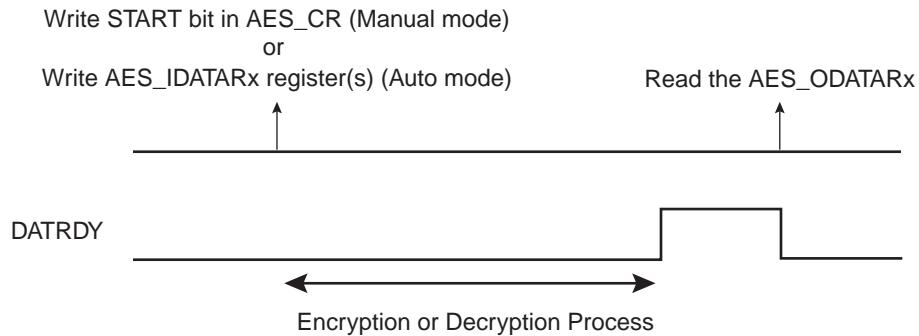
This data are only available on the Output Data Registers (AES\_ODATARx).

### 31.4.6 Manual and Auto Modes

#### 31.4.6.1 If LOD = 0

The DATRDY flag is cleared when at least one of the Output Data Registers is read (See [Figure 31-1](#)).

**Figure 31-1. Manual and Auto Modes with LOD = 0**

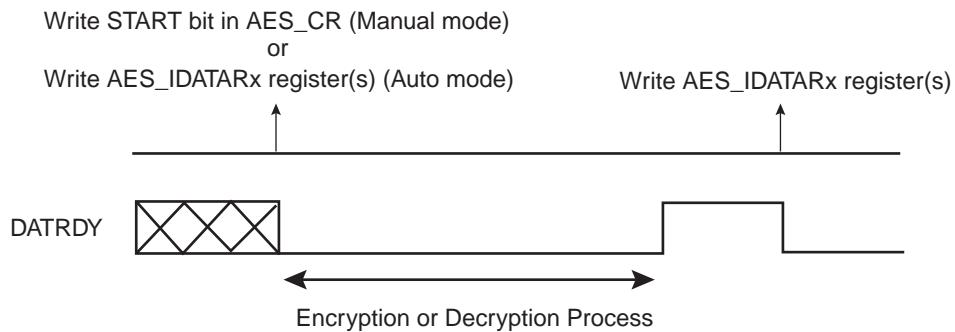


If the user does not want to read the output data registers between each encryption/decryption, the DATRDY flag will not be cleared. If the DATRDY flag is not cleared, the user cannot know the end of the following encryptions/decryptions.

#### 31.4.6.2 If LOD = 1

The DATRDY flag is cleared when at least one Input Data Register is written, so before the start of a new transfer (See [Figure 31-2](#)). No more Output Data Register reads are necessary between consecutive encryptions/decryptions.

**Figure 31-2. Manual and Auto Modes with LOD = 1**



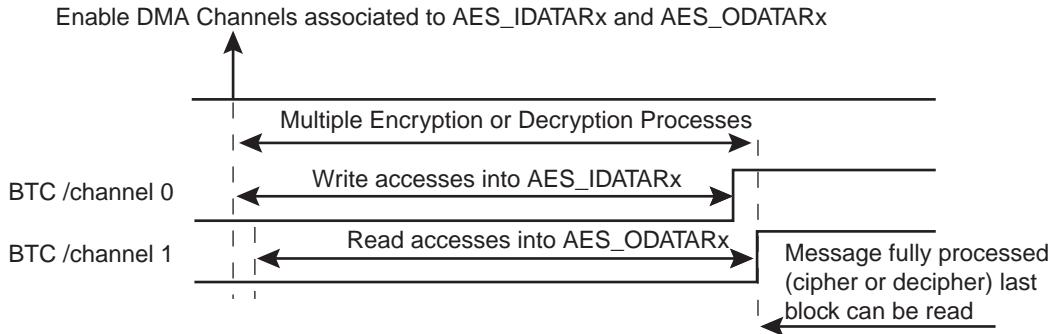
### 31.4.7 DMA Mode

#### 31.4.7.1 If LOD = 0

This mode may be used for all AES operating modes except CBC-MAC where LOD=1 mode is recommended.

The end of the encryption/decryption is notified by the end of DMA transfer associated to AES\_ODATARx registers (see [Figure 31-3](#)). 2 channels DMA are required, 1 for writing message blocks to AES\_IDATARx registers and the other one to get back the processed from AES\_ODATARx registers.

**Figure 31-3. DMA transfer with LOD = 0**



#### 31.4.7.2 If LOD = 1

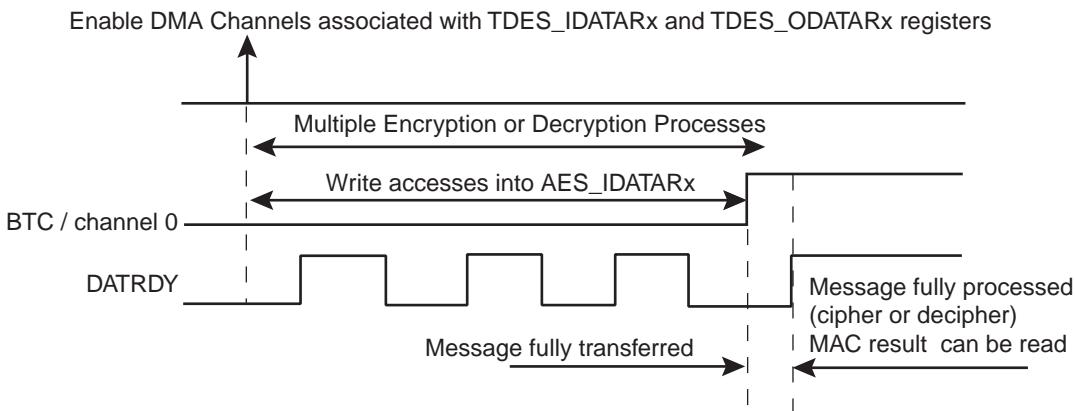
This mode is recommended to process AES CBC-MAC operating mode.

The user must first wait for the DMA flag (BTC = Buffer Transfer Complete) to rise, then for DATRDY to ensure that the encryption/decryption is completed (see [Figure 31-4](#)).

In this case, no receive buffers are required.

The output data are only available on the Output Data Registers (AES\_ODATARx).

**Figure 31-4. DMA transfer with LOD = 1**



[Table 31-4](#) summarizes the different cases.

**Table 31-4. Last Output Data Mode Behavior versus Start Modes**

	Manual and Auto Modes		DMA Transfer	
	LOD = 0	LOD = 1	LOD = 0	LOD = 1
DATRDY Flag Clearing Condition <sup>(1)</sup>	At least one Output Data Register must be read	At least one Input Data Register must be written	Not used	Managed by the DMA
Encrypted/Decrypted Data Result Location	In the Output Data Registers	In the Output Data Registers	Not available	In the Output Data Registers
End of Encryption/Decryption	DATRDY	DATRDY	2 DMA flags (BTC[n/m])	DMA flag (BTC[n]) then AES DATRDY

Note: 1. Depending on the mode, there are other ways of clearing the DATRDY flag. See: [Section 31.6.6 "AES Interrupt Status Register"](#).

**Warning:** In DMA mode, reading to the Output Data registers before the last data transfer may lead to unpredictable results.

## 31.5 Security Features

### 31.5.1 Unspecified Register Access Detection

When an unspecified register access occurs, the URAD bit in the Interrupt Status Register (AES\_ISR) raises. Its source is then reported in the Unspecified Register Access Type field (URAT). Only the last unspecified register access is available through the URAT field.

Several kinds of unspecified register accesses can occur:

- Input Data Register written during the data processing when SMOD=IDATAR0\_START
- Output Data Register read during data processing
- Mode Register written during data processing
- Output Data Register read during sub-keys generation
- Mode Register written during sub-keys generation
- Write-only register read access

The URAD bit and the URAT field can only be reset by the SWRST bit in the AES\_CR control register.

## 31.6 Advanced Encryption Standard (AES) User Interface

Table 31-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	AES_CR	Write-only	—
0x04	Mode Register	AES_MR	Read-write	0x0
0x08-0x0C	Reserved	—	—	—
0x10	Interrupt Enable Register	AES_IER	Write-only	—
0x14	Interrupt Disable Register	AES_IDR	Write-only	—
0x18	Interrupt Mask Register	AES_IMR	Read-only	0x0
0x1C	Interrupt Status Register	AES_ISR	Read-only	0x0
0x20	Key Word Register 0	AES_KEYWR0	Write-only	—
0x24	Key Word Register 1	AES_KEYWR1	Write-only	—
0x28	Key Word Register 2	AES_KEYWR2	Write-only	—
0x2C	Key Word Register 3	AES_KEYWR3	Write-only	—
0x30	Key Word Register 4	AES_KEYWR4	Write-only	—
0x34	Key Word Register 5	AES_KEYWR5	Write-only	—
0x38	Key Word Register 6	AES_KEYWR6	Write-only	—
0x3C	Key Word Register 7	AES_KEYWR7	Write-only	—
0x40	Input Data Register 0	AES_IDATAR0	Write-only	—
0x44	Input Data Register 1	AES_IDATAR1	Write-only	—
0x48	Input Data Register 2	AES_IDATAR2	Write-only	—
0x4C	Input Data Register 3	AES_IDATAR3	Write-only	—
0x50	Output Data Register 0	AES_ODATAR0	Read-only	0x0
0x54	Output Data Register 1	AES_ODATAR1	Read-only	0x0
0x58	Output Data Register 2	AES_ODATAR2	Read-only	0x0
0x5C	Output Data Register 3	AES_ODATAR3	Read-only	0x0
0x60	Initialization Vector Register 0	AES_IVR0	Write-only	—
0x64	Initialization Vector Register 1	AES_IVR1	Write-only	—
0x68	Initialization Vector Register 2	AES_IVR2	Write-only	—
0x6C	Initialization Vector Register 3	AES_IVR3	Write-only	—
0x70 - 0xFC	Reserved	—	—	—

### 31.6.1 AES Control Register

**Name:** AES\_CR

**Address:** 0x40004000

**Access:** Write-only

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

- **START: Start Processing**

0: No effect

1: Starts manual encryption/decryption process.

- **SWRST: Software Reset**

0: No effect.

1: Resets the AES. A software triggered hardware reset of the AES interface is performed.

### 31.6.2 AES Mode Register

**Name:** AES\_MR  
**Address:** 0x40004004  
**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
CKEY				—	CFBS		
15	14	13	12	11	10	9	8
LOD	OPMOD			KEYSIZE		SMOD	
7	6	5	4	3	2	1	0
PROCDLY				DUALBUFF	—	—	CIPHER

- **CIPHER: Processing Mode**

0: Decrypts data.

1: Encrypts data.

- **DUALBUFF: Dual Input BUFFer**

Value	Name	Description
0x0	INACTIVE	AES_IDATARx cannot be written during processing of previous block.
0x1	ACTIVE	AES_IDATARx can be written during processing of previous block when SMOD = 0x2. It speeds up the overall runtime of large files.

- **PROCDLY: Processing Delay**

$$\text{Processing Time} = 12 \times (\text{PROCDLY} + 1)$$

The Processing Time represents the number of clock cycles that the AES needs in order to perform one encryption/decryption.

Note: The best performance is achieved with PROCDLY equal to 0.

- **SMOD: Start Mode**

Value	Name	Description
0x0	MANUAL_START	Manual Mode
0x1	AUTO_START	Auto Mode
0x2	IDATAR0_START	AES_IDATAR0 access only Auto Mode

Values which are not listed in the table must be considered as “reserved”.

If a DMA transfer is used, 0x2 must be configured. Refer to [Section 31.4.4 "DMA Mode"](#) for more details.

- **KEYSIZE: Key Size**

Value	Name	Description
0x0	AES128	AES Key Size is 128 bits
0x1	AES192	AES Key Size is 192 bits
0x2	AES256	AES Key Size is 256 bits

Values which are not listed in the table must be considered as “reserved”.

- **OPMOD: Operation Mode**

Value	Name	Description
0x0	ECB	ECB: Electronic Code Book mode
0x1	CBC	CBC: Cipher Block Chaining mode
0x2	OFB	OFB: Output Feedback mode
0x3	CFB	CFB: Cipher Feedback mode
0x4	CTR	CTR: Counter mode (16-bit internal counter)

Values which are not listed in the table must be considered as “reserved”.

For CBC-MAC operating mode, please set OPMOD to CBC and LOD to 1.

- **LOD: Last Output Data Mode**

0: No effect.

After each end of encryption/decryption, the output data will be available either on the output data registers (Manual and Auto modes) .

In Manual and Auto modes, the DATRDY flag is cleared when at least one of the Output Data registers is read.

1: The DATRDY flag is cleared when at least one of the Input Data Registers is written.

No more Output Data Register reads is necessary between consecutive encryptions/decryptions (see “[Last Output Data Mode](#)” on page 584).

Warning: In DMA mode, reading to the Output Data registers before the last data encryption/decryption process may lead to unpredictable results.

- **CFBS: Cipher Feedback Data Size**

Value	Name	Description
0x0	SIZE_128BIT	128-bit
0x1	SIZE_64BIT	64-bit
0x2	SIZE_32BIT	32-bit
0x3	SIZE_16BIT	16-bit
0x4	SIZE_8BIT	8-bit

Values which are not listed in table must be considered as “reserved”.

- **CKEY: Key**

Value	Name	Description
0xE	PASSWD	This field must be written with 0xE the first time that AES_MR is programmed. For subsequent programming of the AES_MR register, any value can be written, including that of 0xE. Always reads as 0.

### 31.6.3 AES Interrupt Enable Register

**Name:** AES\_IER  
**Address:** 0x40004010  
**Access:** Write-only

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

- **DATRDY: Data Ready Interrupt Enable**
- **URAD: Unspecified Register Access Detection Interrupt Enable**

0: No effect.

1: Enables the corresponding interrupt.

### 31.6.4 AES Interrupt Disable Register

**Name:** AES\_IDR  
**Address:** 0x40004014  
**Access:** Write-only

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

- **DATRDY: Data Ready Interrupt Disable**
- **URAD: Unspecified Register Access Detection Interrupt Disable**

0: No effect.

1: Disables the corresponding interrupt.

### 31.6.5 AES Interrupt Mask Register

**Name:** AES\_IMR

**Address:** 0x40004018

**Access:** Read-only

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

- **DATRDY: Data Ready Interrupt Mask**

- **URAD: Unspecified Register Access Detection Interrupt Mask**

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

### 31.6.6 AES Interrupt Status Register

**Name:** AES\_ISR  
**Address:** 0x4000401C  
**Access:** Read-only

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

- **DATRDY: Data Ready**

0: Output data not valid.

1: Encryption or decryption process is completed.

DATRDY is cleared when a Manual encryption/decryption occurs (START bit in AES\_CR) or when a software triggered hardware reset of the AES interface is performed (SWRST bit in AES\_CR).

**LOD = 0 (AES\_MR):**

In Manual and Auto mode, the DATRDY flag can also be cleared when at least one of the Output Data Registers is read.

In DMA mode, DATRDY is set and cleared automatically.

**LOD = 1 (AES\_MR):**

In Manual and Auto mode, the DATRDY flag can also be cleared when at least one of the Input Data Registers is written.

In DMA mode, DATRDY is set and cleared automatically.

- **URAD: Unspecified Register Access Detection Status**

0: No unspecified register access has been detected since the last SWRST.

1: At least one unspecified register access has been detected since the last SWRST.

URAD bit is reset only by the SWRST bit in the AES\_CR control register.

- **URAT: Unspecified Register Access:**

Value	Name	Description
0x0	IDR_WR_PROCESSING	Input Data Register written during the data processing when SMOD=0x2 mode.
0x1	ODR_RD_PROCESSING	Output Data Register read during the data processing.
0x2	MR_WR_PROCESSING	Mode Register written during the data processing.
0x3	ODR_RD_SUBKGEN	Output Data Register read during the sub-keys generation.
0x4	MR_WR_SUBKGEN	Mode Register written during the sub-keys generation.
0x5	WOR_RD_ACCESS	Write-only register read access.

Only the last Unspecified Register Access Type is available through the URAT field.

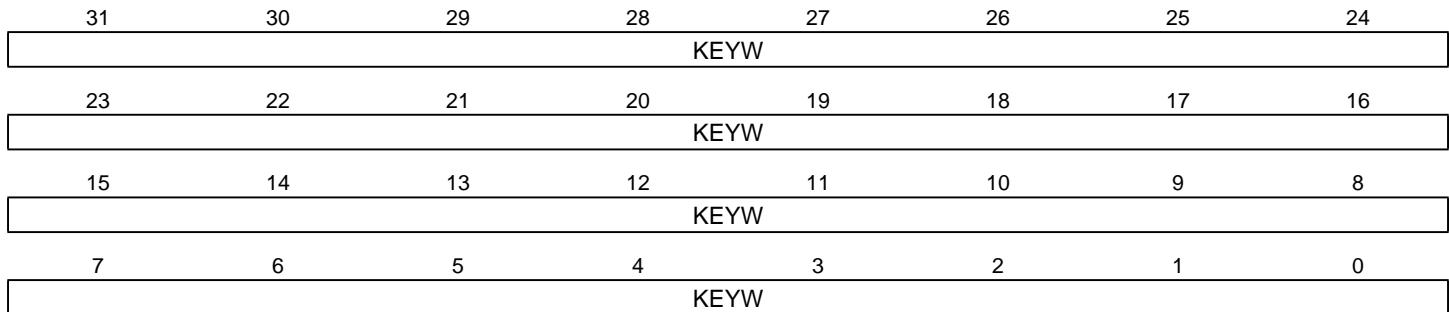
URAT field is reset only by the SWRST bit in the AES\_CR control register.

### 31.6.7 AES Key Word Register x

**Name:** AES\_KEYWRx

**Address:** 0x40004020

**Access:** Write-only



- **KEYW: Key Word**

The four/six/eight 32-bit Key Word registers set the 128-bit/192-bit/256-bit cryptographic key used for encryption/decryption.

AES\_KEYWR0 corresponds to the first word of the key and respectively AES\_KEYWR3/AES\_KEYWR5/AES\_KEYWR7 to the last one.

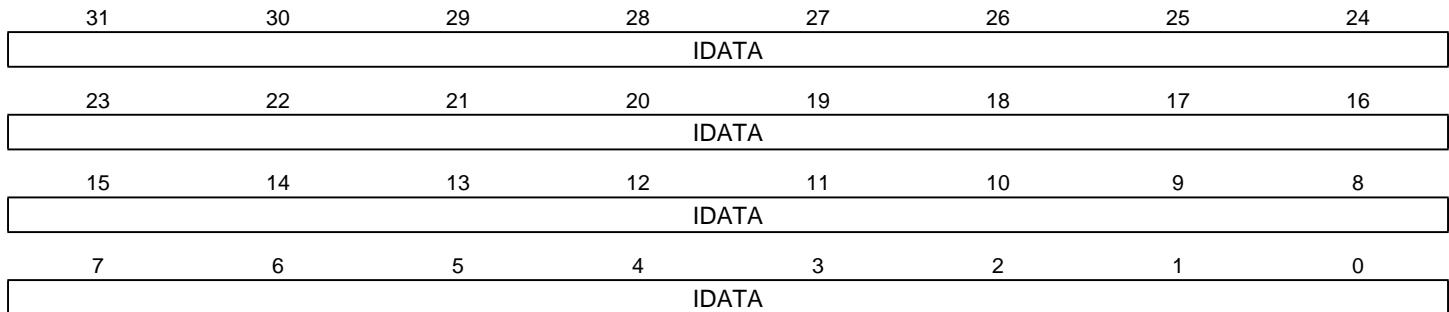
These registers are write-only to prevent the key from being read by another application.

### 31.6.8 AES Input Data Register x

**Name:** AES\_IDATARx

**Address:** 0x40004040

**Access:** Write-only



- **IDATA: Input Data Word**

The four 32-bit Input Data registers set the 128-bit data block used for encryption/decryption.

AES\_IDATAR0 corresponds to the first word of the data to be encrypted/decrypted, and AES\_IDATAR3 to the last one.

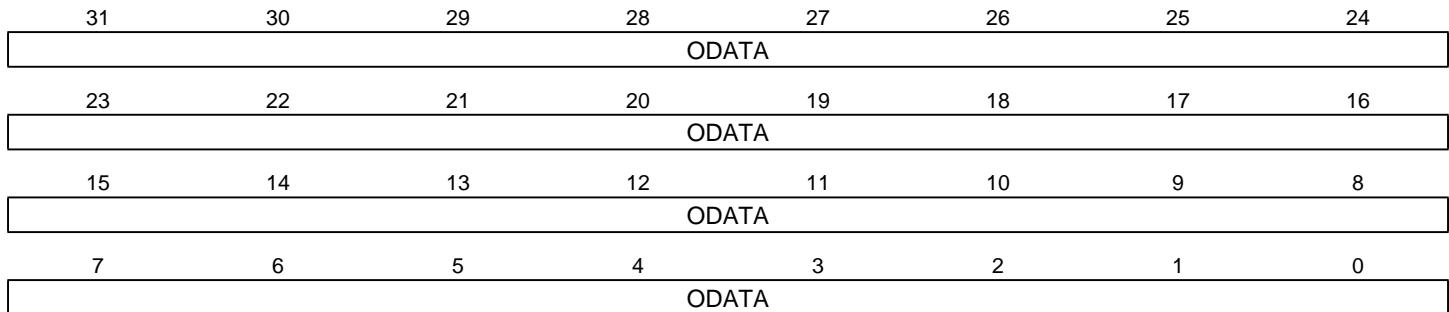
These registers are write-only to prevent the input data from being read by another application.

### 31.6.9 AES Output Data Register x

**Name:** AES\_ODATARx

**Address:** 0x40004050

**Access:** Read-only



- **ODATA: Output Data**

The four 32-bit Output Data registers contain the 128-bit data block that has been encrypted/decrypted.

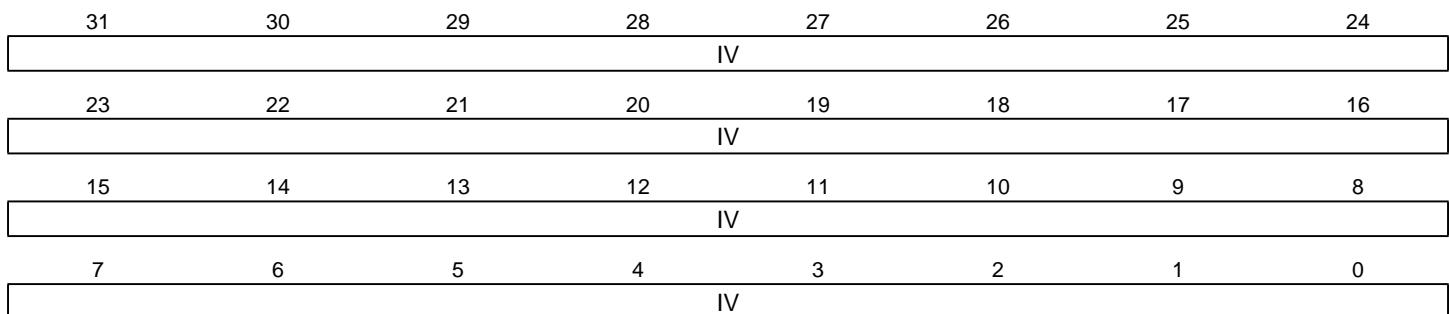
AES\_ODATAR0 corresponds to the first word, AES\_ODATAR3 to the last one.

### 31.6.10 AES Initialization Vector Register x

**Name:** AES\_IVRx

**Address:** 0x40004060

**Access:** Write-only



- **IV: Initialization Vector**

The four 32-bit Initialization Vector registers set the 128-bit Initialization Vector data block that is used by some modes of operation as an additional initial input.

AES\_IVR0 corresponds to the first word of the Initialization Vector, AES\_IVR3 to the last one.

These registers are write-only to prevent the Initialization Vector from being read by another application.

For CBC, OFB and CFB modes, the Initialization Vector corresponds to the initialization vector.

For CTR mode, it corresponds to the counter value.

Note: These registers are not used in ECB mode and must not be written.

## 32. Controller Area Network (CAN) Programmer Datasheet

### 32.1 Description

The CAN controller provides all the features required to implement the serial communication protocol CAN defined by Robert Bosch GmbH, the CAN specification as referred to by ISO/11898A (2.0 Part A and 2.0 Part B) for high speeds and ISO/11519-2 for low speeds. The CAN Controller is able to handle all types of frames (Data, Remote, Error and Overload) and achieves a bitrate of 1 Mbps.

CAN controller accesses are made through configuration registers. 8 independent message objects (mailboxes) are implemented.

Any mailbox can be programmed as a reception buffer block (even non-consecutive buffers). For the reception of defined messages, one or several message objects can be masked without participating in the buffer feature. An interrupt is generated when the buffer is full. According to the mailbox configuration, the first message received can be locked in the CAN controller registers until the application acknowledges it, or this message can be discarded by new received messages.

Any mailbox can be programmed for transmission. Several transmission mailboxes can be enabled in the same time. A priority can be defined for each mailbox independently.

An internal 16-bit timer is used to stamp each received and sent message. This timer starts counting as soon as the CAN controller is enabled. This counter can be reset by the application or automatically after a reception in the last mailbox in Time Triggered Mode.

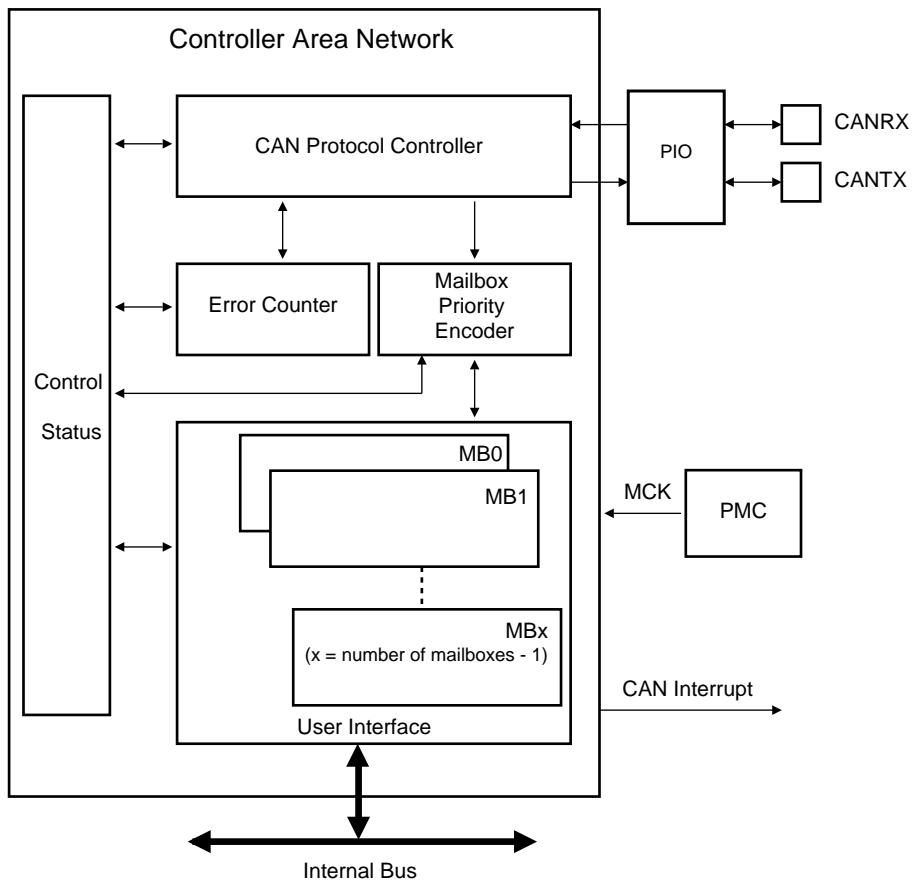
The CAN controller offers optimized features to support the Time Triggered Communication (TTC) protocol.

### 32.2 Embedded Characteristics

- Fully Compliant with CAN 2.0 Part A and 2.0 Part B
- Bit Rates up to 1Mbit/s
- 8 Object Oriented Mailboxes with the Following Properties:
  - CAN Specification 2.0 Part A or 2.0 Part B Programmable for Each Message
  - Object Configurable in Receive (with Overwrite or Not) or Transmit Modes
  - Independent 29-bit Identifier and Mask Defined for Each Mailbox
  - 32-bit Access to Data Registers for Each Mailbox Data Object
  - Uses a 16-bit Timestamp on Receive and Transmit Messages
  - Hardware Concatenation of ID Masked Bitfields To Speed Up Family ID Processing
- 16-bit Internal Timer for Timestamping and Network Synchronization
- Programmable Reception Buffer Length up to 8 Mailbox Objects
- Priority Management between Transmission Mailboxes
- Autobaud and Listening Mode
- Low Power Mode and Programmable Wake-up on Bus Activity or by the Application
- Data, Remote, Error and Overload Frame Handling
- Write Protected Registers

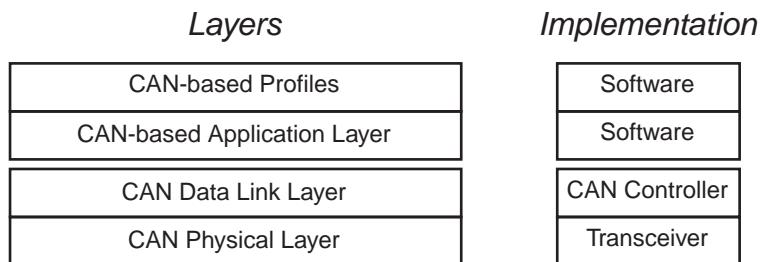
### 32.3 Block Diagram

Figure 32-1. CAN Block Diagram



### 32.4 Application Block Diagram

Figure 32-2. Application Block Diagram



## 32.5 I/O Lines Description

Table 32-1. I/O Lines Description

Name	Description	Type
CANRX	CAN Receive Serial Data	Input
CANTX	CAN Transmit Serial Data	Output

## 32.6 Product Dependencies

### 32.6.1 I/O Lines

The pins used for interfacing the CAN may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired CAN pins to their peripheral function. If I/O lines of the CAN are not used by the application, they can be used for other purposes by the PIO Controller.

Table 32-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
CAN0	CANRX0	PB3	A
CAN0	CANTX0	PB2	A
CAN1	CANRX1	PC12	C
CAN1	CANTX1	PC15	C

### 32.6.2 Power Management

The programmer must first enable the CAN clock in the Power Management Controller (PMC) before using the CAN.

A Low-power Mode is defined for the CAN controller. If the application does not require CAN operations, the CAN clock can be stopped when not needed and be restarted later. Before stopping the clock, the CAN Controller must be in Low-power Mode to complete the current transfer. After restarting the clock, the application must disable the Low-power Mode of the CAN controller.

### 32.6.3 Interrupt

The CAN interrupt line is connected on one of the internal sources of the Advanced Interrupt Controller. Using the CAN interrupt requires the AIC to be programmed first. Note that it is not recommended to use the CAN interrupt line in edge-sensitive mode.

Table 32-3. Peripheral IDs

Instance	ID
CAN0	37
CAN1	38

## 32.7 CAN Controller Features

### 32.7.1 CAN Protocol Overview

The Controller Area Network (CAN) is a multi-master serial communication protocol that efficiently supports real-time control with a very high level of security with bit rates up to 1 Mbit/s.

The CAN protocol supports four different frame types:

- Data frames: They carry data from a transmitter node to the receiver nodes. The overall maximum data frame length is 108 bits for a standard frame and 128 bits for an extended frame.
- Remote frames: A destination node can request data from the source by sending a remote frame with an identifier that matches the identifier of the required data frame. The appropriate data source node then sends a data frame as a response to this node request.
- Error frames: An error frame is generated by any node that detects a bus error.
- Overload frames: They provide an extra delay between the preceding and the successive data frames or remote frames.

The Atmel CAN controller provides the CPU with full functionality of the CAN protocol V2.0 Part A and V2.0 Part B. It minimizes the CPU load in communication overhead. The Data Link Layer and part of the physical layer are automatically handled by the CAN controller itself.

The CPU reads or writes data or messages via the CAN controller mailboxes. An identifier is assigned to each mailbox. The CAN controller encapsulates or decodes data messages to build or to decode bus data frames. Remote frames, error frames and overload frames are automatically handled by the CAN controller under supervision of the software application.

### 32.7.2 Mailbox Organization

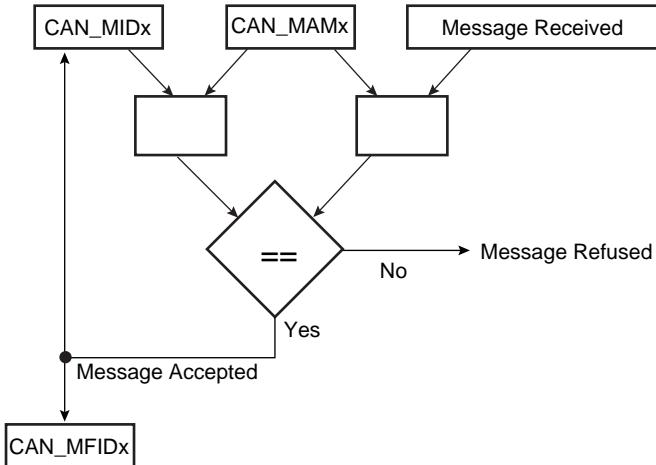
The CAN module has 8 buffers, also called channels or mailboxes. An identifier that corresponds to the CAN identifier is defined for each active mailbox. Message identifiers can match the standard frame identifier or the extended frame identifier. This identifier is defined for the first time during the CAN initialization, but can be dynamically reconfigured later so that the mailbox can handle a new message family. Several mailboxes can be configured with the same ID.

Each mailbox can be configured in receive or in transmit mode independently. The mailbox object type is defined in the MOT field of the CAN\_MMRx register.

#### 32.7.2.1 Message Acceptance Procedure

If the MIDE field in the CAN\_MIDx register is set, the mailbox can handle the extended format identifier; otherwise, the mailbox handles the standard format identifier. Once a new message is received, its ID is masked with the CAN\_MAMx value and compared with the CAN\_MIDx value. If accepted, the message ID is copied to the CAN\_MIDx register.

### **Figure 32-3. Message Acceptance Procedure**



If a mailbox is dedicated to receiving several messages (a family of messages) with different IDs, the acceptance mask defined in the CAN\_MAMx register must mask the variable part of the ID family. Once a message is received, the application must decode the masked bits in the CAN\_MIDx. To speed up the decoding, masked bits are grouped in the family ID register (CAN\_MFIDx).

For example, if the following message IDs are handled by the same mailbox:

ID0	1010001001000100100000100	0	11	00b
ID1	1010001001000100100000100	0	11	01b
ID2	1010001001000100100000100	0	11	10b
ID3	1010001001000100100000100	0	11	11b
ID4	1010001001000100100000100	1	11	00b
ID5	1010001001000100100000100	1	11	01b
ID6	1010001001000100100000100	1	11	10b
ID7	1010001001000100100000100	1	11	11b

The CAN\_MIDx and CAN\_MAMx of Mailbox x must be initialized to the corresponding values:

CAN\_MIDx = 001 101000100100010010000100 x 11 xx**b**  
CAN\_MAMx = 001 11111111111111111111111111111111 0 11 00**b**

If Mailbox x receives a message with ID6, then CAN\_MIDx and CAN\_MFIDx are set:

If the application associates a handler for each message ID, it may define an array of pointers to functions:

```
void (*pHandler[8])(void);
```

When a message is received, the corresponding handler can be invoked using CAN\_MFIDx register and there is no need to check masked bits:

```
unsigned int MFID0_register;
MFID0_register = Get_CAN_MFID0_Register();
// Get_CAN_MFID0_Register() returns the value of the CAN_MFID0 register
pHandler[MFID0_register]();
```

### 32.7.2.2 Receive Mailbox

When the CAN module receives a message, it looks for the first available mailbox with the lowest number and compares the received message ID with the mailbox ID. If such a mailbox is found, then the message is stored in its data registers. Depending on the configuration, the mailbox is disabled as long as the message has not been acknowledged by the application (Receive only), or, if new messages with the same ID are received, then they overwrite the previous ones (Receive with overwrite).

It is also possible to configure a mailbox in Consumer Mode. In this mode, after each transfer request, a remote frame is automatically sent. The first answer received is stored in the corresponding mailbox data registers.

Several mailboxes can be chained to receive a buffer. They must be configured with the same ID in Receive Mode, except for the last one, which can be configured in Receive with Overwrite Mode. The last mailbox can be used to detect a buffer overflow.

**Table 32-4. Receive Mailbox Overview**

Mailbox Object Type	Description
Receive	The first message received is stored in mailbox data registers. Data remain available until the next transfer request.
Receive with overwrite	The last message received is stored in mailbox data register. The next message always overwrites the previous one. The application has to check whether a new message has not overwritten the current one while reading the data registers.
Consumer	A remote frame is sent by the mailbox. The answer received is stored in mailbox data register. This extends Receive mailbox features. Data remain available until the next transfer request.

### 32.7.2.3 Transmit Mailbox

When transmitting a message, the message length and data are written to the transmit mailbox with the correct identifier. For each transmit mailbox, a priority is assigned. The controller automatically sends the message with the highest priority first (set with the field PRIOR in CAN\_MMRx register).

It is also possible to configure a mailbox in Producer Mode. In this mode, when a remote frame is received, the mailbox data are sent automatically. By enabling this mode, a producer can be done using only one mailbox instead of two: one to detect the remote frame and one to send the answer.

**Table 32-5. Transmit Mailbox Overview**

Mailbox Object Type	Description
Transmit	The message stored in the mailbox data registers will try to win the bus arbitration immediately or later according to or not the Time Management Unit configuration (see <a href="#">Section 32.7.3</a> ). The application is notified that the message has been sent or aborted.
Producer	The message prepared in the mailbox data registers will be sent after receiving the next remote frame. This extends transmit mailbox features.

### 32.7.3 Time Management Unit

The CAN Controller integrates a free-running 16-bit internal timer. The counter is driven by the bit clock of the CAN bus line. It is enabled when the CAN controller is enabled (CANEN set in the CAN\_MR register). It is automatically cleared in the following cases:

- After a reset
- When the CAN controller is in Low-power Mode is enabled (LPM bit set in the CAN\_MR and SLEEP bit set in the CAN\_SR)
- After a reset of the CAN controller (CANEN bit in the CAN\_MR register)
- In Time-triggered Mode, when a message is accepted by the last mailbox (rising edge of the MRDY signal in the CAN\_MSR<sub>last\_mailbox\_number</sub> register).

The application can also reset the internal timer by setting TIMRST in the CAN\_TCR register. The current value of the internal timer is always accessible by reading the CAN\_TIM register.

When the timer rolls-over from FFFFh to 0000h, TOVF (Timer Overflow) signal in the CAN\_SR register is set. TOVF bit in the CAN\_SR register is cleared by reading the CAN\_SR register. Depending on the corresponding interrupt mask in the CAN\_IMR register, an interrupt is generated while TOVF is set.

In a CAN network, some CAN devices may have a larger counter. In this case, the application can also decide to freeze the internal counter when the timer reaches FFFFh and to wait for a restart condition from another device. This feature is enabled by setting TIMFRZ in the CAN\_MR register. The CAN\_TIM register is frozen to the FFFFh value. A clear condition described above restarts the timer. A timer overflow (TOVF) interrupt is triggered.

To monitor the CAN bus activity, the CAN\_TIM register is copied to the CAN\_TIMESTP register after each start of frame or end of frame and a TSTP interrupt is triggered. If TEOF bit in the CAN\_MR register is set, the value is captured at each End Of Frame, else it is captured at each Start Of Frame. Depending on the corresponding mask in the CAN\_IMR register, an interrupt is generated while TSTP is set in the CAN\_SR. TSTP bit is cleared by reading the CAN\_SR register.

The time management unit can operate in one of the two following modes:

- Timestamping mode: The value of the internal timer is captured at each Start Of Frame or each End Of Frame
- Time Triggered mode: A mailbox transfer operation is triggered when the internal timer reaches the mailbox trigger.

Timestamping Mode is enabled by clearing TTM field in the CAN\_MR register. Time Triggered Mode is enabled by setting TTM field in the CAN\_MR register.

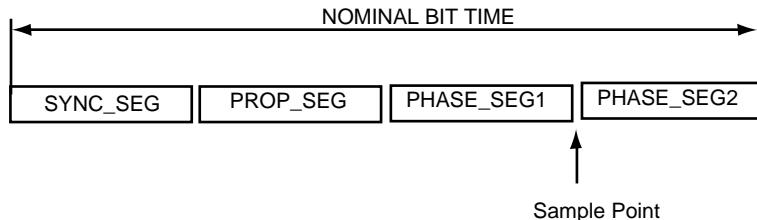
## 32.7.4 CAN 2.0 Standard Features

### 32.7.4.1 CAN Bit Timing Configuration

All controllers on a CAN bus must have the same bit rate and bit length. At different clock frequencies of the individual controllers, the bit rate has to be adjusted by the time segments.

The CAN protocol specification partitions the nominal bit time into four different segments:

**Figure 32-4. Partition of the CAN Bit Time**



- **TIME QUANTUM**

The TIME QUANTUM (TQ) is a fixed unit of time derived from the MCK period. The total number of TIME QUANTA in a bit time is programmable from 8 to 25.

**SYNC SEG:** SYNChronization Segment.

This part of the bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment. It is 1 TQ long.

- **PROP SEG:** PROPaigation Segment.

This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal's propagation time on the bus line, the input comparator delay, and the output driver delay. It is programmable to be 1,2,..., 8 TQ long.

This parameter is defined in the PROPAG field of the "["CAN Baudrate Register"](#)".

- **PHASE SEG1, PHASE SEG2:** PHASE Segment 1 and 2.

The Phase-Buffer-Segments are used to compensate for edge phase errors. These segments can be lengthened (PHASE SEG1) or shortened (PHASE SEG2) by resynchronization.

Phase Segment 1 is programmable to be 1,2,..., 8 TQ long.

Phase Segment 2 length has to be at least as long as the Information Processing Time (IPT) and may not be more than the length of Phase Segment 1.

These parameters are defined in the PHASE1 and PHASE2 fields of the "["CAN Baudrate Register"](#)".

- **INFORMATION PROCESSING TIME:**

The Information Processing Time (IPT) is the time required for the logic to determine the bit level of a sampled bit. The IPT begins at the sample point, is measured in TQ and **is fixed at 2 TQ for the Atmel CAN**. Since Phase Segment 2 also begins at the sample point and is the last segment in the bit time, PHASE SEG2 shall not be less than the IPT.

- **SAMPLE POINT:**

The SAMPLE POINT is the point in time at which the bus level is read and interpreted as the value of that respective bit. Its location is at the end of PHASE\_SEG1.

- **SJW:** ReSynchronization Jump Width.

The ReSynchronization Jump Width defines the limit to the amount of lengthening or shortening of the Phase Segments. SJW is programmable to be the minimum of PHASE SEG1 and 4 TQ.

If the SMP field in the CAN\_BR register is set, then the incoming bit stream is sampled three times with a period of half a CAN clock period, centered on sample point.

In the CAN controller, the length of a bit on the CAN bus is determined by the parameters (BRP, PROPAG, PHASE1 and PHASE2).

$$t_{\text{BIT}} = t_{\text{CSC}} + t_{\text{PRS}} + t_{\text{PHS1}} + t_{\text{PHS2}}$$

The time quantum is calculated as follows:

$$t_{\text{CSC}} = (\text{BRP} + 1)/\text{MCK}$$

**Note:** The BRP field must be within the range [1, 0x7F], i.e., BRP = 0 is not authorized.

$$t_{\text{PRS}} = t_{\text{CSC}} \times (\text{PROPAG} + 1)$$

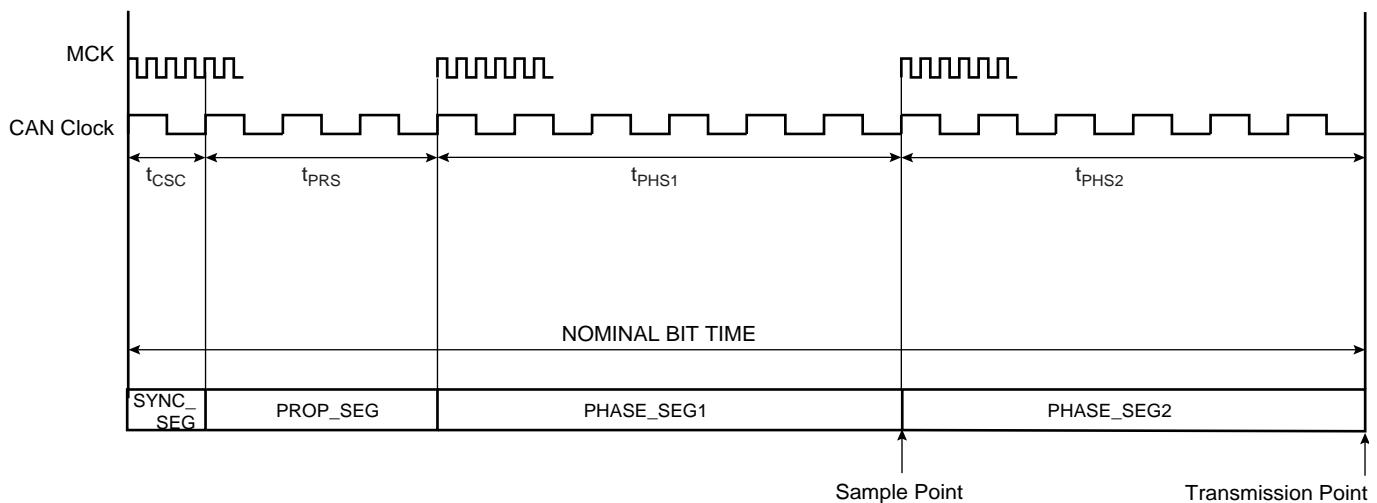
$$t_{\text{PHS1}} = t_{\text{CSC}} \times (\text{PHASE1} + 1)$$

$$t_{\text{PHS2}} = t_{\text{CSC}} \times (\text{PHASE2} + 1)$$

To compensate for phase shifts between clock oscillators of different controllers on the bus, the CAN controller must resynchronize on any relevant signal edge of the current transmission. The resynchronization shortens or lengthens the bit time so that the position of the sample point is shifted with regard to the detected edge. The resynchronization jump width (SJW) defines the maximum of time by which a bit period may be shortened or lengthened by resynchronization.

$$t_{\text{SJW}} = t_{\text{CSC}} \times (\text{SJW} + 1)$$

**Figure 32-5. CAN Bit Timing**



Example of bit timing determination for CAN baudrate of 500 Kbps:

MCK = 48 MHz  
 CAN baudrate= 500 Kbps => bit time= 2us  
 Delay of the bus driver: 50 ns  
 Delay of the receiver: 30ns  
 Delay of the bus line (20m): 110ns

The total number of time quanta in a bit time must be comprised between 8 and 25. If we fix the bit time to 16 time quanta:

$$\begin{aligned} T_{\text{CSC}} &= 1 \text{ time quanta} = \text{bit time} / 16 = 125 \text{ ns} \\ \Rightarrow \text{BRP} &= (T_{\text{CSC}} \times \text{MCK}) - 1 = 5 \end{aligned}$$

The propagation segment time is equal to twice the sum of the signal's propagation time on the bus line, the receiver delay and the output driver delay:

$$\begin{aligned} T_{\text{PRS}} &= 2 * (50+30+110) \text{ ns} = 380 \text{ ns} = 3 \text{ Tcsc} \\ \Rightarrow \text{PROPAG} &= T_{\text{PRS}}/T_{\text{CSC}} - 1 = 2 \end{aligned}$$

The remaining time for the two phase segments is:

$$T_{\text{PHS1}} + T_{\text{PHS2}} = \text{bit time} - T_{\text{CSC}} - T_{\text{PRS}} = (16 - 1 - 3)\text{Tcsc}$$

$T_{PHS1} + T_{PHS2} = 12 \text{ Tcsc}$

Because this number is even, we choose  $T_{PHS2} = T_{PHS1}$  (else we would choose  $T_{PHS2} = T_{PHS1} + \text{Tcsc}$ )  
 $T_{PHS1} = T_{PHS2} = (12/2) \text{ Tcsc} = 6 \text{ Tcsc}$   
 $\Rightarrow \text{PHASE1} = \text{PHASE2} = T_{PHS1}/\text{Tcsc} - 1 = 5$

The resynchronization jump width must be comprised between 1 Tcsc and the minimum of 4 Tcsc and  $T_{PHS1}$ . We choose its maximum value:

$T_{SJW} = \text{Min}(4 \text{ Tcsc}, T_{PHS1}) = 4 \text{ Tcsc}$   
 $\Rightarrow SJW = T_{SJW}/\text{Tcsc} - 1 = 3$

Finally: CAN\_BR = 0x00053255

### CAN Bus Synchronization

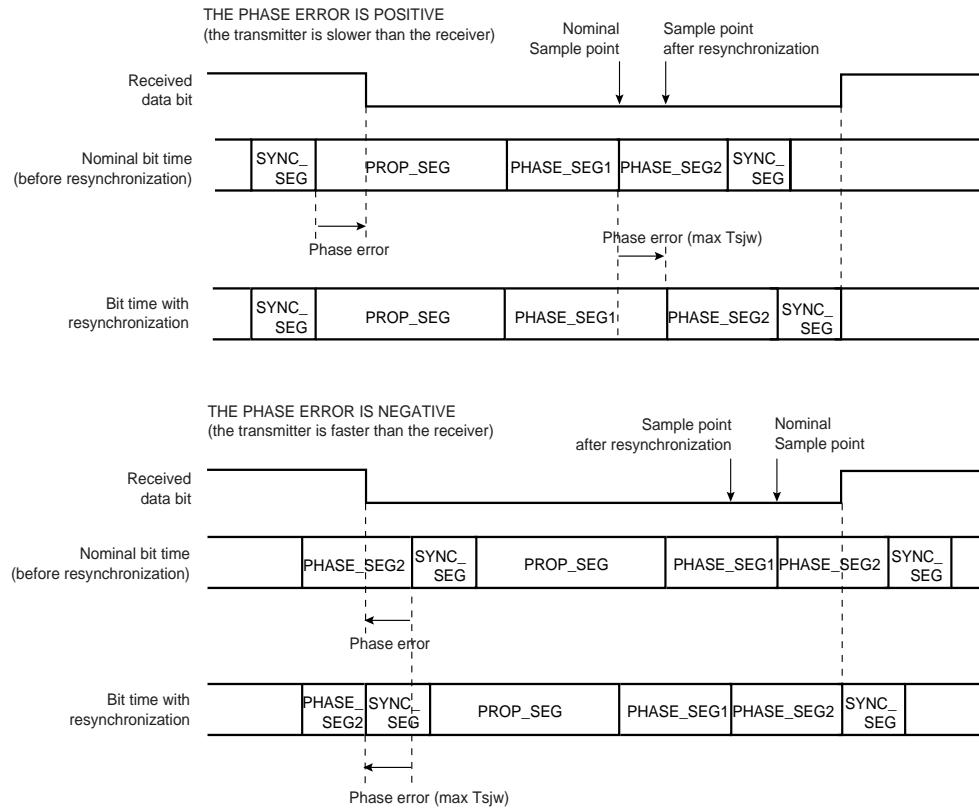
Two types of synchronization are distinguished: “hard synchronization” at the start of a frame and “resynchronization” inside a frame. After a hard synchronization, the bit time is restarted with the end of the SYNC\_SEG segment, regardless of the phase error. Resynchronization causes a reduction or increase in the bit time so that the position of the sample point is shifted with respect to the detected edge.

The effect of resynchronization is the same as that of hard synchronization when the magnitude of the phase error of the edge causing the resynchronization is less than or equal to the programmed value of the resynchronization jump width ( $t_{SJW}$ ).

When the magnitude of the phase error is larger than the resynchronization jump width and

- The phase error is positive, then PHASE\_SEG1 is lengthened by an amount equal to the resynchronization jump width.
- The phase error is negative, then PHASE\_SEG2 is shortened by an amount equal to the resynchronization jump width.

**Figure 32-6. CAN Resynchronization**



#### Autobaud Mode

The autobaud feature is enabled by setting the ABM field in the CAN\_MR register. In this mode, the CAN controller is only listening to the line without acknowledging the received messages. It can not send any message. The errors flags are updated. The bit timing can be adjusted until no error occurs (good configuration found). In this mode, the error counters are frozen. To go back to the standard mode, the ABM bit must be cleared in the CAN\_MR register.

#### 32.7.4.2 Error Detection

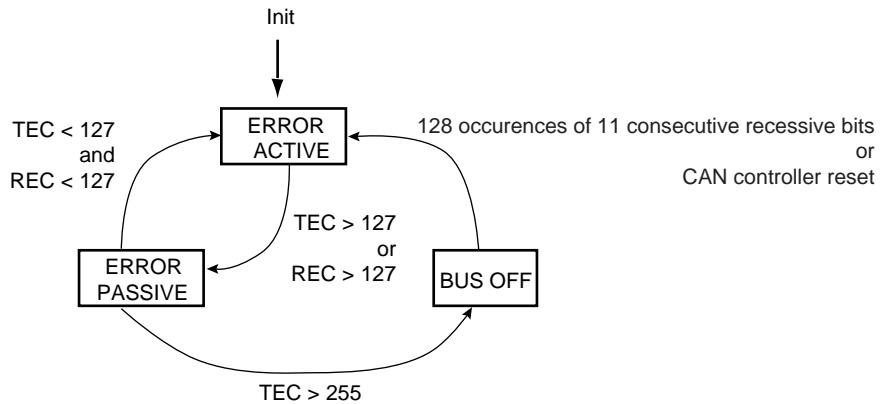
There are five different error types that are not mutually exclusive. Each error concerns only specific fields of the CAN data frame (refer to the Bosch CAN specification for their correspondence):

- CRC error (CERR bit in the CAN\_SR register): With the CRC, the transmitter calculates a checksum for the CRC bit sequence from the Start of Frame bit until the end of the Data Field. This CRC sequence is transmitted in the CRC field of the Data or Remote Frame.
- Bit-stuffing error (SERR bit in the CAN\_SR register): If a node detects a sixth consecutive equal bit level during the bit-stuffing area of a frame, it generates an Error Frame starting with the next bit-time.
- Bit error (BERR bit in CAN\_SR register): A bit error occurs if a transmitter sends a dominant bit but detects a recessive bit on the bus line, or if it sends a recessive bit but detects a dominant bit on the bus line. An error frame is generated and starts with the next bit time.
- Form Error (FERR bit in the CAN\_SR register): If a transmitter detects a dominant bit in one of the fix-formatted segments CRC Delimiter, ACK Delimiter or End of Frame, a form error has occurred and an error frame is generated.
- Acknowledgment error (AERR bit in the CAN\_SR register): The transmitter checks the Acknowledge Slot, which is transmitted by the transmitting node as a recessive bit, contains a dominant bit. If this is the case, at least one other node has received the frame correctly. If not, an Acknowledge Error has occurred and the transmitter will start in the next bit-time an Error Frame transmission.

#### Fault Confinement

To distinguish between temporary and permanent failures, every CAN controller has two error counters: REC (Receive Error Counter) and TEC (Transmit Error Counter). The two counters are incremented upon detected errors and are decremented upon correct transmissions or receptions, respectively. Depending on the counter values, the state of the node changes: the initial state of the CAN controller is Error Active, meaning that the controller can send Error Active flags. The controller changes to the Error Passive state if there is an accumulation of errors. If the CAN controller fails or if there is an extreme accumulation of errors, there is a state transition to Bus Off.

**Figure 32-7. Line Error Mode**



An error active unit takes part in bus communication and sends an active error frame when the CAN controller detects an error.

An error passive unit cannot send an active error frame. It takes part in bus communication, but when an error is detected, a passive error frame is sent. Also, after a transmission, an error passive unit waits before initiating further transmission.

A bus off unit is not allowed to have any influence on the bus.

For fault confinement, two errors counters (TEC and REC) are implemented. These counters are accessible via the CAN\_ECR register. The state of the CAN controller is automatically updated according to these counter values. If the CAN controller enters Error Active state, then the ERRA bit is set in the CAN\_SR register. The corresponding interrupt is pending while the interrupt is not masked in the CAN\_IMR register. If the CAN controller enters Error Passive Mode, then the ERRP bit is set in the CAN\_SR register and an interrupt remains pending while the ERRP bit is set in the CAN\_IMR register. If the CAN enters Bus Off Mode, then the BOFF bit is set in the CAN\_SR register. As for ERRP and ERRA, an interrupt is pending while the BOFF bit is set in the CAN\_IMR register.

When one of the error counters values exceeds 96, an increased error rate is indicated to the controller through the WARN bit in CAN\_SR register, but the node remains error active. The corresponding interrupt is pending while the interrupt is set in the CAN\_IMR register.

Refer to the Bosch CAN specification v2.0 for details on fault confinement.

#### Error Interrupt Handler

ERRA, WARN, ERRP and BOFF (CAN\_SR) store the key transitions of the CAN bus status as defined in [Figure 32-7 on page 610](#). The transitions depend on the TEC and REC (CAN\_ECR) values as described in [Section “Fault Confinement” on page 609](#).

These flags are latched to keep from triggering a spurious interrupt in case these bits are used as the source of an interrupt. Thus, these flags may not reflect the current status of the CAN bus.

The current CAN bus state can be determined by reading the TEC and REC fields of CAN\_ECR.

### 32.7.4.3 Overload

The overload frame is provided to request a delay of the next data or remote frame by the receiver node (“Request overload frame”) or to signal certain error conditions (“Reactive overload frame”) related to the intermission field respectively.

Reactive overload frames are transmitted after detection of the following error conditions:

- Detection of a dominant bit during the first two bits of the intermission field
- Detection of a dominant bit in the last bit of EOF by a receiver, or detection of a dominant bit by a receiver or a transmitter at the last bit of an error or overload frame delimiter

The CAN controller can generate a request overload frame automatically after each message sent to one of the CAN controller mailboxes. This feature is enabled by setting the OVL bit in the CAN\_MR register.

Reactive overload frames are automatically handled by the CAN controller even if the OVL bit in the CAN\_MR register is not set. An overload flag is generated in the same way as an error flag, but error counters do not increment.

## 32.7.5 Low-power Mode

In Low-power Mode, the CAN controller cannot send or receive messages. All mailboxes are inactive.

In Low-power Mode, the SLEEP signal in the CAN\_SR register is set; otherwise, the WAKEUP signal in the CAN\_SR register is set. These two fields are exclusive except after a CAN controller reset (WAKEUP and SLEEP are stuck at 0 after a reset). After power-up reset, the Low-power Mode is disabled and the WAKEUP bit is set in the CAN\_SR register only after detection of 11 consecutive recessive bits on the bus.

### 32.7.5.1 Enabling Low-power Mode

A software application can enable Low-power Mode by setting the LPM bit in the CAN\_MR global register. The CAN controller enters Low-power Mode once all pending transmit messages are sent.

When the CAN controller enters Low-power Mode, the SLEEP signal in the CAN\_SR register is set. Depending on the corresponding mask in the CAN\_IMR register, an interrupt is generated while SLEEP is set.

The SLEEP signal in the CAN\_SR register is automatically cleared once WAKEUP is set. The WAKEUP signal is automatically cleared once SLEEP is set.

Reception is disabled while the SLEEP signal is set to one in the CAN\_SR register. It is important to note that those messages with higher priority than the last message transmitted can be received between the LPM command and entry in Low-power Mode.

Once in Low-power Mode, the CAN controller clock can be switched off by programming the chip’s Power Management Controller (PMC). The CAN controller drains only the static current.

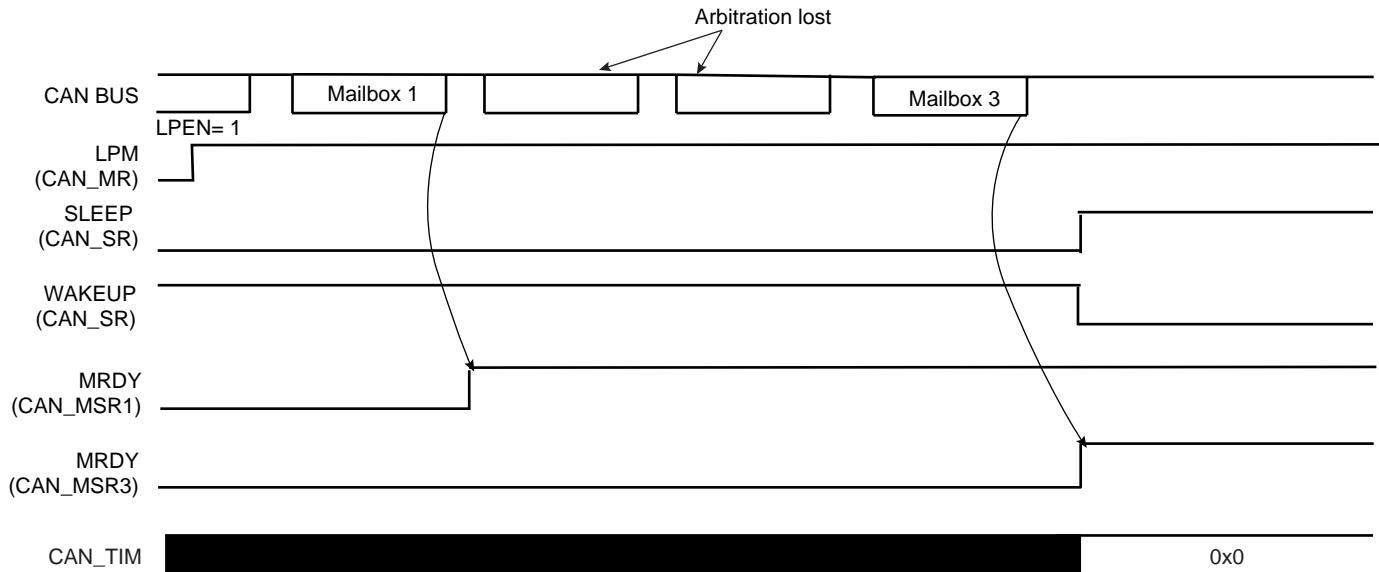
Error counters are disabled while the SLEEP signal is set to one.

Thus, to enter Low-power Mode, the software application must:

- Set LPM field in the CAN\_MR register
- Wait for SLEEP signal rising

Now the CAN Controller clock can be disabled. This is done by programming the Power Management Controller (PMC).

**Figure 32-8. Enabling Low-power Mode**



### 32.7.5.2 Disabling Low-power Mode

The CAN controller can be awake after detecting a CAN bus activity. Bus activity detection is done by an external module that may be embedded in the chip. When it is notified of a CAN bus activity, the software application disables Low-power Mode by programming the CAN controller.

To disable Low-power Mode, the software application must:

- Enable the CAN Controller clock. This is done by programming the Power Management Controller (PMC).
- Clear the LPM field in the CAN\_MR register

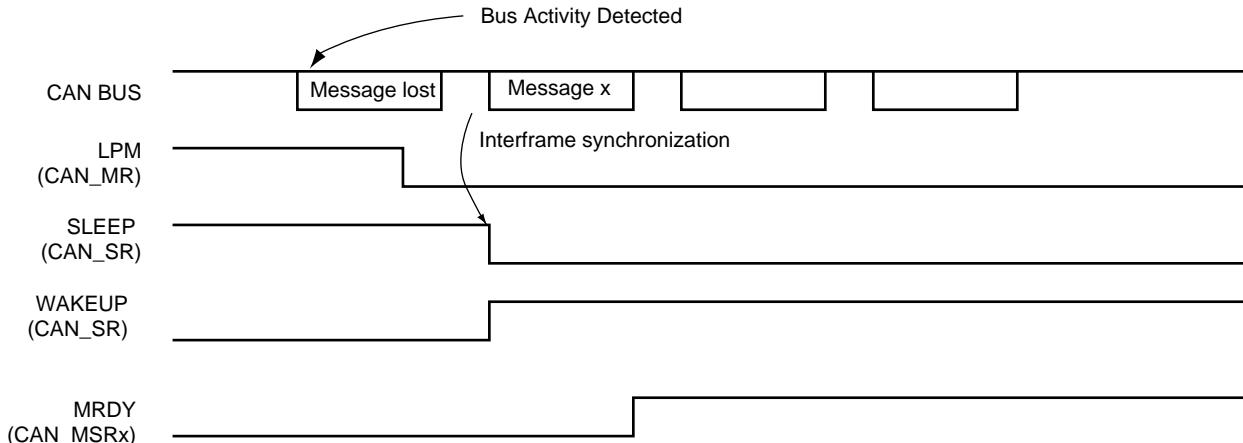
The CAN controller synchronizes itself with the bus activity by checking for eleven consecutive “recessive” bits. Once synchronized, the WAKEUP signal in the CAN\_SR register is set. The SLEEP signal in the CAN\_SR register is automatically cleared once WAKEUP is set. WAKEUP signal is automatically cleared once SLEEP is set.

Depending on the corresponding mask in the CAN\_IMR register, an interrupt is generated while WAKEUP is set. The SLEEP signal in the CAN\_SR register is automatically cleared once WAKEUP is set. WAKEUP signal is automatically cleared once SLEEP is set.

If no message is being sent on the bus, then the CAN controller is able to send a message eleven bit times after disabling Low-power Mode.

If there is bus activity when Low-power mode is disabled, the CAN controller is synchronized with the bus activity in the next interframe. The previous message is lost (see [Figure 32-9](#)).

**Figure 32-9. Disabling Low-power Mode**



## 32.8 Functional Description

### 32.8.1 CAN Controller Initialization

After power-up reset, the CAN controller is disabled. The CAN controller clock must be activated by the Power Management Controller (PMC) and the CAN controller interrupt line must be enabled by the interrupt controller (AIC).

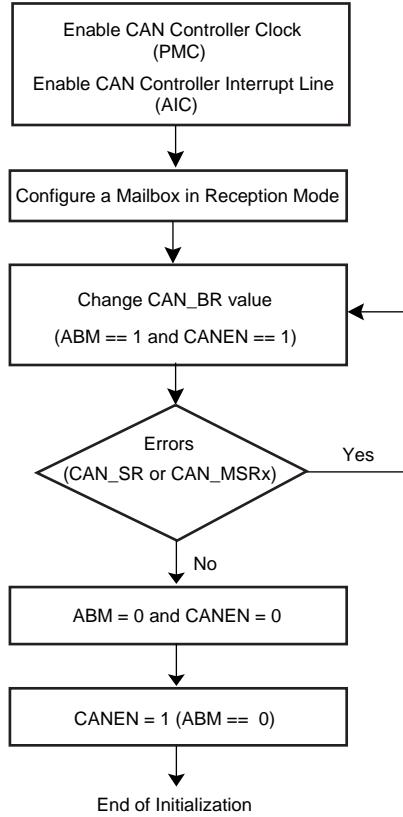
The CAN controller must be initialized with the CAN network parameters. The CAN\_BR register defines the sampling point in the bit time period. CAN\_BR must be set before the CAN controller is enabled by setting the CANEN field in the CAN\_MR register.

The CAN controller is enabled by setting the CANEN flag in the CAN\_MR register. At this stage, the internal CAN controller state machine is reset, error counters are reset to 0, error flags are reset to 0.

Once the CAN controller is enabled, bus synchronization is done automatically by scanning eleven recessive bits. The WAKEUP bit in the CAN\_SR register is automatically set to 1 when the CAN controller is synchronized (WAKEUP and SLEEP are stuck at 0 after a reset).

The CAN controller can start listening to the network in Autobaud Mode. In this case, the error counters are locked and a mailbox may be configured in Receive Mode. By scanning error flags, the CAN\_BR register values synchronize with the network. Once no error has been detected, the application disables the Autobaud Mode, clearing the ABM field in the CAN\_MR register.

**Figure 32-10. Possible Initialization Procedure**



### 32.8.2 CAN Controller Interrupt Handling

There are two different types of interrupts. One type of interrupt is a message-object related interrupt, the other is a system interrupt that handles errors or system-related interrupt sources.

All interrupt sources can be masked by writing the corresponding field in the CAN\_IDR register. They can be unmasked by writing to the CAN\_IER register. After a power-up reset, all interrupt sources are disabled (masked). The current mask status can be checked by reading the CAN\_IMR register.

The CAN\_SR register gives all interrupt source states.

The following events may initiate one of the two interrupts:

- Message object interrupt
  - Data registers in the mailbox object are available to the application. In Receive Mode, a new message was received. In Transmit Mode, a message was transmitted successfully.
  - A sent transmission was aborted.
- System interrupts
  - Bus off interrupt: The CAN module enters the bus off state.
  - Error passive interrupt: The CAN module enters Error Passive Mode.
  - Error Active Mode: The CAN module is neither in Error Passive Mode nor in Bus Off mode.
  - Warn Limit interrupt: The CAN module is in Error-active Mode, but at least one of its error counter value exceeds 96.
  - Wake-up interrupt: This interrupt is generated after a wake-up and a bus synchronization.
  - Sleep interrupt: This interrupt is generated after a Low-power Mode enable once all pending messages in transmission have been sent.
  - Internal timer counter overflow interrupt: This interrupt is generated when the internal timer rolls over.
  - Timestamp interrupt: This interrupt is generated after the reception or the transmission of a start of frame or an end of frame. The value of the internal counter is copied in the CAN\_TIMESTAMP register.

All interrupts are cleared by clearing the interrupt source except for the internal timer counter overflow interrupt and the timestamp interrupt. These interrupts are cleared by reading the CAN\_SR register.

### 32.8.3 CAN Controller Message Handling

#### 32.8.3.1 Receive Handling

Two modes are available to configure a mailbox to receive messages. In **Receive Mode**, the first message received is stored in the mailbox data register. In **Receive with Overwrite Mode**, the last message received is stored in the mailbox.

##### Simple Receive Mailbox

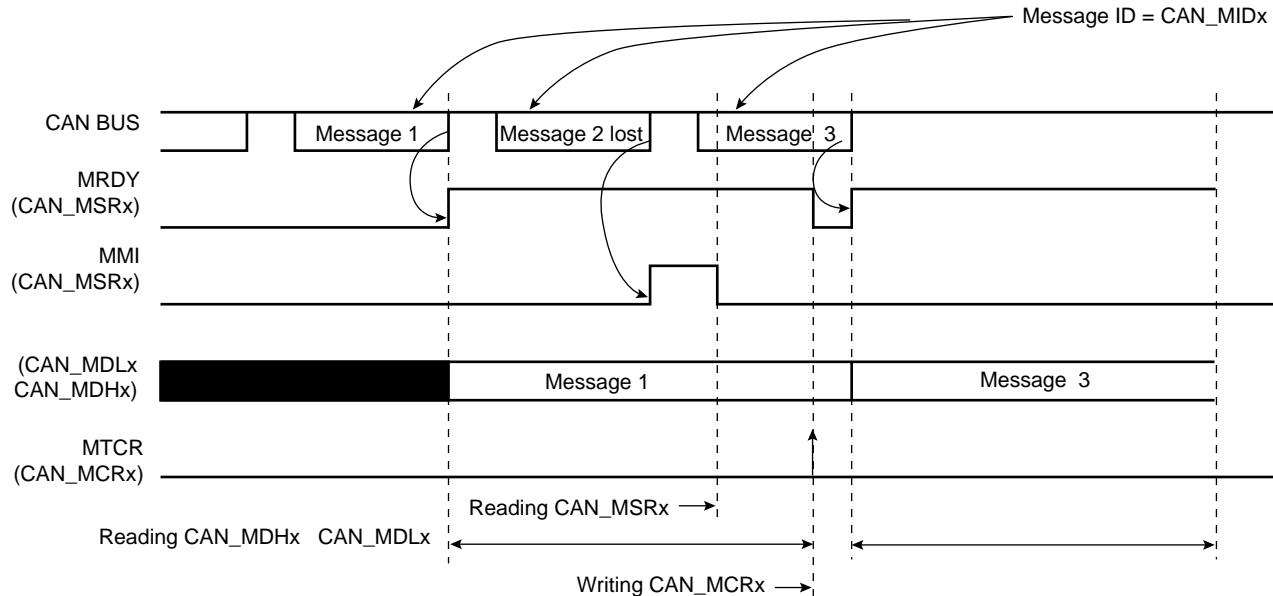
A mailbox is in Receive Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance Mask must be set before the Receive Mode is enabled.

After Receive Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically cleared until the first message is received. When the first message has been accepted by the mailbox, the MRDY flag is set. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked depending on the mailbox flag in the CAN\_IMR global register.

Message data are stored in the mailbox data register until the software application notifies that data processing has ended. This is done by asking for a new transfer command, setting the MTCR flag in the CAN\_MCRx register. This automatically clears the MRDY signal.

The MMI flag in the CAN\_MSRx register notifies the software that a message has been lost by the mailbox. This flag is set when messages are received while MRDY is set in the CAN\_MSRx register. This flag is cleared by reading the CAN\_MSRs register. A receive mailbox prevents from overwriting the first message by new ones while MRDY flag is set in the CAN\_MSRx register. See [Figure 32-11](#).

**Figure 32-11. Receive Mailbox**



Note: In the case of ARM architecture, CAN\_MSRx, CAN\_MDLx, CAN\_MDHx can be read using an optimized ldm assembler instruction.

#### Receive with Overwrite Mailbox

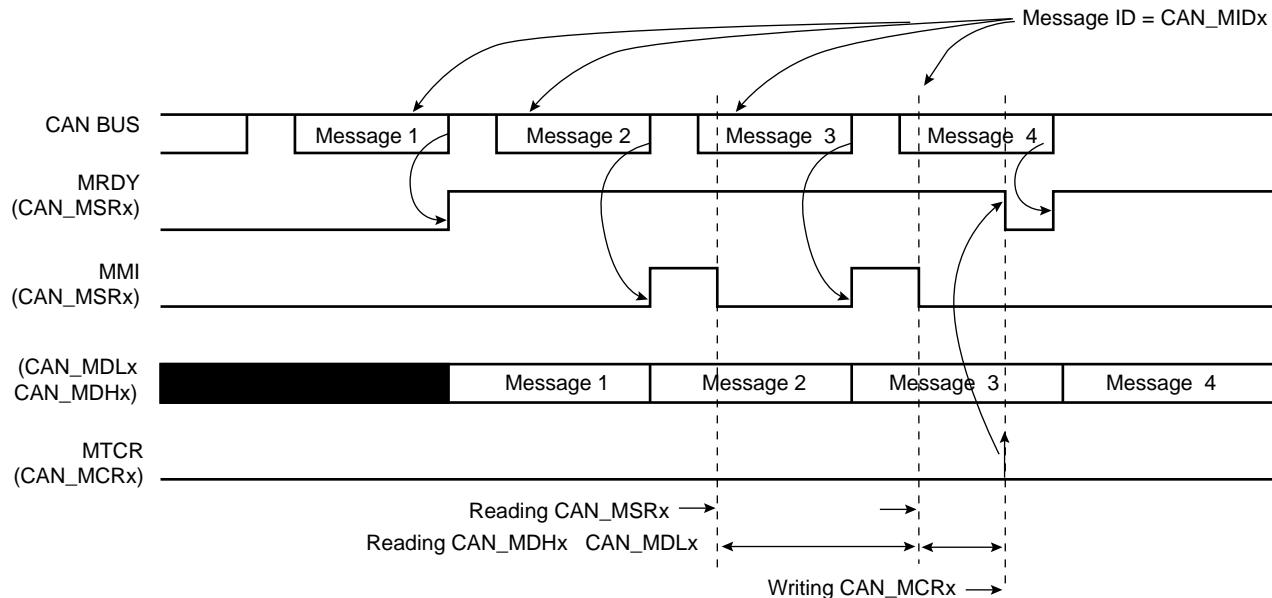
A mailbox is in Receive with Overwrite Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

After Receive Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically cleared until the first message is received. When the first message has been accepted by the mailbox, the MRDY flag is set. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt is masked depending on the mailbox flag in the CAN\_IMR global register.

If a new message is received while the MRDY flag is set, this new message is stored in the mailbox data register, overwriting the previous message. The MMI flag in the CAN\_MSR register notifies the software that a message has been dropped by the mailbox. This flag is cleared when reading the CAN\_MSR register.

The CAN controller may store a new message in the CAN data registers while the application reads them. To check that CAN\_MDHx and CAN\_MDLx do not belong to different messages, the application must check the MMI field in the CAN\_MSR register before and after reading CAN\_MDHx and CAN\_MDLx. If the MMI flag is set again after the data registers have been read, the software application has to re-read CAN\_MDHx and CAN\_MDLx (see [Figure 32-12](#)).

**Figure 32-12. Receive with Overwrite Mailbox**

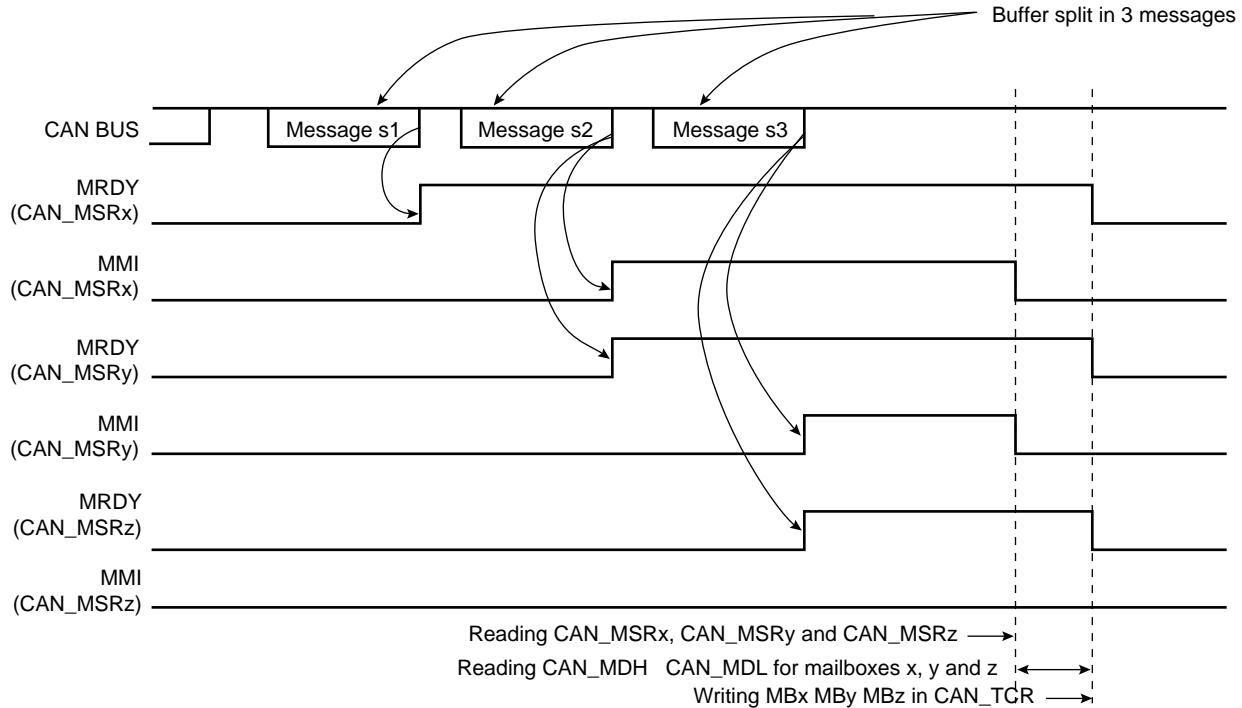


### Chaining Mailboxes

Several mailboxes may be used to receive a buffer split into several messages with the same ID. In this case, the mailbox with the lowest number is serviced first. In the receive and receive with overwrite modes, the field PRIOR in the CAN\_MMRx register has no effect. If Mailbox 0 and Mailbox 5 accept messages with the same ID, the first message is received by Mailbox 0 and the second message is received by Mailbox 5. Mailbox 0 must be configured in Receive Mode (i.e., the first message received is considered) and Mailbox 5 must be configured in Receive with Overwrite Mode. Mailbox 0 cannot be configured in Receive with Overwrite Mode; otherwise, all messages are accepted by this mailbox and Mailbox 5 is never serviced.

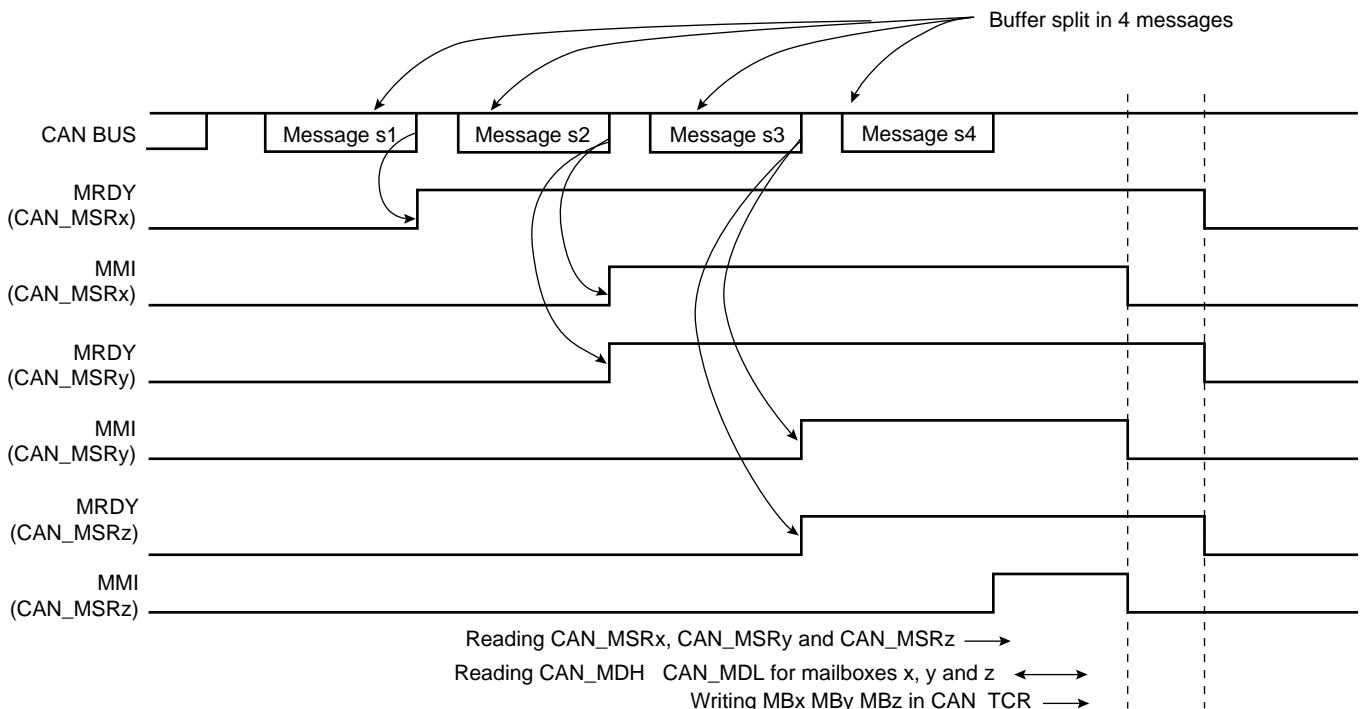
If several mailboxes are chained to receive a buffer split into several messages, all mailboxes except the last one (with the highest number) must be configured in Receive Mode. The first message received is handled by the first mailbox, the second one is refused by the first mailbox and accepted by the second mailbox, the last message is accepted by the last mailbox and refused by previous ones (see [Figure 32-13](#)).

**Figure 32-13. Chaining Three Mailboxes to Receive a Buffer Split into Three Messages**



If the number of mailboxes is not sufficient (the MMI flag of the last mailbox raises), the user must read each data received on the last mailbox in order to retrieve all the messages of the buffer split (see [Figure 32-14](#)).

**Figure 32-14. Chaining Three Mailboxes to Receive a Buffer Split into Four Messages**



### 32.8.3.2 Transmission Handling

A mailbox is in Transmit Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance mask must be set before Receive Mode is enabled.

After Transmit Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically set until the first command is sent. When the MRDY flag is set, the software application can prepare a message to be sent by writing to the CAN\_MDx registers. The message is sent once the software asks for a transfer command setting the MTCR bit and the message data length in the CAN\_MCRx register.

The MRDY flag remains at zero as long as the message has not been sent or aborted. It is important to note that no access to the mailbox data register is allowed while the MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked depending on the mailbox flag in the CAN\_IMR global register.

It is also possible to send a remote frame setting the MRTR bit instead of setting the MDLC field. The answer to the remote frame is handled by another reception mailbox. In this case, the device acts as a consumer but with the help of two mailboxes. It is possible to handle the remote frame emission and the answer reception using only one mailbox configured in Consumer Mode. Refer to the section “[Remote Frame Handling](#)” on page 620.

Several messages can try to win the bus arbitration in the same time. The message with the highest priority is sent first. Several transfer request commands can be generated at the same time by setting MBx bits in the CAN\_TCR register. The priority is set in the PRIOR field of the CAN\_MMRx register. Priority 0 is the highest priority, priority 15 is the lowest priority. Thus it is possible to use a part of the message ID to set the PRIOR field. If two mailboxes have the same priority, the message of the mailbox with the lowest number is sent first. Thus if mailbox 0 and mailbox 5 have the same priority and have a message to send at the same time, then the message of the mailbox 0 is sent first.

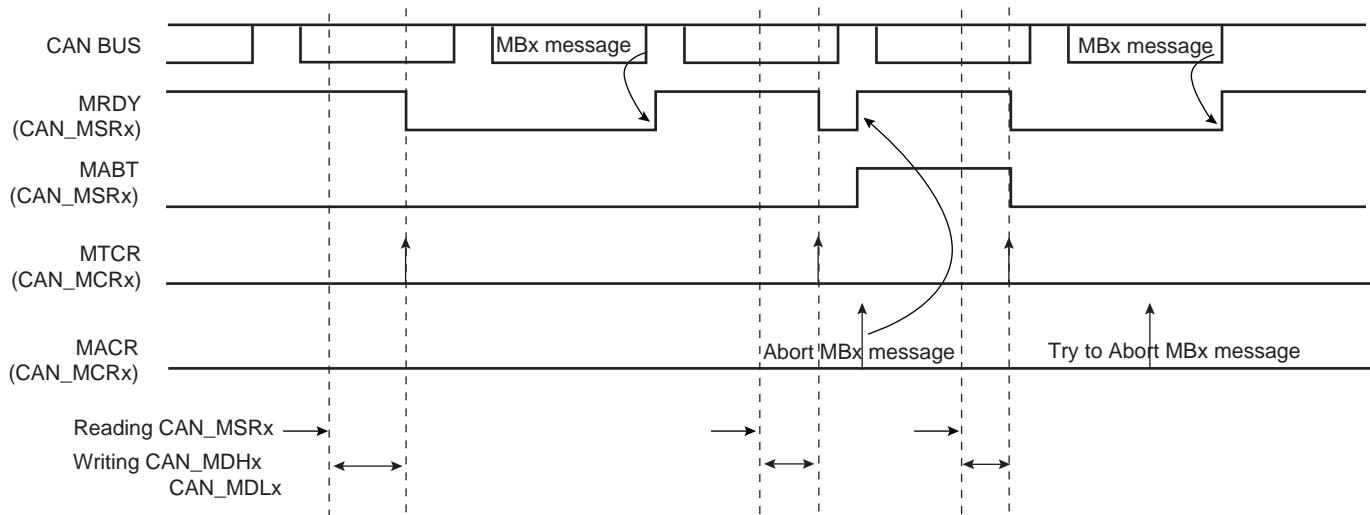
Setting the MACR bit in the CAN\_MCRx register aborts the transmission. Transmission for several mailboxes can be aborted by writing MBx fields in the CAN\_MACR register. If the message is being sent when the abort command is set, then the application is notified by the MRDY bit set and not the MABT in the CAN\_MSRx register. Otherwise, if the message has not been sent, then the MRDY and the MABT are set in the CAN\_MSR register.

When the bus arbitration is lost by a mailbox message, the CAN controller tries to win the next bus arbitration with the same message if this one still has the highest priority. Messages to be sent are re-tried automatically until they win the bus arbitration. This feature can be disabled by setting the bit DRPT in the CAN\_MR register. In this case if the message was not sent the first time it was transmitted to the CAN transceiver, it is automatically aborted. The MABT flag is set in the CAN\_MSRx register until the next transfer command.

[Figure 32-15](#) shows three MBx message attempts being made (MRDY of MBx set to 0).

The first MBx message is sent, the second is aborted and the last one is trying to be aborted but too late because it has already been transmitted to the CAN transceiver.

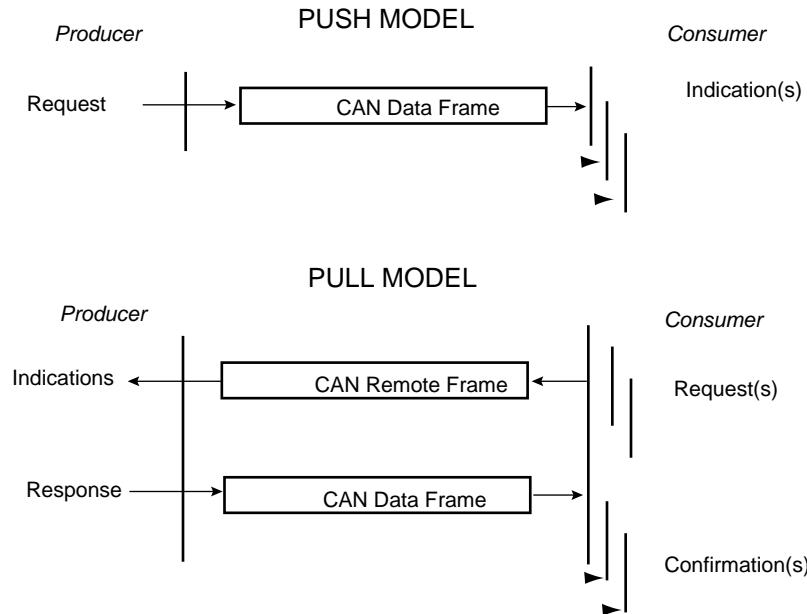
**Figure 32-15. Transmitting Messages**



### 32.8.3.3 Remote Frame Handling

Producer/consumer model is an efficient means of handling broadcasted messages. The push model allows a producer to broadcast messages; the pull model allows a customer to ask for messages.

**Figure 32-16. Producer / Consumer Model**



In Pull Mode, a consumer transmits a remote frame to the producer. When the producer receives a remote frame, it sends the answer accepted by one or many consumers. Using transmit and receive mailboxes, a consumer must dedicate two mailboxes, one in Transmit Mode to send remote frames, and at least one in Receive Mode to capture the producer's answer. The same structure is applicable to a producer: one reception mailbox is required to get the remote frame and one transmit mailbox to answer.

Mailboxes can be configured in Producer or Consumer Mode. A lonely mailbox can handle the remote frame and the answer. With 8 mailboxes, the CAN controller can handle 8 independent producers/consumers.

#### Producer Configuration

A mailbox is in Producer Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

After Producer Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically set until the first transfer command. The software application prepares data to be sent by writing to the CAN\_MDHx and the CAN\_MDLx registers, then by setting the MTCR bit in the CAN\_MCRx register. Data is sent after the reception of a remote frame as soon as it wins the bus arbitration.

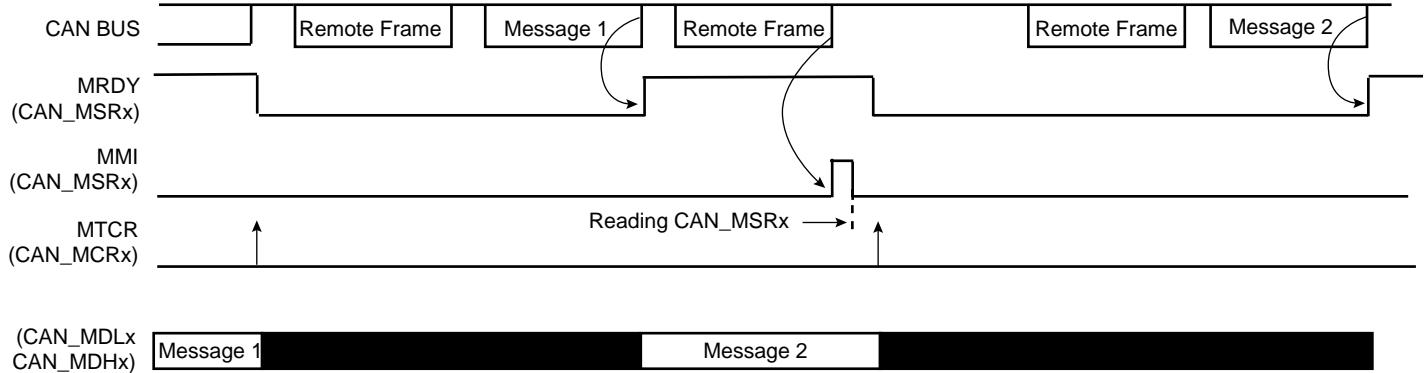
The MRDY flag remains at zero as long as the message has not been sent or aborted. No access to the mailbox data register can be done while MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked according to the mailbox flag in the CAN\_IMR global register.

If a remote frame is received while no data are ready to be sent (signal MRDY set in the CAN\_MSRx register), then the MMI signal is set in the CAN\_MSRx register. This bit is cleared by reading the CAN\_MSRx register.

The MRTR field in the CAN\_MSRx register has no meaning. This field is used only when using Receive and Receive with Overwrite modes.

After a remote frame has been received, the mailbox functions like a transmit mailbox. The message with the highest priority is sent first. The transmitted message may be aborted by setting the MACR bit in the CAN\_MCR register. Please refer to the section "[Transmission Handling](#)" on page 619.

**Figure 32-17. Producer Handling**



#### Consumer Configuration

A mailbox is in Consumer Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

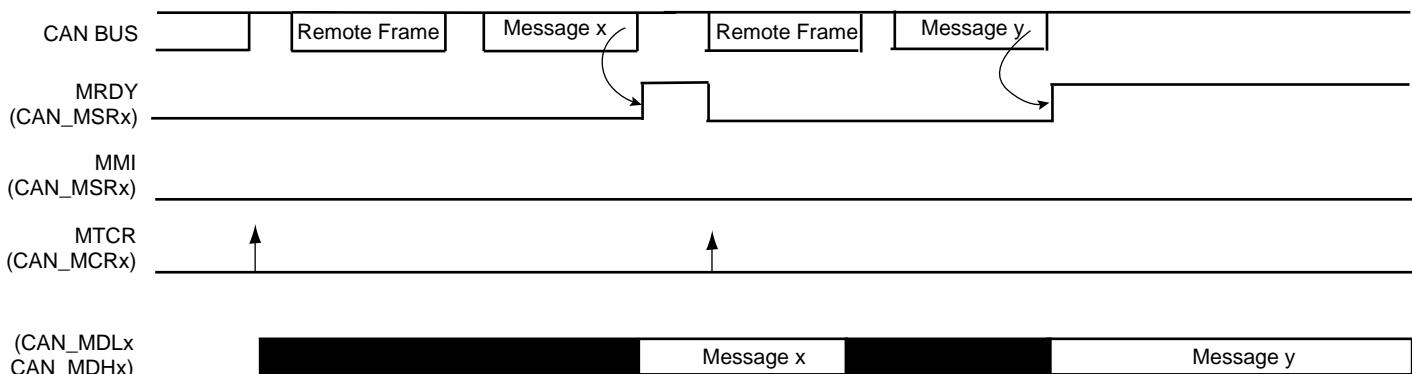
After Consumer Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically cleared until the first transfer request command. The software application sends a remote frame by setting the MTCR bit in the CAN\_MCRx register or the MBx bit in the global CAN\_TCR register. The application is notified of the answer by the MRDY flag set in the CAN\_MSRx register. The application can read the data contents in the CAN\_MDHx and CAN\_MDLx registers. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked according to the mailbox flag in the CAN\_IMR global register.

The MRTR bit in the CAN\_MCRx register has no effect. This field is used only when using Transmit Mode.

After a remote frame has been sent, the consumer mailbox functions as a reception mailbox. The first message received is stored in the mailbox data registers. If other messages intended for this mailbox have been sent while the MRDY flag is set in the CAN\_MSRx register, they will be lost. The application is notified by reading the MMI field in the CAN\_MSRx register. The read operation automatically clears the MMI flag.

If several messages are answered by the Producer, the CAN controller may have one mailbox in consumer configuration, zero or several mailboxes in Receive Mode and one mailbox in Receive with Overwrite Mode. In this case, the consumer mailbox must have a lower number than the Receive with Overwrite mailbox. The transfer command can be triggered for all mailboxes at the same time by setting several MBx fields in the CAN\_TCR register.

**Figure 32-18. Consumer Handling**



### 32.8.4 CAN Controller Timing Modes

Using the free running 16-bit internal timer, the CAN controller can be set in one of the two following timing modes:

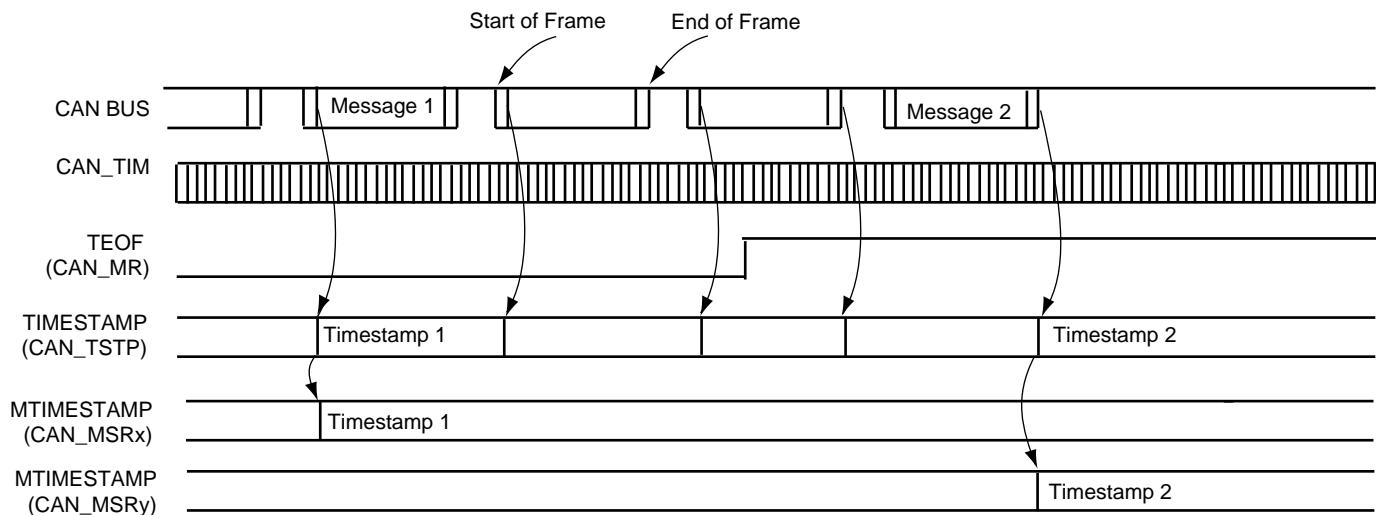
- Timestamping Mode: The value of the internal timer is captured at each Start Of Frame or each End Of Frame.
- Time Triggered Mode: The mailbox transfer operation is triggered when the internal timer reaches the mailbox trigger.

Timestamping Mode is enabled by clearing the TTM bit in the CAN\_MR register. Time Triggered Mode is enabled by setting the TTM bit in the CAN\_MR register.

#### 32.8.4.1 Timestamping Mode

Each mailbox has its own timestamp value. Each time a message is sent or received by a mailbox, the 16-bit value MTIMESTAMP of the CAN\_TIMESTP register is transferred to the LSB bits of the CAN\_MSRx register. The value read in the CAN\_MSRx register corresponds to the internal timer value at the Start Of Frame or the End Of Frame of the message handled by the mailbox.

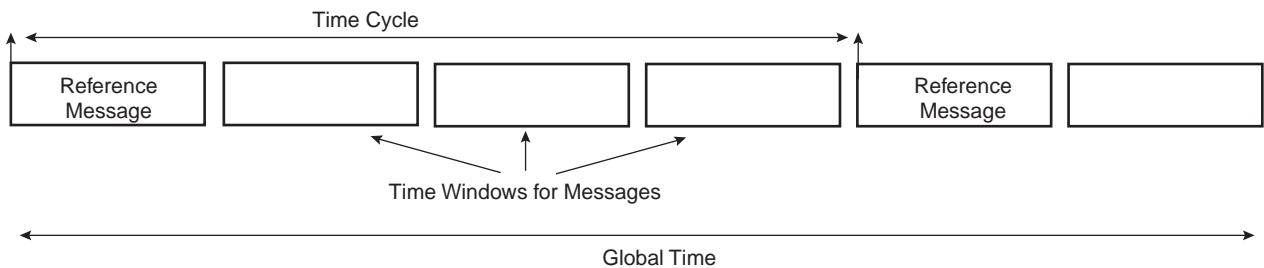
**Figure 32-19. Mailbox Timestamp**



#### 32.8.4.2 Time Triggered Mode

In Time Triggered Mode, basic cycles can be split into several time windows. A basic cycle starts with a reference message. Each time a window is defined from the reference message, a transmit operation should occur within a pre-defined time window. A mailbox must not win the arbitration in a previous time window, and it must not be retried if the arbitration is lost in the time window.

**Figure 32-20. Time Triggered Principle**



Time Trigger Mode is enabled by setting the TTM field in the CAN\_MR register. In Time Triggered Mode, as in Timestamp Mode, the CAN\_TIMESTP field captures the values of the internal counter, but the MTIMESTAMP fields in the CAN\_MSRx registers are not active and are read at 0.

### *Synchronization by a Reference Message*

In Time Triggered Mode, the internal timer counter is automatically reset when a new message is received in the last mailbox. This reset occurs after the reception of the End Of Frame on the rising edge of the MRDY signal in the CAN\_MSRx register. This allows synchronization of the internal timer counter with the reception of a reference message and the start a new time window.

### *Transmitting within a Time Window*

A time mark is defined for each mailbox. It is defined in the 16-bit MTIMEMARK field of the CAN\_MMRx register. At each internal timer clock cycle, the value of the CAN\_TIM is compared with each mailbox time mark. When the internal timer counter reaches the MTIMEMARK value, an internal timer event for the mailbox is generated for the mailbox.

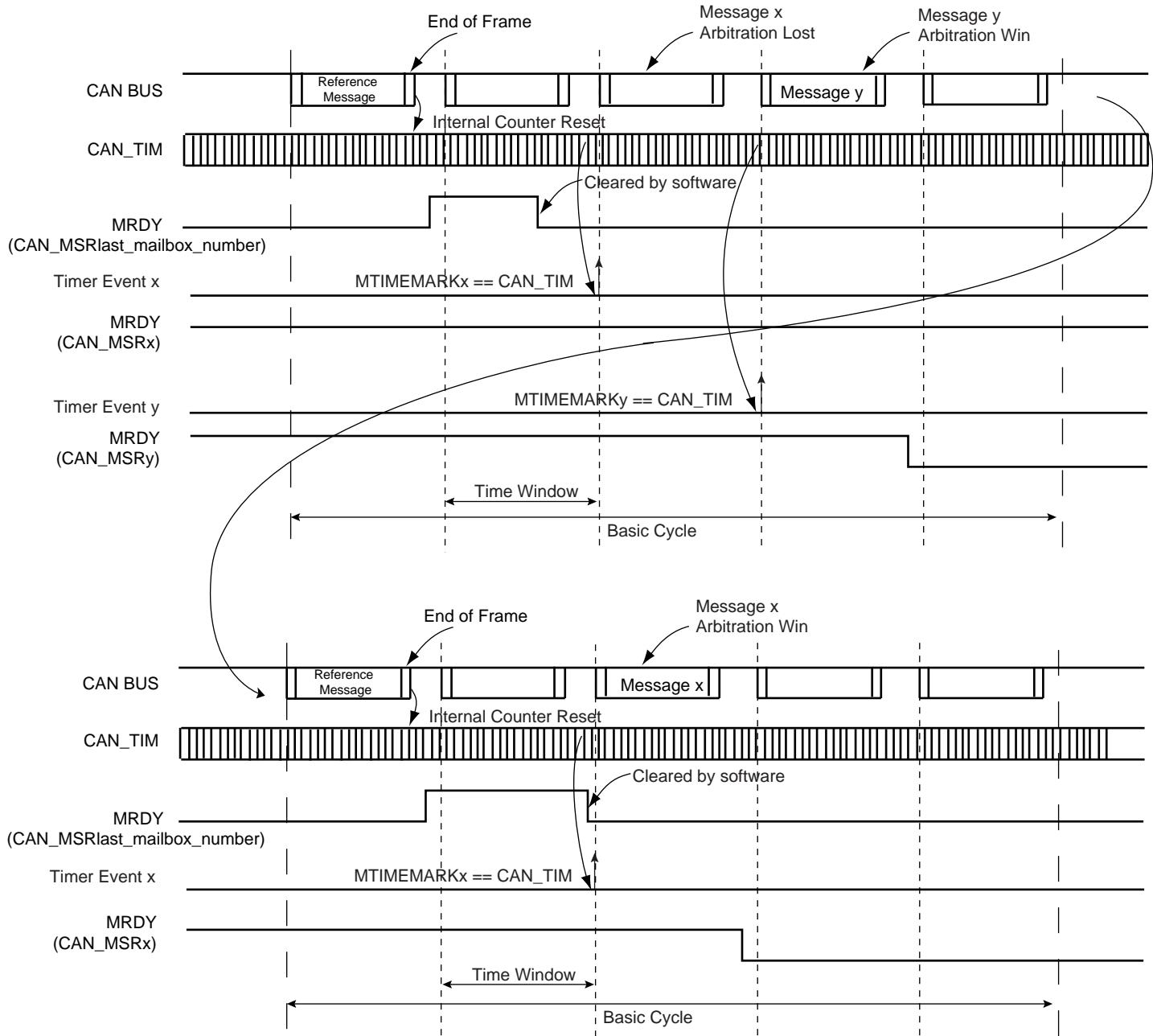
In Time Triggered Mode, transmit operations are delayed until the internal timer event for the mailbox. The application prepares a message to be sent by setting the MTCR in the CAN\_MCRx register. The message is not sent until the CAN\_TIM value is less than the MTIMEMARK value defined in the CAN\_MMRx register.

If the transmit operation is failed, i.e., the message loses the bus arbitration and the next transmit attempt is delayed until the next internal time trigger event. This prevents overlapping the next time window, but the message is still pending and is retried in the next time window when CAN\_TIM value equals the MTIMEMARK value. It is also possible to prevent a retry by setting the DRPT field in the CAN\_MR register.

### *Freezing the Internal Timer Counter*

The internal counter can be frozen by setting TIMFRZ in the CAN\_MR register. This prevents an unexpected roll-over when the counter reaches FFFFh. When this occurs, it automatically freezes until a new reset is issued, either due to a message received in the last mailbox or any other reset counter operations. The TOVF bit in the CAN\_SR register is set when the counter is frozen. The TOVF bit in the CAN\_SR register is cleared by reading the CAN\_SR register. Depending on the corresponding interrupt mask in the CAN\_IMR register, an interrupt is generated when TOVF is set.

**Figure 32-21. Time Triggered Operations**



### 32.8.5 Write Protected Registers

To prevent any single software error that may corrupt CAN behavior, the registers listed below can be write-protected by setting the WPEN bit in the CAN Write Protection Mode Register (CAN\_WPMR).

If a write access in a write-protected register is detected, then the WPVS flag in the CAN Write Protection Status Register (CAN\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is automatically reset after reading the CAN Write Protection Status Register (CAN\_WPSR).

The protected registers are:

- “CAN Mode Register”
- “CAN Baudrate Register”
- “CAN Message Mode Register”
- “CAN Message Acceptance Mask Register”
- “CAN Message ID Register”

## 32.9 Controller Area Network (CAN) User Interface

Table 32-6. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Mode Register	CAN_MR	Read-write	0x0
0x0004	Interrupt Enable Register	CAN_IER	Write-only	-
0x0008	Interrupt Disable Register	CAN_IDR	Write-only	-
0x000C	Interrupt Mask Register	CAN_IMR	Read-only	0x0
0x0010	Status Register	CAN_SR	Read-only	0x0
0x0014	Baudrate Register	CAN_BR	Read-write	0x0
0x0018	Timer Register	CAN_TIM	Read-only	0x0
0x001C	Timestamp Register	CAN_TIMESTAMP	Read-only	0x0
0x0020	Error Counter Register	CAN_ECR	Read-only	0x0
0x0024	Transfer Command Register	CAN_TCR	Write-only	-
0x0028	Abort Command Register	CAN_ACR	Write-only	-
0x002C - 0x00E0	Reserved	-	-	-
0x00E4	Write Protect Mode Register	CAN_WPMR	Read-write	0x0
0x00E8	Write Protect Status Register	CAN_WPSR	Read-only	0x0
0x00EC - 0x01FC	Reserved	-	-	-
0x0200 + MB * 0x20 + 0x00	Mailbox Mode Register <sup>(1)</sup>	CAN_MMR	Read-write	0x0
0x0200 + MB * 0x20 + 0x04	Mailbox Acceptance Mask Register	CAN_MAM	Read-write	0x0
0x0200 + MB * 0x20 + 0x08	Mailbox ID Register	CAN_MID	Read-write	0x0
0x0200 + MB * 0x20 + 0x0C	Mailbox Family ID Register	CAN_MFID	Read-only	0x0
0x0200 + MB * 0x20 + 0x10	Mailbox Status Register	CAN_MSR	Read-only	0x0
0x0200 + MB * 0x20 + 0x14	Mailbox Data Low Register	CAN_MDL	Read-write	0x0
0x0200 + MB * 0x20 + 0x18	Mailbox Data High Register	CAN_MDH	Read-write	0x0
0x0200 + MB * 0x20 + 0x1C	Mailbox Control Register	CAN_MCR	Write-only	-

Note: 1. Mailbox number ranges from 0 to 7.

### 32.9.1 CAN Mode Register

**Name:** CAN\_MR

**Address:** 0x40010000 (0), 0x40014000 (1)

**Access:** Read-write

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
DRPT	TIMFRZ	TTM	TEOF	OVL	ABM	LPM	CANEN

This register can only be written if the WPEN bit is cleared in "CAN Write Protection Mode Register".

- **CANEN: CAN Controller Enable**

0: The CAN Controller is disabled.

1: The CAN Controller is enabled.

- **LPM: Disable/Enable Low Power Mode**

w Power Mode.

1: Enable Low Power M

CAN controller enters Low Power Mode once all pending messages have been transmitted.

- **ABM: Disable/Enable Autobaud/Listen mode**

0: Disable Autobaud/listen mode.

1: Enable Autobaud/listen mode.

- **OVL: Disable/Enable Overload Frame**

0: No overload frame is generated.

1: An overload frame is generated after each successful reception for mailboxes configured in Receive with/without overwrite Mode, Producer and Consumer.

- **TEOF: Timestamp messages at each end of Frame**

0: The value of CAN\_TIM is captured in the CAN\_TIMESTAMP register at each Start Of Frame.

1: The value of CAN\_TIM is captured in the CAN\_TIMESTAMP register at each End Of Frame.

- **TTM: Disable/Enable Time Triggered Mode**

0: Time Triggered Mode is disabled.

1: Time Triggered Mode is enabled.

- **TIMFRZ: Enable Timer Freeze**

0: The internal timer continues to be incremented after it reached 0xFFFF.

1: The internal timer stops incrementing after reaching 0xFFFF. It is restarted after a timer reset. See "Freezing the Internal Timer Counter" on page 623.

- **DRPT: Disable Repeat**

0: When a transmit mailbox loses the bus arbitration, the transfer request remains pending.

1: When a transmit mailbox lose the bus arbitration, the transfer request is automatically aborted. It automatically raises the MABT and MRDT flags in the corresponding CAN\_MSRx.

### 32.9.2 CAN Interrupt Enable Register

**Name:** CAN\_IER

**Address:** 0x40010004 (0), 0x40014004 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
-	-	-	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- MBx: Mailbox x Interrupt Enable**

0: No effect.

1: Enable Mailbox x interrupt.

- ERRA: Error Active Mode Interrupt Enable**

0: No effect.

1: Enable ERRA interrupt.

- WARN: Warning Limit Interrupt Enable**

0: No effect.

1: Enable WARN interrupt.

- ERRP: Error Passive Mode Interrupt Enable**

0: No effect.

1: Enable ERRP interrupt.

- BOFF: Bus Off Mode Interrupt Enable**

0: No effect.

1: Enable BOFF interrupt.

- SLEEP: Sleep Interrupt Enable**

0: No effect.

1: Enable SLEEP interrupt.

- WAKEUP: Wake-up Interrupt Enable**

0: No effect.

1: Enable WAKEUP interrupt.

- TOVF: Timer Overflow Interrupt Enable**

0: No effect.

1: Enable TOVF interrupt.

- **TSTP: TimeStamp Interrupt Enable**

0: No effect.

1: Enable TSTP interrupt.

- **CERR: CRC Error Interrupt Enable**

0: No effect.

1: Enable CRC Error interrupt.

- **SERR: Stuffing Error Interrupt Enable**

0: No effect.

1: Enable Stuffing Error interrupt.

- **AERR: Acknowledgment Error Interrupt Enable**

0: No effect.

1: Enable Acknowledgment Error interrupt.

- **FERR: Form Error Interrupt Enable**

0: No effect.

1: Enable Form Error interrupt.

- **BERR: Bit Error Interrupt Enable**

0: No effect.

1: Enable Bit Error interrupt.

### 32.9.3 CAN Interrupt Disable Register

**Name:** CAN\_IDR

**Address:** 0x40010008 (0), 0x40014008 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Disable**

0: No effect.

1: Disable Mailbox x interrupt.

- **ERRA: Error Active Mode Interrupt Disable**

0: No effect.

1: Disable ERRA interrupt.

- **WARN: Warning Limit Interrupt Disable**

0: No effect.

1: Disable WARN interrupt.

- **ERRP: Error Passive Mode Interrupt Disable**

0: No effect.

1: Disable EERP interrupt.

- **BOFF: Bus Off Mode Interrupt Disable**

0: No effect.

1: Disable BOFF interrupt.

- **SLEEP: Sleep Interrupt Disable**

0: No effect.

1: Disable SLEEP interrupt.

- **WAKEUP: Wake-up Interrupt Disable**

0: No effect.

1: Disable WAKEUP interrupt.

- **TOVF: Timer Overflow Interrupt**

0: No effect.

1: Disable TOVF interrupt.

- **TSTP: TimeStamp Interrupt Disable**

0: No effect.

1: Disable TSTP interrupt.

- **CERR: CRC Error Interrupt Disable**

0: No effect.

1: Disable CRC Error interrupt.

- **SERR: Stuffing Error Interrupt Disable**

0: No effect.

1: Disable Stuffing Error interrupt.

- **AERR: Acknowledgment Error Interrupt Disable**

0: No effect.

1: Disable Acknowledgment Error interrupt.

- **FERR: Form Error Interrupt Disable**

0: No effect.

1: Disable Form Error interrupt.

- **BERR: Bit Error Interrupt Disable**

0: No effect.

1: Disable Bit Error interrupt.

### 32.9.4 CAN Interrupt Mask Register

**Name:** CAN\_IMR

**Address:** 0x4001000C (0), 0x4001400C (1)

**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Mask**

0: Mailbox x interrupt is disabled.

1: Mailbox x interrupt is enabled.

- **ERRA: Error Active Mode Interrupt Mask**

0: ERRA interrupt is disabled.

1: ERRA interrupt is enabled.

- **WARN: Warning Limit Interrupt Mask**

0: Warning Limit interrupt is disabled.

1: Warning Limit interrupt is enabled.

- **ERRP: Error Passive Mode Interrupt Mask**

0: EERRP interrupt is disabled.

1: EERRP interrupt is enabled.

- **BOFF: Bus Off Mode Interrupt Mask**

0: BOFF interrupt is disabled.

1: BOFF interrupt is enabled.

- **SLEEP: Sleep Interrupt Mask**

0: SLEEP interrupt is disabled.

1: SLEEP interrupt is enabled.

- **WAKEUP: Wake-up Interrupt Mask**

0: WAKEUP interrupt is disabled.

1: WAKEUP interrupt is enabled.

- **TOVF: Timer Overflow Interrupt Mask**

0: TOVF interrupt is disabled.

1: TOVF interrupt is enabled.

- **TSTP: Timestamp Interrupt Mask**

0: TSTP interrupt is disabled.

1: TSTP interrupt is enabled.

- **CERR: CRC Error Interrupt Mask**

0: CRC Error interrupt is disabled.

1: CRC Error interrupt is enabled.

- **SERR: Stuffing Error Interrupt Mask**

0: Bit Stuffing Error interrupt is disabled.

1: Bit Stuffing Error interrupt is enabled.

- **AERR: Acknowledgment Error Interrupt Mask**

0: Acknowledgment Error interrupt is disabled.

1: Acknowledgment Error interrupt is enabled.

- **FERR: Form Error Interrupt Mask**

0: Form Error interrupt is disabled.

1: Form Error interrupt is enabled.

- **BERR: Bit Error Interrupt Mask**

0: Bit Error interrupt is disabled.

1: Bit Error interrupt is enabled.

### 32.9.5 CAN Status Register

**Name:** CAN\_SR

**Address:** 0x40010010 (0), 0x40014010 (1)

**Access:** Read-only

31	30	29	28	27	26	25	24
OVLSY	TBSY	RBSY	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	—
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Event**

0: No event occurred on Mailbox x.

1: An event occurred on Mailbox x.

An event corresponds to MRDY, MABT fields in the CAN\_MSRx register.

- **ERRA: Error Active Mode**

0: CAN controller has not reached Error Active Mode since the last read of CAN\_SR.

1: CAN controller has reached Error Active Mode since the last read of CAN\_SR.

This flag is automatically cleared by reading CAN\_SR register.

This flag is set depending on TEC and REC counter values. It is set when a node is neither in Error Passive Mode nor in Bus Off Mode.

- **WARN: Warning Limit**

0: CAN controller Warning Limit has not been reached since the last read of CAN\_SR.

1: CAN controller Warning Limit has been reached since the last read of CAN\_SR.

This flag is automatically cleared by reading CAN\_SR register.

This flag is set depending on TEC and REC counter values. It is set when at least one of the counter values has reached a value greater or equal to 96.

- **ERRP: Error Passive Mode**

0: CAN controller has not reached Error Passive Mode since the last read of CAN\_SR.

1: CAN controller has reached Error Passive Mode since the last read of CAN\_SR.

This flag is set depending on TEC and REC counters values.

This flag is automatically cleared by reading CAN\_SR register.

A node is in error passive state when TEC counter is greater or equal to 128 (decimal) or when the REC counter is greater or equal to 128 (decimal).

- **BOFF: Bus Off Mode**

0: CAN controller has not reached Bus Off Mode.

1: CAN controller has reached Bus Off Mode since the last read of CAN\_SR.

This flag is automatically cleared by reading CAN\_SR register.

This flag is set depending on TEC counter value. A node is in bus off state when TEC counter is greater or equal to 256 (decimal).

- **SLEEP: CAN controller in Low power Mode**

0: CAN controller is not in low power mode.

1: CAN controller is in low power mode.

This flag is automatically reset when Low power mode is disabled

- **WAKEUP: CAN controller is not in Low power Mode**

0: CAN controller is in low power mode.

1: CAN controller is not in low power mode.

When a WAKEUP event occurs, the CAN controller is synchronized with the bus activity. Messages can be transmitted or received. The CAN controller clock must be available when a WAKEUP event occurs. This flag is automatically reset when the CAN Controller enters Low Power mode.

- **TOVF: Timer Overflow**

0: The timer has not rolled-over FFFFh to 0000h.

1: The timer rolls-over FFFFh to 0000h.

This flag is automatically cleared by reading CAN\_SR register.

- **TSTP Timestamp**

0: No bus activity has been detected.

1: A start of frame or an end of frame has been detected (according to the TEOF field in the CAN\_MR register).

This flag is automatically cleared by reading the CAN\_SR register.

- **CERR: Mailbox CRC Error**

0: No CRC error occurred during a previous transfer.

1: A CRC error occurred during a previous transfer.

A CRC error has been detected during last reception.

This flag is automatically cleared by reading CAN\_SR register.

- **SERR: Mailbox Stuffing Error**

0: No stuffing error occurred during a previous transfer.

1: A stuffing error occurred during a previous transfer.

A form error results from the detection of more than five consecutive bit with the same polarity.

This flag is automatically cleared by reading CAN\_SR register.

- **AERR: Acknowledgment Error**

0: No acknowledgment error occurred during a previous transfer.

1: An acknowledgment error occurred during a previous transfer.

An acknowledgment error is detected when no detection of the dominant bit in the acknowledge slot occurs.

This flag is automatically cleared by reading CAN\_SR register.

- **FERR: Form Error**

0: No form error occurred during a previous transfer

1: A form error occurred during a previous transfer

A form error results from violations on one or more of the fixed form of the following bit fields:

- CRC delimiter

- ACK delimiter
- End of frame
- Error delimiter
- Overload delimiter

This flag is automatically cleared by reading CAN\_SR register.

- **BERR: Bit Error**

0: No bit error occurred during a previous transfer.

1: A bit error occurred during a previous transfer.

A bit error is set when the bit value monitored on the line is different from the bit value sent.

This flag is automatically cleared by reading CAN\_SR register.

- **RBSY: Receiver busy**

0: CAN receiver is not receiving a frame.

1: CAN receiver is receiving a frame.

Receiver busy. This status bit is set by hardware while CAN receiver is acquiring or monitoring a frame (remote, data, overload or error frame). It is automatically reset when CAN is not receiving.

- **TBSY: Transmitter busy**

0: CAN transmitter is not transmitting a frame.

1: CAN transmitter is transmitting a frame.

Transmitter busy. This status bit is set by hardware while CAN transmitter is generating a frame (remote, data, overload or error frame). It is automatically reset when CAN is not transmitting.

- **OVLSY: Overload busy**

0: CAN transmitter is not transmitting an overload frame.

1: CAN transmitter is transmitting a overload frame.

It is automatically reset when the bus is not transmitting an overload frame.

### 32.9.6 CAN Baudrate Register

**Name:** CAN\_BR

**Address:** 0x40010014 (0), 0x40014014 (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	SMP
23	22	21	20	19	18	17	16
BRP							
15	14	13	12	11	10	9	8
—	—	SJW		—	PROPAG		
7	6	5	4	3	2	1	0
—	PHASE1			—	PHASE2		

This register can only be written if the WPEN bit is cleared in "CAN Write Protection Mode Register".

Any modification on one of the fields of the CAN\_BR register must be done while CAN module is disabled.

To compute the different Bit Timings, please refer to the [Section 32.7.4.1 "CAN Bit Timing Configuration" on page 606](#).

- **PHASE2: Phase 2 segment**

This phase is used to compensate the edge phase error.

$$t_{PHS2} = t_{CSC} \times (\text{PHASE2} + 1)$$

**Warning:** PHASE2 value must be different from 0.

- **PHASE1: Phase 1 segment**

This phase is used to compensate for edge phase error.

$$t_{PHS1} = t_{CSC} \times (\text{PHASE1} + 1)$$

- **PROPAG: Programming time segment**

This part of the bit time is used to compensate for the physical delay times within the network.

$$t_{PRS} = t_{CSC} \times (\text{PROPAG} + 1)$$

- **SJW: Re-synchronization jump width**

To compensate for phase shifts between clock oscillators of different controllers on bus. The controller must re-synchronize on any relevant signal edge of the current transmission. The synchronization jump width defines the maximum of clock cycles a bit period may be shortened or lengthened by re-synchronization.

$$t_{SJW} = t_{CSC} \times (\text{SJW} + 1)$$

- **BRP: Baudrate Prescaler.**

This field allows user to program the period of the CAN system clock to determine the individual bit timing.

$$t_{CSC} = (\text{BRP} + 1)/\text{MCK}$$

The BRP field must be within the range [1, 0x7F], i.e., BRP = 0 is not authorized.

- **SMP: Sampling Mode**

0 (ONCE): The incoming bit stream is sampled once at sample point.

1 (THREE): The incoming bit stream is sampled three times with a period of a MCK clock period, centered on sample point.

SMP Sampling Mode is automatically disabled if BRP = 0.

### 32.9.7 CAN Timer Register

**Name:** CAN\_TIM

**Address:** 0x40010018 (0), 0x40014018 (1)

**Access:** Read-only

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

- **TIMER: Timer**

This field represents the internal CAN controller 16-bit timer value.

### 32.9.8 CAN Timestamp Register

**Name:** CAN\_TIMESTAMP

**Address:** 0x4001001C (0), 0x4001401C (1)

**Access:** Read-only

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

- **MTIMESTAMP: Timestamp**

This field carries the value of the internal CAN controller 16-bit timer value at the start or end of frame.

If the TEOF bit is cleared in the CAN\_MR register, the internal Timer Counter value is captured in the MTIMESTAMP field at each start of frame else the value is captured at each end of frame. When the value is captured, the TSTP flag is set in the CAN\_SR register. If the TSTP mask in the CAN\_IMR register is set, an interrupt is generated while TSTP flag is set in the CAN\_SR register. This flag is cleared by reading the CAN\_SR register.

Note: The CAN\_TIMESTAMP register is reset when the CAN is disabled then enabled thanks to the CANEN bit in the CAN\_MR.

### 32.9.9 CAN Error Counter Register

**Name:** CAN\_ECR

**Address:** 0x40010020 (0), 0x40014020 (1)

**Access:** Read-only

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

- **REC: Receive Error Counter**

When a receiver detects an error, REC will be increased by one, except when the detected error is a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG.

When a receiver detects a dominant bit as the first bit after sending an ERROR FLAG, REC is increased by 8.

When a receiver detects a BIT ERROR while sending an ACTIVE ERROR FLAG, REC is increased by 8.

Any node tolerates up to 7 consecutive dominant bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive dominant bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive dominant bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive dominant bits, each receiver increases its REC by 8.

After successful reception of a message, REC is decreased by 1 if it was between 1 and 127. If REC was 0, it stays 0, and if it was greater than 127, then it is set to a value between 119 and 127.

- **TEC: Transmit Error Counter**

When a transmitter sends an ERROR FLAG, TEC is increased by 8 except when

- the transmitter is error passive and detects an ACKNOWLEDGMENT ERROR because of not detecting a dominant ACK and does not detect a dominant bit while sending its PASSIVE ERROR FLAG.
- the transmitter sends an ERROR FLAG because a STUFF ERROR occurred during arbitration and should have been recessive and has been sent as recessive but monitored as dominant.

When a transmitter detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG, the TEC will be increased by 8.

Any node tolerates up to 7 consecutive dominant bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive dominant bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive dominant bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive dominant bits every transmitter increases its TEC by 8.

After a successful transmission the TEC is decreased by 1 unless it was already 0.

### 32.9.10 CAN Transfer Command Register

**Name:** CAN\_TCR

**Address:** 0x40010024 (0), 0x40014024 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
TIMRST	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	—
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

This register initializes several transfer requests at the same time.

- **MBx: Transfer Request for Mailbox x**

Mailbox Object Type	Description
Receive	It receives the next message.
Receive with overwrite	This triggers a new reception.
Transmit	Sends data prepared in the mailbox as soon as possible.
Consumer	Sends a remote frame.
Producer	Sends data prepared in the mailbox after receiving a remote frame from a consumer.

This flag clears the MRDY and MABT flags in the corresponding CAN\_MSRx register.

When several mailboxes are requested to be transmitted simultaneously, they are transmitted in turn, starting with the mailbox with the highest priority. If several mailboxes have the same priority, then the mailbox with the lowest number is sent first (i.e., MB0 will be transferred before MB1).

- **TIMRST: Timer Reset**

Resets the internal timer counter. If the internal timer counter is frozen, this command automatically re-enables it. This command is useful in Time Triggered mode.

### 32.9.11 CAN Abort Command Register

**Name:** CAN\_ACR

**Address:** 0x40010028 (0), 0x40014028 (1)

**Access:** Write-only

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
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

This register initializes several abort requests at the same time.

- **MBx: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame is not serviced.

It is possible to set MACR field (in the CAN\_MCRx register) for each mailbox.

### 32.9.12 CAN Write Protection Mode Register

**Name:** CAN\_WPMR

**Address:** 0x400100E4 (0), 0x400140E4 (1)

**Access:** Read-write

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

- **WPEN: Write Protection Enable**

0: The Write Protection is Disabled

1: The Write Protection is Enabled

- **WPKEY: SPI Write Protection Key Password**

If a value is written in WPEN, the value is taken into account only if WPKEY is written with "CAN" (CAN written in ASCII Code, i.e. 0x43414E in hexadecimal).

Protects the registers:

- “CAN Mode Register”
- “CAN Baudrate Register”
- “CAN Message Mode Register”
- “CAN Message Acceptance Mask Register”
- “CAN Message ID Register”

### 32.9.13 CAN Write Protection Status Register

**Name:** CAN\_WPSR

**Address:** 0x400100E8 (0), 0x400140E8 (1)

**Access:** Read-only

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

- **WPVS: Write Protection Violation Status**

0 = No Write Protect Violation has occurred since the last read of the CAN\_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the CAN\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

This Field indicates the offset of the register concerned by the violation

### 32.9.14 CAN Message Mode Register

**Name:** CAN\_MMRx [x=0..7]

**Address:** 0x40010200 (0)[0], 0x40010220 (0)[1], 0x40010240 (0)[2], 0x40010260 (0)[3], 0x40010280 (0)[4],  
0x400102A0 (0)[5], 0x400102C0 (0)[6], 0x400102E0 (0)[7], 0x40014200 (1)[0], 0x40014220 (1)[1],  
0x40014240 (1)[2], 0x40014260 (1)[3], 0x40014280 (1)[4], 0x400142A0 (1)[5], 0x400142C0 (1)[6],  
0x400142E0 (1)[7]

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in ["CAN Write Protection Mode Register"](#).

- **MTIMEMARK: Mailbox Timemark**

This field is active in Time Triggered Mode. Transmit operations are allowed when the internal timer counter reaches the Mailbox Timemark. See ["Transmitting within a Time Window" on page 623](#).

In Timestamp Mode, MTIMEMARK is set to 0.

- **PRIOR: Mailbox Priority**

This field has no effect in receive and receive with overwrite modes. In these modes, the mailbox with the lowest number is serviced first.

When several mailboxes try to transmit a message at the same time, the mailbox with the highest priority is serviced first. If several mailboxes have the same priority, the mailbox with the lowest number is serviced first (i.e., MBx0 is serviced before MBx 15 if they have the same priority).

- **MOT: Mailbox Object Type**

This field allows the user to define the type of the mailbox. All mailboxes are independently configurable. Five different types are possible for each mailbox:

Value	Name	Description
0	MB_DISABLED	Mailbox is disabled. This prevents receiving or transmitting any messages with this mailbox.
1	MB_RX	Reception Mailbox. Mailbox is configured for reception. If a message is received while the mailbox data register is full, it is discarded.
2	MB_RX_OVERWRITE	Reception mailbox with overwrite. Mailbox is configured for reception. If a message is received while the mailbox is full, it overwrites the previous message.
3	MB_TX	Transmit mailbox. Mailbox is configured for transmission.
4	MB_CONSUMER	Consumer Mailbox. Mailbox is configured in reception but behaves as a Transmit Mailbox, i.e., it sends a remote frame and waits for an answer.
5	MB_PRODUCER	Producer Mailbox. Mailbox is configured in transmission but also behaves like a reception mailbox, i.e., it waits to receive a Remote Frame before sending its contents.
6	–	Reserved

### 32.9.15 CAN Message Acceptance Mask Register

**Name:** CAN\_MAMx [x=0..7]

**Address:** 0x40010204 (0)[0], 0x40010224 (0)[1], 0x40010244 (0)[2], 0x40010264 (0)[3], 0x40010284 (0)[4],  
0x400102A4 (0)[5], 0x400102C4 (0)[6], 0x400102E4 (0)[7], 0x40014204 (1)[0], 0x40014224 (1)[1],  
0x40014244 (1)[2], 0x40014264 (1)[3], 0x40014284 (1)[4], 0x400142A4 (1)[5], 0x400142C4 (1)[6],  
0x400142E4 (1)[7]

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in "["CAN Write Protection Mode Register"](#)".

To prevent concurrent access with the internal CAN core, the application must disable the mailbox before writing to CAN\_MAMx registers.

- **MIDvB: Complementary bits for identifier in extended frame mode**

Acceptance mask for corresponding field of the message IDvB register of the mailbox.

- **MIDvA: Identifier for standard frame mode**

Acceptance mask for corresponding field of the message IDvA register of the mailbox.

- **MIDE: Identifier Version**

0: Compares IDvA from the received frame with the CAN\_MIDx register masked with CAN\_MAMx register.

1: Compares IDvA and IDvB from the received frame with the CAN\_MIDx register masked with CAN\_MAMx register.

### 32.9.16 CAN Message ID Register

**Name:** CAN\_MIDx [x=0..7]

**Address:** 0x40010208 (0)[0], 0x40010228 (0)[1], 0x40010248 (0)[2], 0x40010268 (0)[3], 0x40010288 (0)[4],  
0x400102A8 (0)[5], 0x400102C8 (0)[6], 0x400102E8 (0)[7], 0x40014208 (1)[0], 0x40014228 (1)[1],  
0x40014248 (1)[2], 0x40014268 (1)[3], 0x40014288 (1)[4], 0x400142A8 (1)[5], 0x400142C8 (1)[6],  
0x400142E8 (1)[7]

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in "["CAN Write Protection Mode Register"](#)".

To prevent concurrent access with the internal CAN core, the application must disable the mailbox before writing to CAN\_MIDx registers.

- **MIDvB: Complementary bits for identifier in extended frame mode**

If MIDE is cleared, MIDvB value is 0.

- **MIDE: Identifier Version**

This bit allows the user to define the version of messages processed by the mailbox. If set, mailbox is dealing with version 2.0 Part B messages; otherwise, mailbox is dealing with version 2.0 Part A messages.

- **MIDvA: Identifier for standard frame mode**

### 32.9.17 CAN Message Family ID Register

**Name:** CAN\_MFIDx [x=0..7]

**Address:** 0x4001020C (0)[0], 0x4001022C (0)[1], 0x4001024C (0)[2], 0x4001026C (0)[3], 0x4001028C (0)[4],  
0x400102AC (0)[5], 0x400102CC (0)[6], 0x400102EC (0)[7], 0x4001420C (1)[0], 0x4001422C (1)[1],  
0x4001424C (1)[2], 0x4001426C (1)[3], 0x4001428C (1)[4], 0x400142AC (1)[5], 0x400142CC (1)[6],  
0x400142EC (1)[7]

**Access:** Read-only

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

- **MFID: Family ID**

This field contains the concatenation of CAN\_MIDx register bits masked by the CAN\_MAMx register. This field is useful to speed up message ID decoding. The message acceptance procedure is described below.

As an example:

```
CAN_MIDx = 0x305A4321  
CAN_MAMx = 0x3FF0F0FF  
CAN_MFIDx = 0x000000A3
```

### 32.9.18 CAN Message Status Register

**Name:** CAN\_MSRx [x=0..7]

**Address:** 0x40010210 (0)[0], 0x40010230 (0)[1], 0x40010250 (0)[2], 0x40010270 (0)[3], 0x40010290 (0)[4],  
0x400102B0 (0)[5], 0x400102D0 (0)[6], 0x400102F0 (0)[7], 0x40014210 (1)[0], 0x40014230 (1)[1],  
0x40014250 (1)[2], 0x40014270 (1)[3], 0x40014290 (1)[4], 0x400142B0 (1)[5], 0x400142D0 (1)[6],  
0x400142F0 (1)[7]

**Access:** Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	MMI
23	22	21	20	19	18	17	16
MRDY	MABT	—	MRTR	—	MDLC	—	—
15	14	13	12	11	10	9	8
MTIMESTAMP							
7	6	5	4	3	2	1	0
MTIMESTAMP							

These register fields are updated each time a message transfer is received or aborted.

MMI is cleared by reading the CAN\_MSRx register.

MRDY, MABT are cleared by writing MTCR or MACR in the CAN\_MCRx register.

**Warning:** MRTR and MDLC state depends partly on the mailbox object type.

- **MTIMESTAMP: Timer value**

This field is updated only when time-triggered operations are disabled (TTM cleared in CAN\_MR register). If the TEOF field in the CAN\_MR register is cleared, TIMESTAMP is the internal timer value at the start of frame of the last message received or sent by the mailbox. If the TEOF field in the CAN\_MR register is set, TIMESTAMP is the internal timer value at the end of frame of the last message received or sent by the mailbox.

In Time Triggered Mode, MTIMESTAMP is set to 0.

- **MDLC: Mailbox Data Length Code**

Mailbox Object Type	Description
Receive	Length of the first mailbox message received
Receive with overwrite	Length of the last mailbox message received
Transmit	No action
Consumer	Length of the mailbox message received
Producer	Length of the mailbox message to be sent after the remote frame reception

- **MRTR: Mailbox Remote Transmission Request**

Mailbox Object Type	Description
Receive	The first frame received has the RTR bit set.
Receive with overwrite	The last frame received has the RTR bit set.
Transmit	Reserved
Consumer	Reserved. After setting the MOT field in the CAN_MMR, MRTR is reset to 1.
Producer	Reserved. After setting the MOT field in the CAN_MMR, MRTR is reset to 0.

- **MABT: Mailbox Message Abort**

An interrupt is triggered when MABT is set.

0: Previous transfer is not aborted.

1: Previous transfer has been aborted.

This flag is cleared by writing to CAN\_MCRx register

Mailbox Object Type	Description
Receive	Reserved
Receive with overwrite	Reserved
Transmit	Previous transfer has been aborted
Consumer	The remote frame transfer request has been aborted.
Producer	The response to the remote frame transfer has been aborted.

- **MRDY: Mailbox Ready**

An interrupt is triggered when MRDY is set.

0: Mailbox data registers can not be read/written by the software application. CAN\_MDx are locked by the CAN\_MDx.

1: Mailbox data registers can be read/written by the software application.

This flag is cleared by writing to CAN\_MCRx register

Mailbox Object Type	Description
Receive	At least one message has been received since the last mailbox transfer order. Data from the first frame received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Receive with overwrite	At least one frame has been received since the last mailbox transfer order. Data from the last frame received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Transmit	Mailbox data have been transmitted. After setting the MOT field in the CAN_MMR, MRDY is reset to 1.
Consumer	At least one message has been received since the last mailbox transfer order. Data from the first message received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Producer	A remote frame has been received, mailbox data have been transmitted. After setting the MOT field in the CAN_MMR, MRDY is reset to 1.

- **MMI: Mailbox Message Ignored**

0: No message has been ignored during the previous transfer

1: At least one message has been ignored during the previous transfer

Cleared by reading the CAN\_MSRx register.

Mailbox Object Type	Description
Receive	Set when at least two messages intended for the mailbox have been sent. The first one is available in the mailbox data register. Others have been ignored. A mailbox with a lower priority may have accepted the message.
Receive with overwrite	Set when at least two messages intended for the mailbox have been sent. The last one is available in the mailbox data register. Previous ones have been lost.

Transmit	Reserved
Consumer	A remote frame has been sent by the mailbox but several messages have been received. The first one is available in the mailbox data register. Others have been ignored. Another mailbox with a lower priority may have accepted the message.
Producer	A remote frame has been received, but no data are available to be sent.

### 32.9.19 CAN Message Data Low Register

**Name:** CAN\_MDLx [x=0..7]

**Address:** 0x40010214 (0)[0], 0x40010234 (0)[1], 0x40010254 (0)[2], 0x40010274 (0)[3], 0x40010294 (0)[4],  
0x400102B4 (0)[5], 0x400102D4 (0)[6], 0x400102F4 (0)[7], 0x40014214 (1)[0], 0x40014234 (1)[1],  
0x40014254 (1)[2], 0x40014274 (1)[3], 0x40014294 (1)[4], 0x400142B4 (1)[5], 0x400142D4 (1)[6],  
0x400142F4 (1)[7]

**Access:** Read-write

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

- **MDL: Message Data Low Value**

When MRDY field is set in the CAN\_MSRx register, the lower 32 bits of a received message can be read or written by the software application. Otherwise, the MDL value is locked by the CAN controller to send/receive a new message.

In Receive with overwrite, the CAN controller may modify MDL value while the software application reads MDH and MDL registers. To check that MDH and MDL do not belong to different messages, the application has to check the MMI field in the CAN\_MSRx register. In this mode, the software application must re-read CAN\_MDH and CAN\_MDL, while the MMI bit in the CAN\_MSRx register is set.

Bytes are received/sent on the bus in the following order:

1. CAN\_MDL[7:0]
2. CAN\_MDL[15:8]
3. CAN\_MDL[23:16]
4. CAN\_MDL[31:24]
5. CAN\_MDH[7:0]
6. CAN\_MDH[15:8]
7. CAN\_MDH[23:16]
8. CAN\_MDH[31:24]

### 32.9.20 CAN Message Data High Register

**Name:** CAN\_MDHx [x=0..7]

**Address:** 0x40010218 (0)[0], 0x40010238 (0)[1], 0x40010258 (0)[2], 0x40010278 (0)[3], 0x40010298 (0)[4],  
0x400102B8 (0)[5], 0x400102D8 (0)[6], 0x400102F8 (0)[7], 0x40014218 (1)[0], 0x40014238 (1)[1],  
0x40014258 (1)[2], 0x40014278 (1)[3], 0x40014298 (1)[4], 0x400142B8 (1)[5], 0x400142D8 (1)[6],  
0x400142F8 (1)[7]

**Access:** Read-write

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

- **MDH: Message Data High Value**

When MRDY field is set in the CAN\_MSRx register, the upper 32 bits of a received message are read or written by the software application. Otherwise, the MDH value is locked by the CAN controller to send/receive a new message.

In Receive with overwrite, the CAN controller may modify MDH value while the software application reads MDH and MDL registers. To check that MDH and MDL do not belong to different messages, the application has to check the MMI field in the CAN\_MSRx register. In this mode, the software application must re-read CAN\_MDH and CAN\_MDL, while the MMI bit in the CAN\_MSRx register is set.

Bytes are received/sent on the bus in the following order:

1. CAN\_MDL[7:0]
2. CAN\_MDL[15:8]
3. CAN\_MDL[23:16]
4. CAN\_MDL[31:24]
5. CAN\_MDH[7:0]
6. CAN\_MDH[15:8]
7. CAN\_MDH[23:16]
8. CAN\_MDH[31:24]

### 32.9.21 CAN Message Control Register

**Name:** CAN\_MCRx [x=0..7]

**Address:** 0x4001021C (0)[0], 0x4001023C (0)[1], 0x4001025C (0)[2], 0x4001027C (0)[3], 0x4001029C (0)[4],  
0x400102BC (0)[5], 0x400102DC (0)[6], 0x400102FC (0)[7], 0x4001421C (1)[0], 0x4001423C (1)[1],  
0x4001425C (1)[2], 0x4001427C (1)[3], 0x4001429C (1)[4], 0x400142BC (1)[5], 0x400142DC (1)[6],  
0x400142FC (1)[7]

**Access:** Write-only

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

- **MDLC: Mailbox Data Length Code**

Mailbox Object Type	Description
Receive	No action.
Receive with overwrite	No action.
Transmit	Length of the mailbox message.
Consumer	No action.
Producer	Length of the mailbox message to be sent after the remote frame reception.

- **MRTR: Mailbox Remote Transmission Request**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Set the RTR bit in the sent frame
Consumer	No action, the RTR bit in the sent frame is set automatically
Producer	No action

Consumer situations can be handled automatically by setting the mailbox object type in Consumer. This requires only one mailbox.

It can also be handled using two mailboxes, one in reception, the other in transmission. The MRTR and the MTCR bits must be set in the same time.

- **MACR: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame will not be serviced.

It is possible to set MACR field for several mailboxes in the same time, setting several bits to the CAN\_ACR register.

- **MTCR: Mailbox Transfer Command**

Mailbox Object Type	Description
Receive	Allows the reception of the next message.
Receive with overwrite	Triggers a new reception.
Transmit	Sends data prepared in the mailbox as soon as possible.
Consumer	Sends a remote transmission frame.
Producer	Sends data prepared in the mailbox after receiving a remote frame from a Consumer.

This flag clears the MRDY and MABT flags in the CAN\_MSRx register.

When several mailboxes are requested to be transmitted simultaneously, they are transmitted in turn. The mailbox with the highest priority is serviced first. If several mailboxes have the same priority, the mailbox with the lowest number is serviced first (i.e., MBx0 will be serviced before MBx 15 if they have the same priority).

It is possible to set MTCR for several mailboxes at the same time by writing to the CAN\_TCR register.



## 33. Parallel Input/Output (PIO) Controller

### 33.1 Description

The Parallel Input/Output Controller (PIO) manages up to 32 fully programmable input/output lines. Each I/O line may be dedicated as a general-purpose I/O or be assigned to a function of an embedded peripheral. This assures effective optimization of the pins of a product.

Each I/O line is associated with a bit number in all of the 32-bit registers of the 32-bit wide User Interface.

Each I/O line of the PIO Controller features:

- An input change interrupt enabling level change detection on any I/O line.
- Additional Interrupt modes enabling rising edge, falling edge, low level or high level detection on any I/O line.
- A glitch filter providing rejection of glitches lower than one-half of PIO clock cycle.
- A debouncing filter providing rejection of unwanted pulses from key or push button operations.
- Multi-drive capability similar to an open drain I/O line.
- Control of the pull-up and pull-down of the I/O line.
- Input visibility and output control.

The PIO Controller also features a synchronous output providing up to 32 bits of data output in a single write operation.

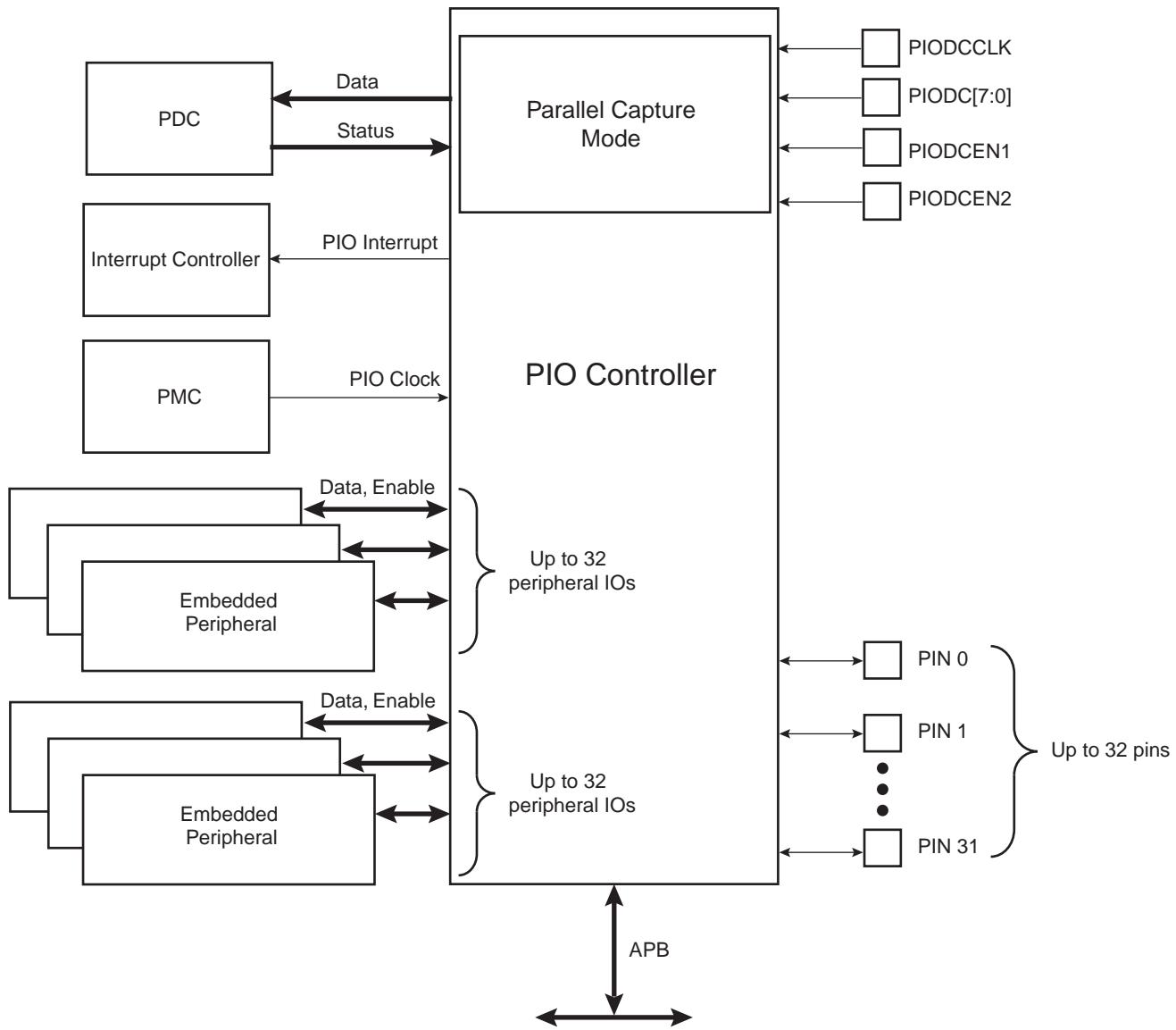
An 8-bit parallel capture mode is also available which can be used to interface a CMOS digital image sensor, an ADC, a DSP synchronous port in synchronous mode, etc.

### 33.2 Embedded Characteristics

- Up to 32 Programmable I/O Lines
- Fully Programmable through Set/Clear Registers
- Multiplexing of Four Peripheral Functions per I/O Line
- For each I/O Line (Whether Assigned to a Peripheral or Used as General Purpose I/O)
  - Input Change Interrupt
  - Programmable Glitch Filter
  - Programmable Debouncing Filter
  - Multi-drive Option Enables Driving in Open Drain
  - Programmable Pull Up on Each I/O Line
  - Pin Data Status Register, Supplies Visibility of the Level on the Pin at Any Time
  - Additional Interrupt Modes on a Programmable Event: Rising Edge, Falling Edge, Low Level or High Level
  - Lock of the Configuration by the Connected Peripheral
- Synchronous Output, Provides Set and Clear of Several I/O lines in a Single Write
- Write Protect Registers
- Programmable Schmitt Trigger Inputs
- Programmable I/O Delay
- Parallel Capture Mode
  - Can be used to interface a CMOS digital image sensor, an ADC....
  - One Clock, 8-bit Parallel Data and Two Data Enable on I/O Lines
  - Data Can be Sampled one time of out two (For Chrominance Sampling Only)
  - Supports Connection of one Peripheral DMA Controller Channel (PDC) Which Offers Buffer Reception Without Processor Intervention

### 33.3 Block Diagram

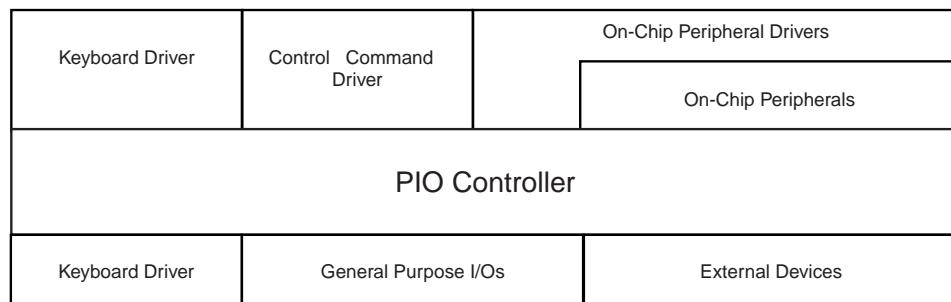
**Figure 33-1. Block Diagram**



**Table 33-1. Signal Description**

Signal Name	Signal Description	Signal Type
PIODCCLK	Parallel Capture Mode Clock	Input
PIODC[7:0]	Parallel Capture Mode Data	Input
PIODCEN1	Parallel Capture Mode Data Enable 1	Input
PIODCEN2	Parallel Capture Mode Data Enable 2	Input

**Figure 33-2. Application Block Diagram**



## 33.4 Product Dependencies

### 33.4.1 Pin Multiplexing

Each pin is configurable, according to product definition as either a general-purpose I/O line only, or as an I/O line multiplexed with one or two peripheral I/Os. As the multiplexing is hardware defined and thus product-dependent, the hardware designer and programmer must carefully determine the configuration of the PIO controllers required by their application. When an I/O line is general-purpose only, i.e. not multiplexed with any peripheral I/O, programming of the PIO Controller regarding the assignment to a peripheral has no effect and only the PIO Controller can control how the pin is driven by the product.

### 33.4.2 External Interrupt Lines

The interrupt signals FIQ and IRQ0 to IRQn are most generally multiplexed through the PIO Controllers. However, it is not necessary to assign the I/O line to the interrupt function as the PIO Controller has no effect on inputs and the interrupt lines (FIQ or IRQs) are used only as inputs.

### 33.4.3 Power Management

The Power Management Controller controls the PIO Controller clock in order to save power. Writing any of the registers of the user interface does not require the PIO Controller clock to be enabled. This means that the configuration of the I/O lines does not require the PIO Controller clock to be enabled.

However, when the clock is disabled, not all of the features of the PIO Controller are available, including glitch filtering. Note that the Input Change Interrupt, Interrupt Modes on a programmable event and the read of the pin level require the clock to be validated.

After a hardware reset, the PIO clock is disabled by default.

The user must configure the Power Management Controller before any access to the input line information.

### 33.4.4 Interrupt Generation

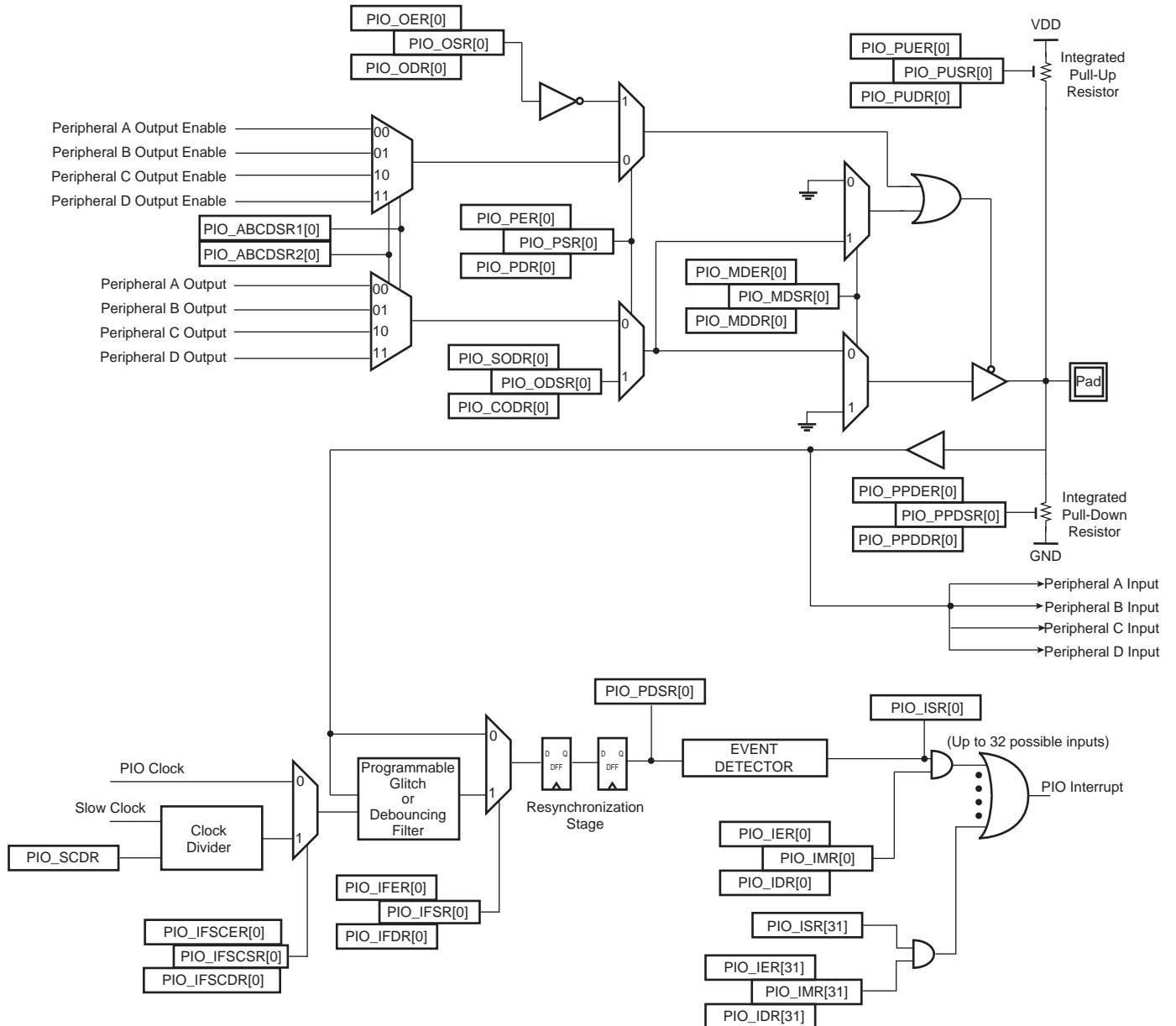
For interrupt handling, the PIO Controllers are considered as user peripherals. This means that the PIO Controller interrupt lines are connected among the interrupt sources. Refer to the PIO Controller peripheral identifier in the product description to identify the interrupt sources dedicated to the PIO Controllers. Using the PIO Controller requires the Interrupt Controller to be programmed first.

The PIO Controller interrupt can be generated only if the PIO Controller clock is enabled.

### 33.5 Functional Description

The PIO Controller features up to 32 fully-programmable I/O lines. Most of the control logic associated to each I/O is represented in [Figure 33-3](#). In this description each signal shown represents but one of up to 32 possible indexes.

**Figure 33-3. I/O Line Control Logic**



### 33.5.1 Pull-up and Pull-down Resistor Control

Each I/O line is designed with an embedded pull-up resistor and an embedded pull-down resistor. The pull-up resistor can be enabled or disabled by writing respectively PIO\_PUER (Pull-up Enable Register) and PIO\_PUDR (Pull-up Disable Register). Writing in these registers results in setting or clearing the corresponding bit in PIO\_PUSR (Pull-up Status Register). Reading a 1 in PIO\_PUSR means the pull-up is disabled and reading a 0 means the pull-up is enabled. The pull-down resistor can be enabled or disabled by writing respectively PIO\_PPDER (Pull-down Enable Register) and PIO\_PPDDR (Pull-down Disable Register). Writing in these registers results in setting or clearing the corresponding bit in PIO\_PPDSR (Pull-down Status Register). Reading a 1 in PIO\_PPDSR means the pull-up is disabled and reading a 0 means the pull-down is enabled.

Enabling the pull-down resistor while the pull-up resistor is still enabled is not possible. In this case, the write of PIO\_PPDER for the concerned I/O line is discarded. Likewise, enabling the pull-up resistor while the pull-down resistor is still enabled is not possible. In this case, the write of PIO\_PUER for the concerned I/O line is discarded.

Control of the pull-up resistor is possible regardless of the configuration of the I/O line.

After reset, all of the pull-ups are enabled, i.e. PIO\_PUSR resets at the value 0x0, and all the pull-downs are disabled, i.e. PIO\_PPDSR resets at the value 0xFFFFFFFF.

### 33.5.2 I/O Line or Peripheral Function Selection

When a pin is multiplexed with one or two peripheral functions, the selection is controlled with the registers PIO\_PER (PIO Enable Register) and PIO\_PDR (PIO Disable Register). The register PIO\_PSR (PIO Status Register) is the result of the set and clear registers and indicates whether the pin is controlled by the corresponding peripheral or by the PIO Controller. A value of 0 indicates that the pin is controlled by the corresponding on-chip peripheral selected in the PIO\_ABCDSR1 and PIO\_ABCDSR2 (ABCD Select Registers). A value of 1 indicates the pin is controlled by the PIO controller.

If a pin is used as a general purpose I/O line (not multiplexed with an on-chip peripheral), PIO\_PER and PIO\_PDR have no effect and PIO\_PSR returns 1 for the corresponding bit.

After reset, most generally, the I/O lines are controlled by the PIO controller, i.e. PIO\_PSR resets at 1. However, in some events, it is important that PIO lines are controlled by the peripheral (as in the case of memory chip select lines that must be driven inactive after reset or for address lines that must be driven low for booting out of an external memory). Thus, the reset value of PIO\_PSR is defined at the product level, depending on the multiplexing of the device.

### 33.5.3 Peripheral A or B or C or D Selection

The PIO Controller provides multiplexing of up to four peripheral functions on a single pin. The selection is performed by writing PIO\_ABCDSR1 and PIO\_ABCDSR2 (ABCD Select Registers).

For each pin:

- The corresponding bit at level 0 in PIO\_ABCDSR1 and the corresponding bit at level 0 in PIO\_ABCDSR2 means peripheral A is selected.
- The corresponding bit at level 1 in PIO\_ABCDSR1 and the corresponding bit at level 0 in PIO\_ABCDSR2 means peripheral B is selected.
- The corresponding bit at level 0 in PIO\_ABCDSR1 and the corresponding bit at level 1 in PIO\_ABCDSR2 means peripheral C is selected.
- The corresponding bit at level 1 in PIO\_ABCDSR1 and the corresponding bit at level 1 in PIO\_ABCDSR2 means peripheral D is selected.

Note that multiplexing of peripheral lines A, B, C and D only affects the output line. The peripheral input lines are always connected to the pin input.

Writing in PIO\_ABCDSR1 and PIO\_ABCDSR2 manages the multiplexing regardless of the configuration of the pin. However, assignment of a pin to a peripheral function requires a write in the peripheral selection registers (PIO\_ABCDSR1 and PIO\_ABCDSR2) in addition to a write in PIO\_PDR.

After reset, PIO\_ABCDSR1 and PIO\_ABCDSR2 are 0, thus indicating that all the PIO lines are configured on peripheral A. However, peripheral A generally does not drive the pin as the PIO Controller resets in I/O line mode.

#### 33.5.4 Output Control

When the I/O line is assigned to a peripheral function, i.e. the corresponding bit in PIO\_PSR is at 0, the drive of the I/O line is controlled by the peripheral. Peripheral A or B or C or D depending on the value in PIO\_ABCDSR1 and PIO\_ABCDSR2 (ABCD Select Registers) determines whether the pin is driven or not.

When the I/O line is controlled by the PIO controller, the pin can be configured to be driven. This is done by writing PIO\_OER (Output Enable Register) and PIO\_ODR (Output Disable Register). The results of these write operations are detected in PIO\_OSR (Output Status Register). When a bit in this register is at 0, the corresponding I/O line is used as an input only. When the bit is at 1, the corresponding I/O line is driven by the PIO controller.

The level driven on an I/O line can be determined by writing in PIO\_SODR (Set Output Data Register) and PIO\_CODR (Clear Output Data Register). These write operations respectively set and clear PIO\_ODSR (Output Data Status Register), which represents the data driven on the I/O lines. Writing in PIO\_OER and PIO\_ODR manages PIO\_OSR whether the pin is configured to be controlled by the PIO controller or assigned to a peripheral function. This enables configuration of the I/O line prior to setting it to be managed by the PIO Controller.

Similarly, writing in PIO\_SODR and PIO\_CODR effects PIO\_ODSR. This is important as it defines the first level driven on the I/O line.

#### 33.5.5 Synchronous Data Output

Clearing one (or more) PIO line(s) and setting another one (or more) PIO line(s) synchronously cannot be done by using PIO\_SODR and PIO\_CODR registers. It requires two successive write operations into two different registers. To overcome this, the PIO Controller offers a direct control of PIO outputs by single write access to PIO\_ODSR (Output Data Status Register). Only bits unmasked by PIO\_OWSR (Output Write Status Register) are written. The mask bits in PIO\_OWSR are set by writing to PIO\_OWER (Output Write Enable Register) and cleared by writing to PIO\_OWDR (Output Write Disable Register).

After reset, the synchronous data output is disabled on all the I/O lines as PIO\_OWSR resets at 0x0.

#### 33.5.6 Multi Drive Control (Open Drain)

Each I/O can be independently programmed in Open Drain by using the Multi Drive feature. This feature permits several drivers to be connected on the I/O line which is driven low only by each device. An external pull-up resistor (or enabling of the internal one) is generally required to guarantee a high level on the line.

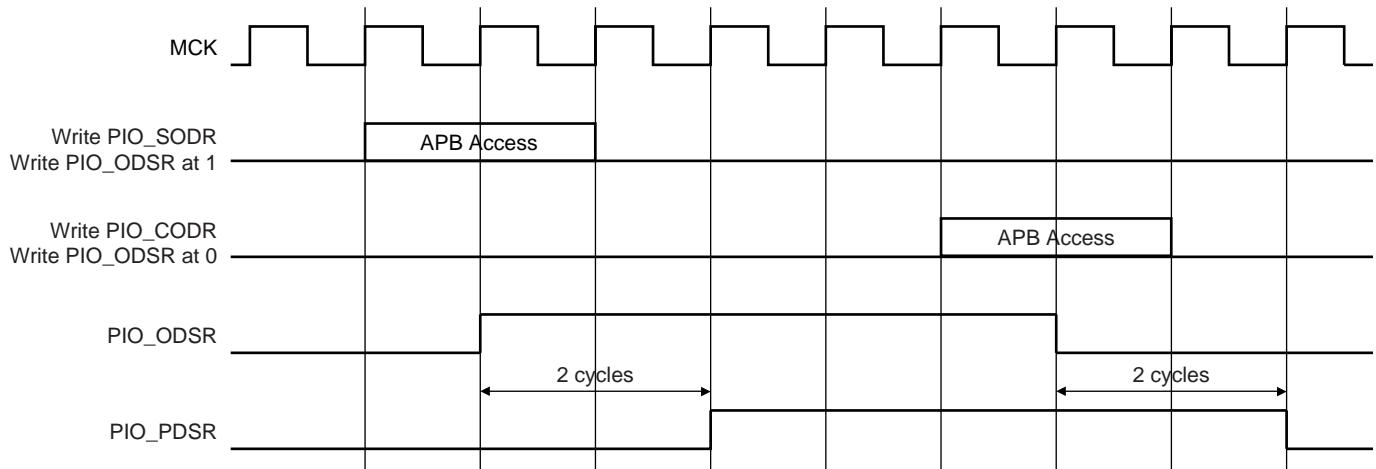
The Multi Drive feature is controlled by PIO\_MDER (Multi-driver Enable Register) and PIO\_MDDR (Multi-driver Disable Register). The Multi Drive can be selected whether the I/O line is controlled by the PIO controller or assigned to a peripheral function. PIO\_MDSR (Multi-driver Status Register) indicates the pins that are configured to support external drivers.

After reset, the Multi Drive feature is disabled on all pins, i.e. PIO\_MDSR resets at value 0x0.

#### 33.5.7 Output Line Timings

[Figure 33-4](#) shows how the outputs are driven either by writing PIO\_SODR or PIO\_CODR, or by directly writing PIO\_ODSR. This last case is valid only if the corresponding bit in PIO\_OWSR is set. [Figure 33-4](#) also shows when the feedback in PIO\_PDSR is available.

**Figure 33-4. Output Line Timings**



### 33.5.8 Inputs

The level on each I/O line can be read through PIO\_PDSR (Pin Data Status Register). This register indicates the level of the I/O lines regardless of their configuration, whether uniquely as an input or driven by the PIO controller or driven by a peripheral.

Reading the I/O line levels requires the clock of the PIO controller to be enabled, otherwise PIO\_PDSR reads the levels present on the I/O line at the time the clock was disabled.

### 33.5.9 Input Glitch and Debouncing Filters

Optional input glitch and debouncing filters are independently programmable on each I/O line.

The glitch filter can filter a glitch with a duration of less than 1/2 Master Clock (MCK) and the debouncing filter can filter a pulse of less than 1/2 Period of a Programmable Divided Slow Clock.

The selection between glitch filtering or debounce filtering is done by writing in the registers PIO\_IFSCDR (PIO Input Filter Slow Clock Disable Register) and PIO\_IFSCER (PIO Input Filter Slow Clock Enable Register). Writing PIO\_IFSCDR and PIO\_IFSCER respectively, sets and clears bits in PIO\_IFCSR.

The current selection status can be checked by reading the register PIO\_IFCSR (Input Filter Slow Clock Status Register).

- If  $\text{PIO\_IFCSR}[i] = 0$ : The glitch filter can filter a glitch with a duration of less than 1/2 Period of Master Clock.
- If  $\text{PIO\_IFCSR}[i] = 1$ : The debouncing filter can filter a pulse with a duration of less than 1/2 Period of the Programmable Divided Slow Clock.

For the debouncing filter, the Period of the Divided Slow Clock is performed by writing in the DIV field of the PIO\_SCDR (Slow Clock Divider Register)

$$T_{div\_slclk} = ((\text{DIV}+1)*2) \cdot T_{slow\_clock}$$

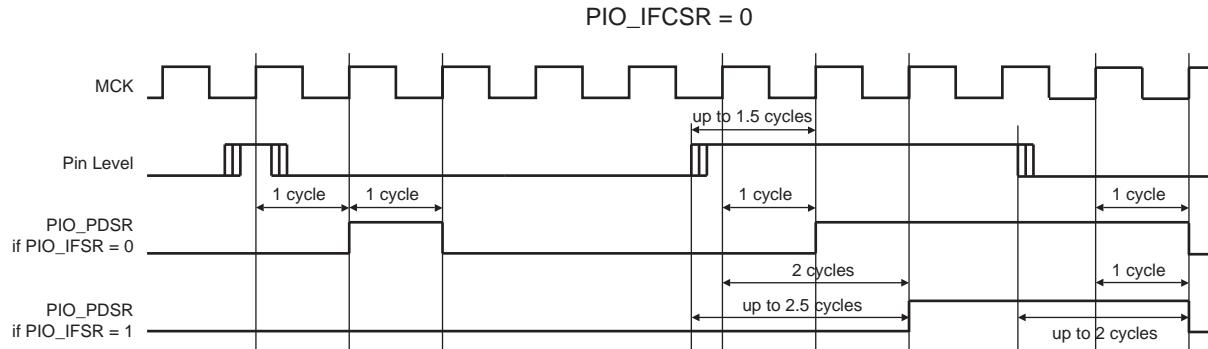
When the glitch or debouncing filter is enabled, a glitch or pulse with a duration of less than 1/2 Selected Clock Cycle (Selected Clock represents MCK or Divided Slow Clock depending on PIO\_IFSCDR and PIO\_IFSCER programming) is automatically rejected, while a pulse with a duration of 1 Selected Clock (MCK or Divided Slow Clock) cycle or more is accepted. For pulse durations between 1/2 Selected Clock cycle and 1 Selected Clock cycle the pulse may or may not be taken into account, depending on the precise timing of its occurrence. Thus for a pulse to be visible it must exceed 1 Selected Clock cycle, whereas for a glitch to be reliably filtered out, its duration must not exceed 1/2 Selected Clock cycle.

The filters also introduce some latencies, this is illustrated in [Figure 33-5](#) and [Figure 33-6](#).

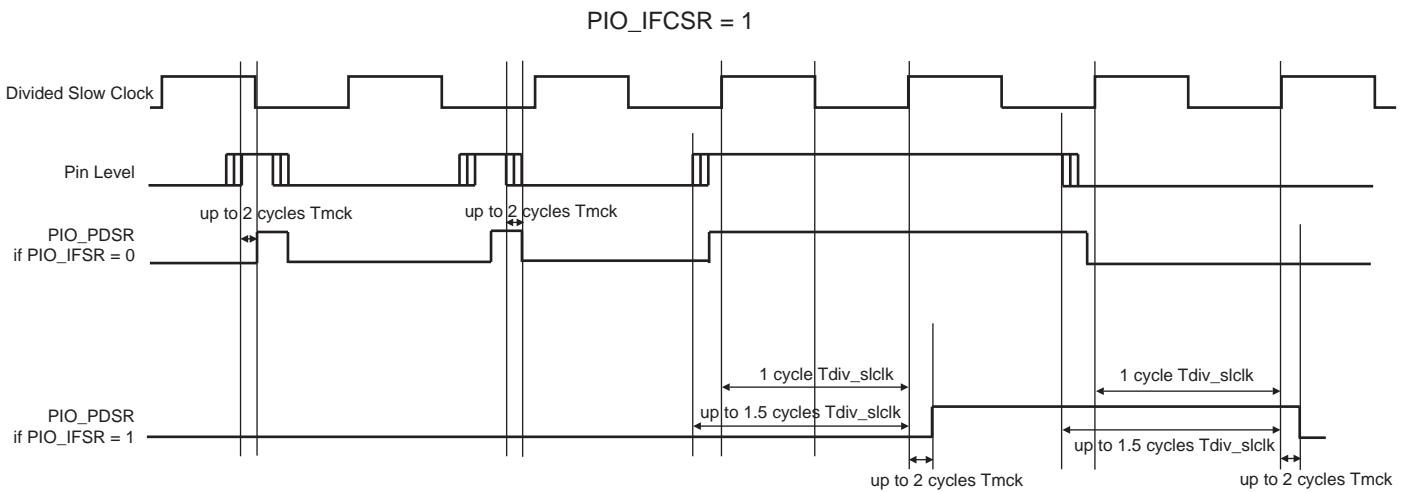
The glitch filters are controlled by the register set: PIO\_IFER (Input Filter Enable Register), PIO\_IFDR (Input Filter Disable Register) and PIO\_IFSR (Input Filter Status Register). Writing PIO\_IFER and PIO\_IFDR respectively sets and clears bits in PIO\_IFSR. This last register enables the glitch filter on the I/O lines.

When the glitch and/or debouncing filter is enabled, it does not modify the behavior of the inputs on the peripherals. It acts only on the value read in PIO\_PDSR and on the input change interrupt detection. The glitch and debouncing filters require that the PIO Controller clock is enabled.

**Figure 33-5. Input Glitch Filter Timing**



**Figure 33-6. Input Debouncing Filter Timing**



### 33.5.10 Input Edge/Level Interrupt

The PIO Controller can be programmed to generate an interrupt when it detects an edge or a level on an I/O line. The Input Edge/Level Interrupt is controlled by writing PIO\_IER (Interrupt Enable Register) and PIO\_IDR (Interrupt Disable Register), which respectively enable and disable the input change interrupt by setting and clearing the corresponding bit in PIO\_IMR (Interrupt Mask Register). As Input change detection is possible only by comparing two successive samplings of the input of the I/O line, the PIO Controller clock must be enabled. The Input Change Interrupt is available, regardless of the configuration of the I/O line, i.e. configured as an input only, controlled by the PIO Controller or assigned to a peripheral function.

By default, the interrupt can be generated at any time an edge is detected on the input.

Some additional Interrupt modes can be enabled/disabled by writing in the PIO\_AIMER (Additional Interrupt Modes Enable Register) and PIO\_AMDR (Additional Interrupt Modes Disable Register). The current state of this selection can be read through the PIO\_AMMR (Additional Interrupt Modes Mask Register)

These Additional Modes are:

- Rising Edge Detection
- Falling Edge Detection
- Low Level Detection
- High Level Detection

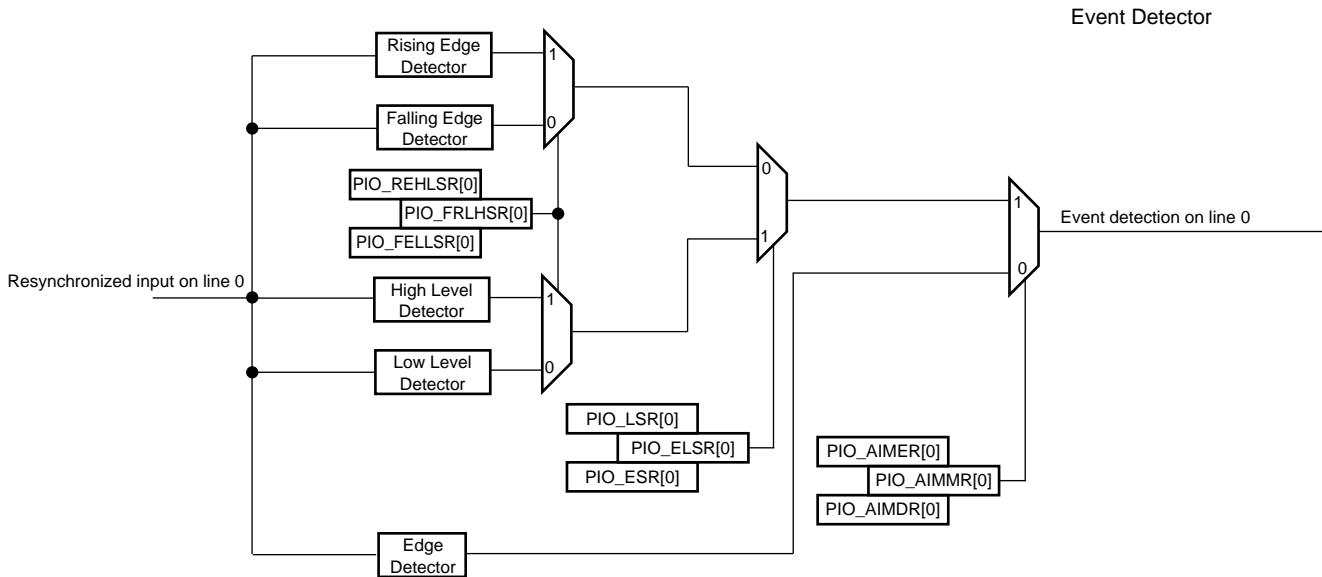
In order to select an Additional Interrupt Mode:

- The type of event detection (Edge or Level) must be selected by writing in the set of registers; PIO\_ESR (Edge Select Register) and PIO\_LSR (Level Select Register) which enable respectively, the Edge and Level Detection. The current status of this selection is accessible through the PIO\_ELSR (Edge/Level Status Register).
- The Polarity of the event detection (Rising/Falling Edge or High/Low Level) must be selected by writing in the set of registers; PIO\_FELLSR (Falling Edge /Low Level Select Register) and PIO\_REHLSR (Rising Edge/High Level Select Register) which allow to select Falling or Rising Edge (if Edge is selected in the PIO\_ELSR) Edge or High or Low Level Detection (if Level is selected in the PIO\_ELSR). The current status of this selection is accessible through the PIO\_FRLHSR (Fall/Rise - Low/High Status Register).

When an input Edge or Level is detected on an I/O line, the corresponding bit in PIO\_ISR (Interrupt Status Register) is set. If the corresponding bit in PIO\_IMR is set, the PIO Controller interrupt line is asserted. The interrupt signals of the thirty-two channels are ORed-wired together to generate a single interrupt signal to the interrupt controller.

When the software reads PIO\_ISR, all the interrupts are automatically cleared. This signifies that all the interrupts that are pending when PIO\_ISR is read must be handled. When an Interrupt is enabled on a "Level", the interrupt is generated as long as the interrupt source is not cleared, even if some read accesses in PIO\_ISR are performed.

**Figure 33-7. Event Detector on Input Lines (Figure represents line 0)**



### 33.5.10.1 Example

If generating an interrupt is required on the following:

- Rising edge on PIO line 0
- Falling edge on PIO line 1
- Rising edge on PIO line 2
- Low Level on PIO line 3
- High Level on PIO line 4
- High Level on PIO line 5
- Falling edge on PIO line 6
- Rising edge on PIO line 7
- Any edge on the other lines

The configuration required is described below.

### 33.5.10.2 Interrupt Mode Configuration

All the interrupt sources are enabled by writing 32'hFFFF\_FFFF in PIO\_IER.

Then the Additional Interrupt Mode is enabled for line 0 to 7 by writing 32'h0000\_00FF in PIO\_AIMER.

### 33.5.10.3 Edge or Level Detection Configuration

Lines 3, 4 and 5 are configured in Level detection by writing 32'h0000\_0038 in PIO\_LSR.

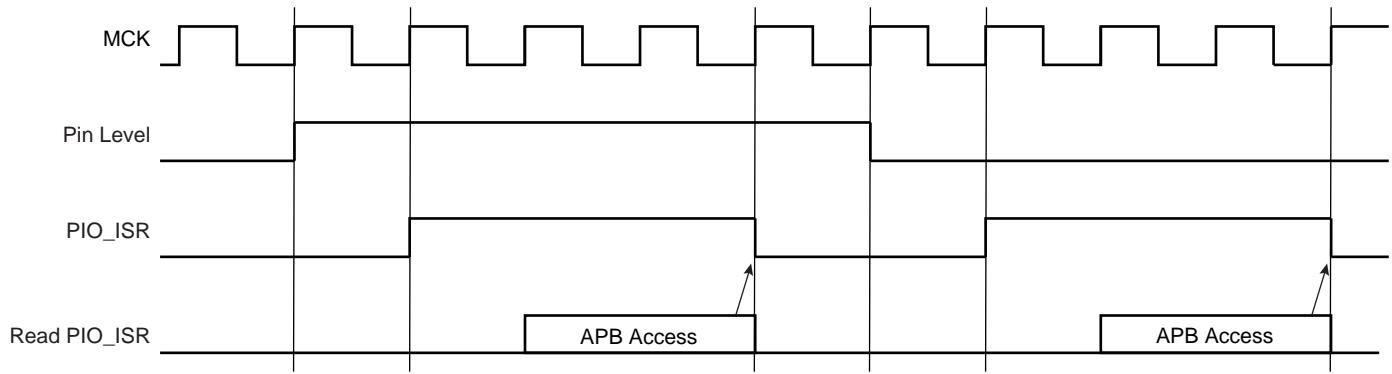
The other lines are configured in Edge detection by default, if they have not been previously configured. Otherwise, lines 0, 1, 2, 6 and 7 must be configured in Edge detection by writing 32'h0000\_00C7 in PIO\_ESR.

### 33.5.10.4 Falling/Rising Edge or Low/High Level Detection Configuration.

Lines 0, 2, 4, 5 and 7 are configured in Rising Edge or High Level detection by writing 32'h0000\_00B5 in PIO\_REHLSR.

The other lines are configured in Falling Edge or Low Level detection by default, if they have not been previously configured. Otherwise, lines 1, 3 and 6 must be configured in Falling Edge/Low Level detection by writing 32'h0000\_004A in PIO\_FELLSR.

**Figure 33-8. Input Change Interrupt Timings if there are no Additional Interrupt Modes**



**Figure 33-9.**

### 33.5.11 I/O Lines Lock

When an I/O line is controlled by a peripheral (particularly the Pulse Width Modulation Controller PWM), it can become locked by the action of this peripheral via an input of the PIO controller. When an I/O line is locked, the write of the corresponding bit in the registers PIO\_PER, PIO\_PDR, PIO\_MDER, PIO\_MDDR, PIO\_PUDR, PIO\_PUER, PIO\_ABCDSR1 and PIO\_ABCDSR2 is discarded in order to lock its configuration. The user can know at anytime which I/O line is locked by reading the PIO Lock Status register PIO\_LOCKSR. Once an I/O line is locked, the only way to unlock it is to apply a hardware reset to the PIO Controller.

### 33.5.12 Programmable I/O Delays

The PIO interface consists of a series of signals driven by peripherals or directly by software. The simultaneous switching outputs on these busses may lead to a peak of current in the internal and external power supply lines.

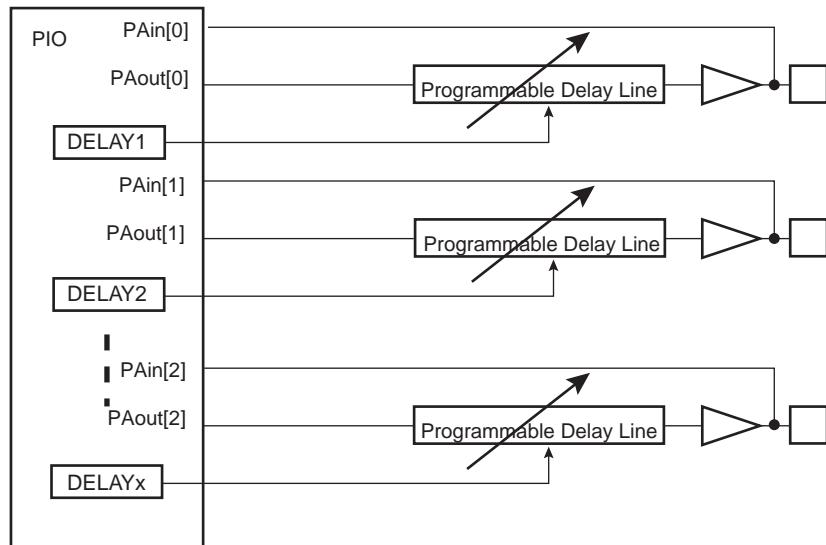
In order to reduce the current peak in such cases, additional propagation delays can be adjusted independently for pad buffers by means of configuration registers, PIO\_DELAY.

The additional programmable delays for each supporting range from 0 to - ns (Worst Case PVT). The delay can differ between I/Os supporting this feature. Delay can be modified per programming for each I/O. The minimal additional delay that can be programmed on a PAD supporting this feature is 1/16 of the maximum programmable delay.

Only PADs PA26-PA27-PA30-PA31 can be configured.

When programming 0x0 in fields, no delay is added (reset value) and the propagation delay of the pad buffers is the inherent delay of the pad buffer. When programming 0xF in fields, the propagation delay of the corresponding pad is maximal.

**Figure 33-10. Programmable I/O Delays**



### 33.5.13 Programmable Schmitt Trigger

It is possible to configure each input for the Schmitt Trigger. By default the Schmitt trigger is active. Disabling the Schmitt Trigger is requested when using the QTouch™ Library.

### 33.5.14 Parallel Capture Mode

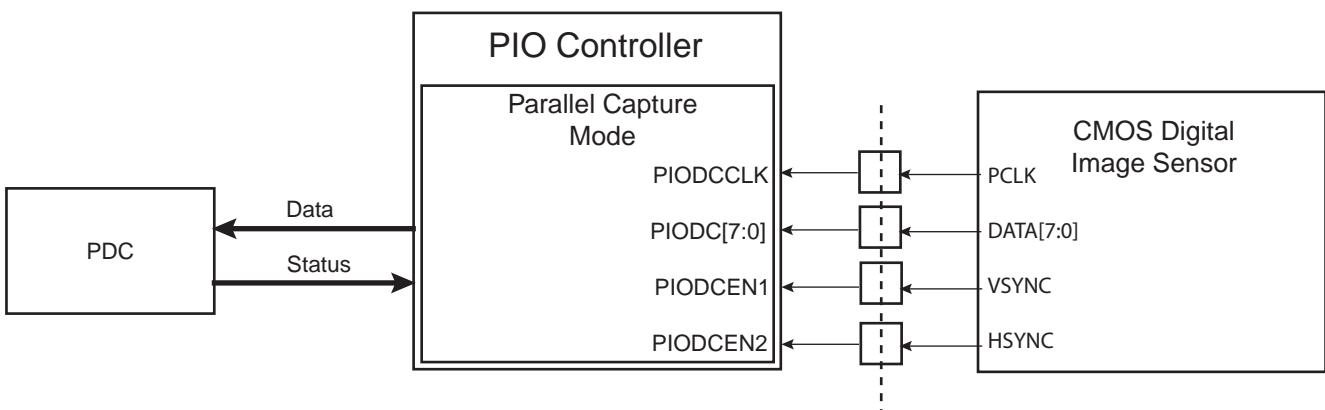
#### 33.5.14.1 Overview

The PIO Controller integrates an interface able to read data from a CMOS digital image sensor, a high-speed parallel ADC, a DSP synchronous port in synchronous mode, etc.... For better understanding and to ease reading, the following description uses an example with a CMOS digital image sensor.

#### 33.5.14.2 Functional Description

The CMOS digital image sensor provides a sensor clock, an 8-bit data synchronous with the sensor clock, and two data enables which are synchronous with the sensor clock too.

**Figure 33-11. PIO controller connection with CMOS digital image sensor**



As soon as the parallel capture mode is enabled by writing the PCEN bit at 1 in [PIO\\_PCMR \("PIO Parallel Capture Mode Register"\)](#), the I/O lines connected to the sensor clock (PIODCCLK), the sensor data (PIODC[7:0]) and the sensor data enable signals (PIODCEN1 and PIODCEN2) are configured automatically as INPUTS. To know which I/O lines are

associated with the sensor clock, the sensor data and the sensor data enable signals, refer to the I/O multiplexing table(s) in the product datasheet.

Once it is enabled, the parallel capture mode samples the data at rising edge of the sensor clock and resynchronizes it with the PIO clock domain.

The size of the data which can be read in PIO\_PCRHR ("[PIO Parallel Capture Reception Holding Register](#)") can be programmed thanks to the DSIZE field in PIO\_PCMR. If this data size is larger than 8 bits, then the parallel capture mode samples several sensor data to form a concatenated data of size defined by DSIZE. Then this data is stored in PIO\_PCRHR and the flag DRDY is set to 1 in PIO\_PCISR ("[PIO Parallel Capture Interrupt Status Register](#)").

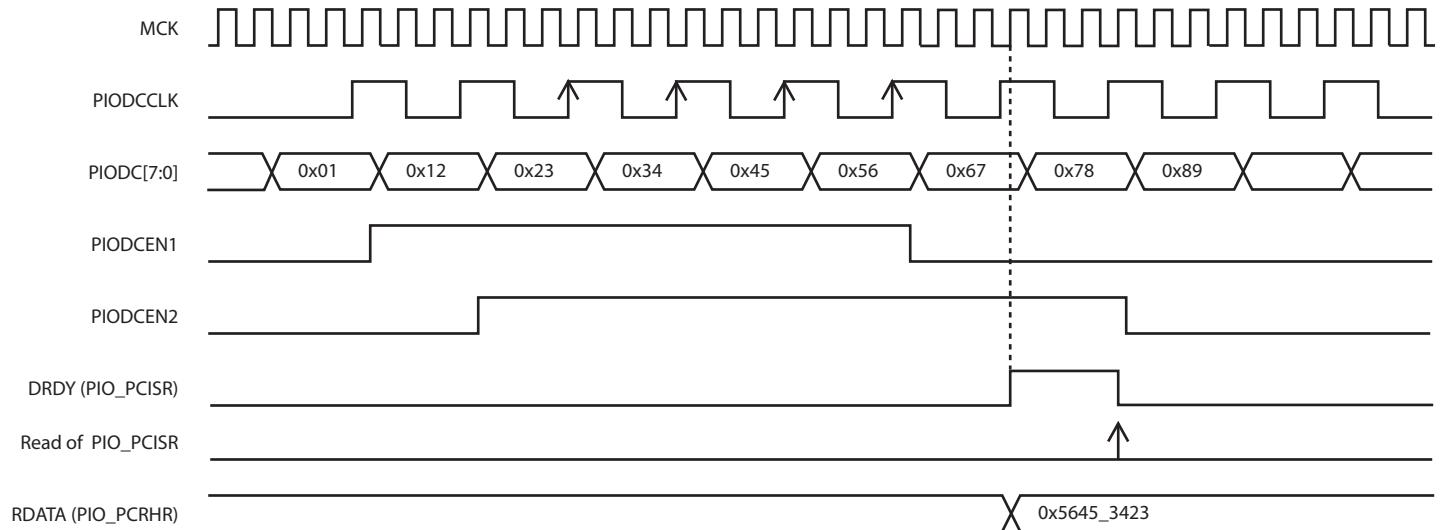
The parallel capture mode can be associated with a reception channel of the Peripheral DMA Controller (PDC). This enables performing reception transfer from parallel capture mode to a memory buffer without any intervention from the CPU. Transfer status signals from PDC are available in PIO\_PCISR through the flags ENDRX and RXBUFF (see "[PIO Parallel Capture Interrupt Status Register](#)" on page 719).

The parallel capture mode can take into account the sensor data enable signals or not. If the bit ALWYS is set to 0 in PIO\_PCMR, the parallel capture mode samples the sensor data at the rising edge of the sensor clock only if both data enable signals are active (at 1). If the bit ALWYS is set to 1, the parallel capture mode samples the sensor data at the rising edge of the sensor clock whichever the data enable signals are.

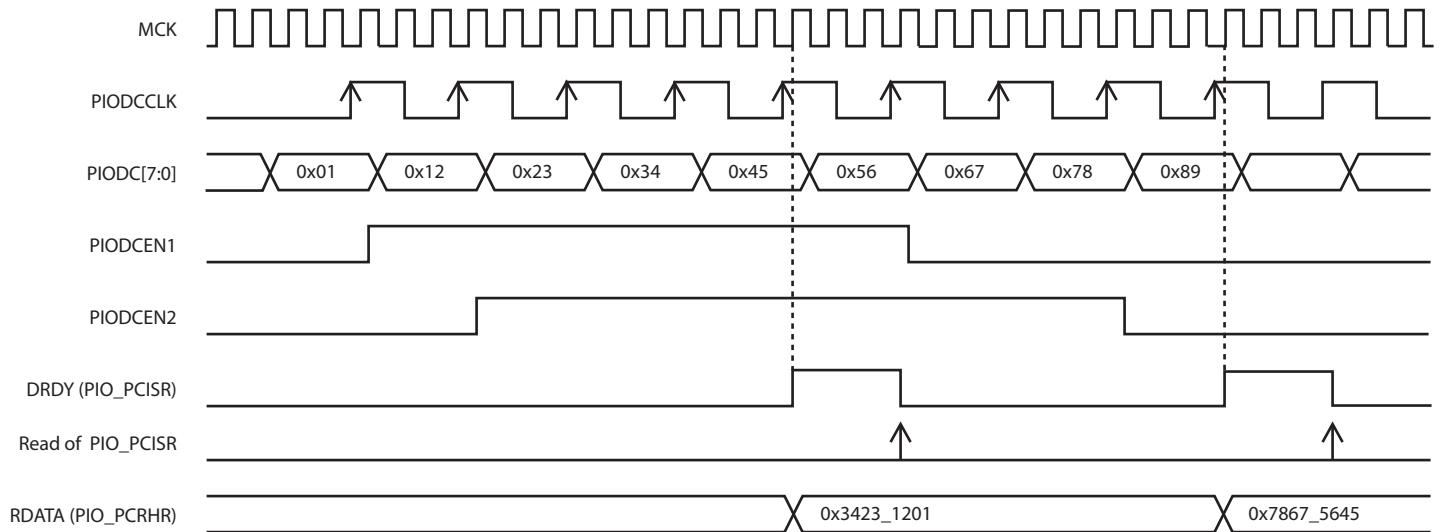
The parallel capture mode can sample the sensor data only one time out of two. This is particularly useful when the user wants only to sample the luminance Y of a CMOS digital image sensor which outputs a YUV422 data stream. If the HALFS bit is set to 0 in PIO\_PCMR, the parallel capture mode samples the sensor data in the conditions described above. If the HALFS bit is set to 1 in PIO\_PCMR, the parallel capture mode samples the sensor data in the conditions described above, but only one time out of two. Depending on the FRSTS bit in PIO\_PCMR, the sensor can either sample the even or odd sensor data. If sensor data are numbered in the order that they are received with an index from 0 to n, if FRSTS = 0 then only data with an even index are sampled, if FRSTS = 1 then only data with an odd index are sampled. If data is ready in PIO\_PCRHR and it is not read before a new data is stored in PIO\_PCRHR, then an overrun error occurs. The previous data is lost and the OVRE flag in PIO\_PCISR is set to 1. This flag is automatically reset when PIO\_PCISR is read (reset after read).

The flags DRDY, OVRE, ENDRX and RXBUFF can be a source of the PIO interrupt.

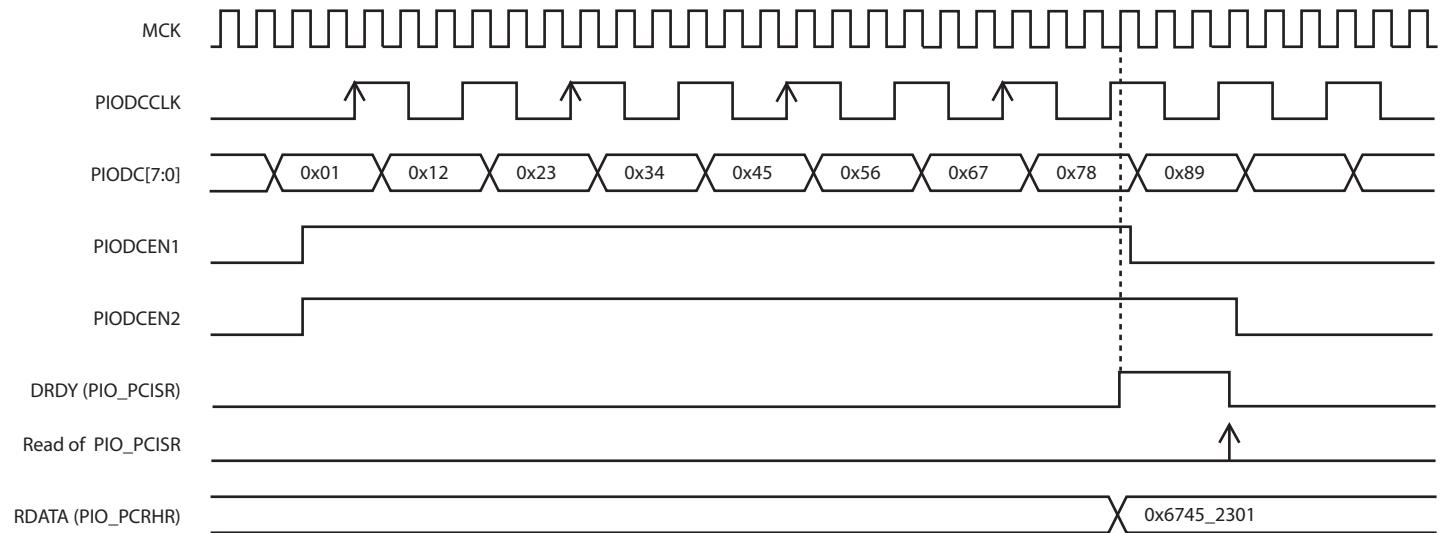
**Figure 33-12. Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 0, HALFS = 0)**



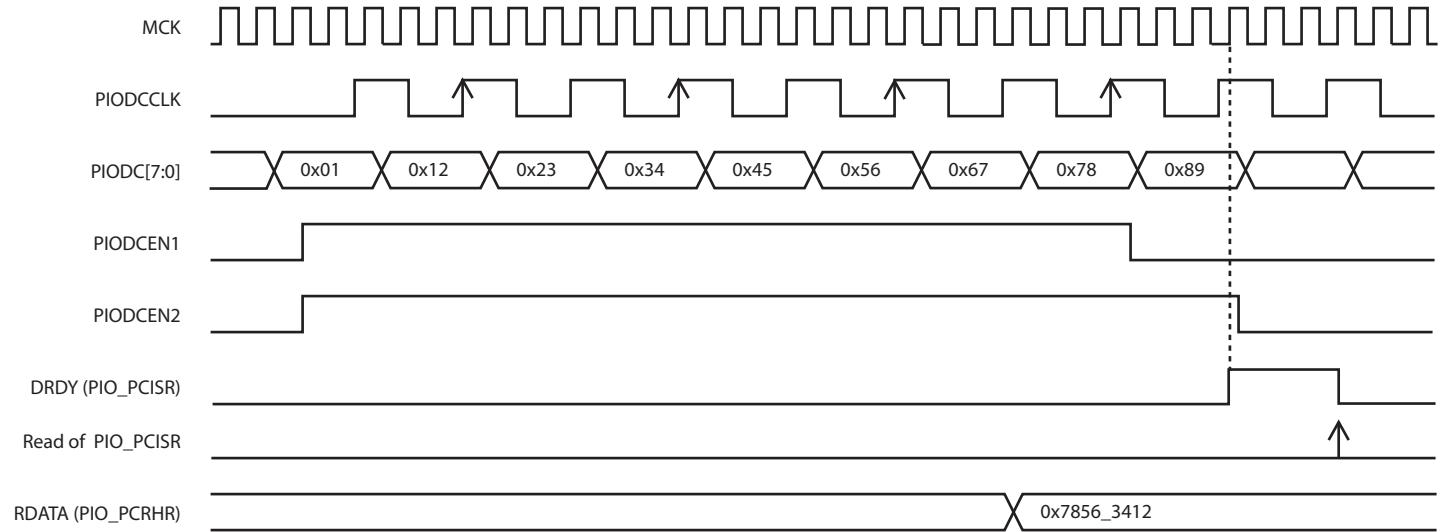
**Figure 33-13. Parallel Capture Mode Waveforms (DSIZE=2, ALWYS=1, HALFS=0)**



**Figure 33-14. Parallel Capture Mode Waveforms (DSIZE=2, ALWYS=0, HALFS=1, FRSTS=0)**



**Figure 33-15. Parallel Capture Mode Waveforms (DSIZE=2, ALWYS=0, HALFS=1, FRSTS=1)**



### 33.5.14.3 Restrictions

- Configuration fields DSIZE, ALWYS, HALFS and FRSTS in PIO\_PCMR (“[PIO Parallel Capture Mode Register](#)”) can be changed **ONLY** if the parallel capture mode is disabled at this time (PCEN = 0 in PIO\_PCMR).
- Frequency of PIO controller clock must be strictly superior to 2 times the frequency of the clock of the device which generates the parallel data.

### 33.5.14.4 Programming Sequence

*Without PDC*

1. Write PIO\_PCIDR and PIO\_PCIER (“[PIO Parallel Capture Interrupt Disable Register](#)” and “[PIO Parallel Capture Interrupt Enable Register](#)”) in order to configure the parallel capture mode interrupt mask.
2. Write PIO\_PCMR (“[PIO Parallel Capture Mode Register](#)”) to set the fields DSIZE, ALWYS, HALFS and FRSTS in order to configure the parallel capture mode **WITHOUT** enabling the parallel capture mode.
3. Write PIO\_PCMR to set the PCEN bit to 1 in order to enable the parallel capture mode **WITHOUT** changing the previous configuration.
4. Wait for a data ready by polling the DRDY flag in PIO\_PCISR (“[PIO Parallel Capture Interrupt Status Register](#)”) or by waiting the corresponding interrupt.
5. Check OVRE flag in PIO\_PCISR.
6. Read the data in PIO\_PCRHR (“[PIO Parallel Capture Reception Holding Register](#)”).
7. If new data are expected go to step 4.
8. Write PIO\_PCMR to set the PCEN bit to 0 in order to disable the parallel capture mode **WITHOUT** changing the previous configuration.

*With PDC*

1. Write PIO\_PCIDR and PIO\_PCIER (“[PIO Parallel Capture Interrupt Disable Register](#)” and “[PIO Parallel Capture Interrupt Enable Register](#)”) in order to configure the parallel capture mode interrupt mask.
2. Configure PDC transfer in PDC registers.
3. Write PIO\_PCMR (“[PIO Parallel Capture Mode Register](#)”) to set the fields DSIZE, ALWYS, HALFS and FRSTS in order to configure the parallel capture mode **WITHOUT** enabling the parallel capture mode.
4. Write PIO\_PCMR to set PCEN bit to 1 in order to enable the parallel capture mode **WITHOUT** changing the previous configuration.
5. Wait for end of transfer by waiting the interrupt corresponding the flag ENDRX in PIO\_PCISR (“[PIO Parallel Capture Interrupt Status Register](#)”).
6. Check OVRE flag in PIO\_PCISR.

7. If a new buffer transfer is expected go to step 5.
8. Write PIO\_PCMR to set the PCEN bit to 0 in order to disable the parallel capture mode **WITHOUT** changing the previous configuration.

### 33.5.15 Write Protection Registers

To prevent any single software error that may corrupt PIO behavior, certain address spaces can be write-protected by setting the WPEN bit in the “[PIO Write Protect Mode Register](#)” (PIO\_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the PIO Write Protect Status Register (PIO\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the PIO Write Protect Mode Register (PIO\_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- “[PIO Enable Register](#)” on page 679
- “[PIO Disable Register](#)” on page 679
- “[PIO Output Enable Register](#)” on page 681
- “[PIO Output Disable Register](#)” on page 681
- “[PIO Input Filter Enable Register](#)” on page 683
- “[PIO Input Filter Disable Register](#)” on page 683
- “[PIO Multi-driver Enable Register](#)” on page 691
- “[PIO Multi-driver Disable Register](#)” on page 691
- “[PIO Pull Up Disable Register](#)” on page 693
- “[PIO Pull Up Enable Register](#)” on page 693
- “[PIO Peripheral ABCD Select Register 1](#)” on page 695
- “[PIO Peripheral ABCD Select Register 2](#)” on page 696
- “[PIO Output Write Enable Register](#)” on page 702
- “[PIO Output Write Disable Register](#)” on page 702
- “[PIO Pad Pull Down Disable Register](#)” on page 700
- “[PIO Pad Pull Down Status Register](#)” on page 701
- “[PIO Parallel Capture Mode Register](#)” on page 715

## 33.6 I/O Lines Programming Example

The programming example as shown in [Table 33-2](#) below is used to obtain the following configuration.

- 4-bit output port on I/O lines 0 to 3, (should be written in a single write operation), open-drain, with pull-up resistor
- Four output signals on I/O lines 4 to 7 (to drive LEDs for example), driven high and low, no pull-up resistor, no pull-down resistor
- Four input signals on I/O lines 8 to 11 (to read push-button states for example), with pull-up resistors, glitch filters and input change interrupts
- Four input signals on I/O line 12 to 15 to read an external device status (polled, thus no input change interrupt), no pull-up resistor, no glitch filter
- I/O lines 16 to 19 assigned to peripheral A functions with pull-up resistor
- I/O lines 20 to 23 assigned to peripheral B functions with pull-down resistor
- I/O line 24 to 27 assigned to peripheral C with Input Change Interrupt, no pull-up resistor and no pull-down resistor
- I/O line 28 to 31 assigned to peripheral D, no pull-up resistor and no pull-down resistor

**Table 33-2. Programming Example**

Register	Value to be Written
PIO_PER	0x0000_FFFF
PIO_PDR	0xFFFF_0000
PIO_OER	0x0000_00FF

**Table 33-2. Programming Example**

PIO_ODR	0xFFFF_FF00
PIO_IFER	0x0000_0F00
PIO_IFDR	0xFFFF_F0FF
PIO_SODR	0x0000_0000
PIO_CODR	0xFFFF_FFFF
PIO_IER	0x0F00_0F00
PIO_IDR	0xF0FF_F0FF
PIO_MDER	0x0000_000F
PIO_MDDR	0xFFFF_FFF0
PIO_PUDR	0xFFF0_00F0
PIO_PUER	0x000F_FF0F
PIO_PPDDR	0xFF0F_FFFF
PIO_PPDER	0x00F0_0000
PIO_ABCDSR1	0xF0F0_0000
PIO_ABCDSR2	0xFF00_0000
PIO_OWER	0x0000_000F
PIO_OWDR	0x0FFF_FFF0

### 33.7 Parallel Input/Output Controller (PIO) User Interface

Each I/O line controlled by the PIO Controller is associated with a bit in each of the PIO Controller User Interface registers. Each register is 32 bits wide. If a parallel I/O line is not defined, writing to the corresponding bits has no effect. Undefined bits read zero. If the I/O line is not multiplexed with any peripheral, the I/O line is controlled by the PIO Controller and PIO\_PSR returns 1 systematically.

**Table 33-3. Register Mapping**

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved	–	–	–
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved	–	–	–
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved	–	–	–
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	–
0x0038	Output Data Status Register	PIO_ODSR	Read-only or(2) Read-write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register <sup>(4)</sup>	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved	–	–	–
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	(1)
0x006C	Reserved	–	–	–

**Table 33-3. Register Mapping (Continued)**

Offset	Register	Name	Access	Reset
0x0070	Peripheral Select Register 1	PIO_ABCDSR1	Read-write	0x00000000
0x0074	Peripheral Select Register 2	PIO_ABCDSR2	Read-write	0x00000000
0x0078 to 0x007C	Reserved	—	—	—
0x0080	Input Filter Slow Clock Disable Register	PIO_IFSCDR	Write-only	—
0x0084	Input Filter Slow Clock Enable Register	PIO_IFSCER	Write-only	—
0x0088	Input Filter Slow Clock Status Register	PIO_IFCSR	Read-only	0x00000000
0x008C	Slow Clock Divider Debouncing Register	PIO_SCDR	Read-write	0x00000000
0x0090	Pad Pull-down Disable Register	PIO_PPDDR	Write-only	—
0x0094	Pad Pull-down Enable Register	PIO_PPDER	Write-only	—
0x0098	Pad Pull-down Status Register	PIO_PPDSR	Read-only	(1)
0x009C	Reserved	—	—	—
0x00A0	Output Write Enable	PIO_OWER	Write-only	—
0x00A4	Output Write Disable	PIO_OWDR	Write-only	—
0x00A8	Output Write Status Register	PIO_OWSR	Read-only	0x00000000
0x00AC	Reserved	—	—	—
0x00B0	Additional Interrupt Modes Enable Register	PIO_AIMER	Write-only	—
0x00B4	Additional Interrupt Modes Disables Register	PIO_AIMDR	Write-only	—
0x00B8	Additional Interrupt Modes Mask Register	PIO_AIMMR	Read-only	0x00000000
0x00BC	Reserved	—	—	—
0x00C0	Edge Select Register	PIO_ESR	Write-only	—
0x00C4	Level Select Register	PIO_LSR	Write-only	—
0x00C8	Edge/Level Status Register	PIO_ELSR	Read-only	0x00000000
0x00CC	Reserved	—	—	—
0x00D0	Falling Edge/Low Level Select Register	PIO_FELLSR	Write-only	—
0x00D4	Rising Edge/ High Level Select Register	PIO_REHLSR	Write-only	—
0x00D8	Fall/Rise - Low/High Status Register	PIO_FRLHSR	Read-only	0x00000000
0x00DC	Reserved	—	—	—
0x00E0	Lock Status	PIO_LOCKSR	Read-only	0x00000000
0x00E4	Write Protect Mode Register	PIO_WPMR	Read-write	0x0
0x00E8	Write Protect Status Register	PIO_WPSR	Read-only	0x0
0x00EC to 0x00F8	Reserved	—	—	—
0x0100	Schmitt Trigger Register	PIO_SCHMITT	Read-write	0x00000000
0x0104- 0x010C	Reserved	—	—	—
0x0110	IO Delay Register	PIO_DELAYR	Read-write	0x00000000
0x0114- 0x011C	Reserved	—	—	—

**Table 33-3. Register Mapping (Continued)**

Offset	Register	Name	Access	Reset
0x0120 to 0x014C	Reserved	-	-	-
0x150	Parallel Capture Mode Register	PIO_PCMR	Read-write	0x00000000
0x154	Parallel Capture Interrupt Enable Register	PIO_PCIER	Write-only	-
0x158	Parallel Capture Interrupt Disable Register	PIO_PCIDR	Write-only	-
0x15C	Parallel Capture Interrupt Mask Register	PIO_PCIMR	Read-only	0x00000000
0x160	Parallel Capture Interrupt Status Register	PIO_PCISR	Read-only	0x00000000
0x164	Parallel Capture Reception Holding Register	PIO_PCRHR	Read-only	0x00000000
0x0168 to 0x018C	Reserved for PDC Registers	-	-	-

- Notes:
1. Reset value depends on the product implementation.
  2. PIO\_ODSR is Read-only or Read/Write depending on PIO\_OWSR I/O lines.
  3. Reset value of PIO\_PDSR depends on the level of the I/O lines. Reading the I/O line levels requires the clock of the PIO Controller to be enabled, otherwise PIO\_PDSR reads the levels present on the I/O line at the time the clock was disabled.
  4. PIO\_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.

Note: If an offset is not listed in the table, it must be considered as reserved.

### 33.7.1 PIO Enable Register

**Name:** PIO\_PER

**Address:** 0x400E0E00 (PIOA), 0x400E1000 (PIOB), 0x400E1200 (PIOC), 0x400E1400 (PIOD), 0x400E1600 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: PIO Enable**

0: No effect.

1: Enables the PIO to control the corresponding pin (disables peripheral control of the pin).

### 33.7.2 PIO Disable Register

**Name:** PIO\_PDR

**Address:** 0x400E0E04 (PIOA), 0x400E1004 (PIOB), 0x400E1204 (PIOC), 0x400E1404 (PIOD), 0x400E1604 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: PIO Disable**

0: No effect.

1: Disables the PIO from controlling the corresponding pin (enables peripheral control of the pin).

### 33.7.3 PIO Status Register

**Name:** PIO\_PSR

**Address:** 0x400E0E08 (PIOA), 0x400E1008 (PIOB), 0x400E1208 (PIOC), 0x400E1408 (PIOD), 0x400E1608 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: PIO Status**

0: PIO is inactive on the corresponding I/O line (peripheral is active).

1: PIO is active on the corresponding I/O line (peripheral is inactive).

### 33.7.4 PIO Output Enable Register

**Name:** PIO\_OER

**Address:** 0x400E0E10 (PIOA), 0x400E1010 (PIOB), 0x400E1210 (PIOC), 0x400E1410 (PIOD), 0x400E1610 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Output Enable**

0: No effect.

1: Enables the output on the I/O line.

### 33.7.5 PIO Output Disable Register

**Name:** PIO\_ODR

**Address:** 0x400E0E14 (PIOA), 0x400E1014 (PIOB), 0x400E1214 (PIOC), 0x400E1414 (PIOD), 0x400E1614 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Output Disable**

0: No effect.

1: Disables the output on the I/O line.

### 33.7.6 PIO Output Status Register

**Name:** PIO\_OSR

**Address:** 0x400E0E18 (PIOA), 0x400E1018 (PIOB), 0x400E1218 (PIOC), 0x400E1418 (PIOD), 0x400E1618 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Status**

0: The I/O line is a pure input.

1: The I/O line is enabled in output.

### 33.7.7 PIO Input Filter Enable Register

**Name:** PIO\_IFER

**Address:** 0x400E0E20 (PIOA), 0x400E1020 (PIOB), 0x400E1220 (PIOC), 0x400E1420 (PIOD), 0x400E1620 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Input Filter Enable**

0: No effect.

1: Enables the input glitch filter on the I/O line.

### 33.7.8 PIO Input Filter Disable Register

**Name:** PIO\_IFDR

**Address:** 0x400E0E24 (PIOA), 0x400E1024 (PIOB), 0x400E1224 (PIOC), 0x400E1424 (PIOD), 0x400E1624 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Input Filter Disable**

0: No effect.

1: Disables the input glitch filter on the I/O line.

### 33.7.9 PIO Input Filter Status Register

**Name:** PIO\_IFSR

**Address:** 0x400E0E28 (PIOA), 0x400E1028 (PIOB), 0x400E1228 (PIOC), 0x400E1428 (PIOD), 0x400E1628 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Filer Status**

0: The input glitch filter is disabled on the I/O line.

1: The input glitch filter is enabled on the I/O line.

### 33.7.10 PIO Set Output Data Register

**Name:** PIO\_SODR

**Address:** 0x400E0E30 (PIOA), 0x400E1030 (PIOB), 0x400E1230 (PIOC), 0x400E1430 (PIOD), 0x400E1630 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Set Output Data**

0: No effect.

1: Sets the data to be driven on the I/O line.

### 33.7.11 PIO Clear Output Data Register

**Name:** PIO\_CODR

**Address:** 0x400E0E34 (PIOA), 0x400E1034 (PIOB), 0x400E1234 (PIOC), 0x400E1434 (PIOD), 0x400E1634 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Clear Output Data**

0: No effect.

1: Clears the data to be driven on the I/O line.

### 33.7.12 PIO Output Data Status Register

**Name:** PIO\_ODSR

**Address:** 0x400E0E38 (PIOA), 0x400E1038 (PIOB), 0x400E1238 (PIOC), 0x400E1438 (PIOD), 0x400E1638 (PIOE)

**Access:** Read-only or Read-write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Data Status**

0: The data to be driven on the I/O line is 0.

1: The data to be driven on the I/O line is 1.

### 33.7.13 PIO Pin Data Status Register

**Name:** PIO\_PDSR

**Address:** 0x400E0E3C (PIOA), 0x400E103C (PIOB), 0x400E123C (PIOC), 0x400E143C (PIOD), 0x400E163C (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Data Status**

0: The I/O line is at level 0.

1: The I/O line is at level 1.

### 33.7.14 PIO Interrupt Enable Register

**Name:** PIO\_IER

**Address:** 0x400E0E40 (PIOA), 0x400E1040 (PIOB), 0x400E1240 (PIOC), 0x400E1440 (PIOD), 0x400E1640 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Change Interrupt Enable**

0: No effect.

1: Enables the Input Change Interrupt on the I/O line.

### 33.7.15 PIO Interrupt Disable Register

**Name:** PIO\_IDR

**Address:** 0x400E0E44 (PIOA), 0x400E1044 (PIOB), 0x400E1244 (PIOC), 0x400E1444 (PIOD), 0x400E1644 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Change Interrupt Disable**

0: No effect.

1: Disables the Input Change Interrupt on the I/O line.

### 33.7.16 PIO Interrupt Mask Register

**Name:** PIO\_IMR

**Address:** 0x400E0E48 (PIOA), 0x400E1048 (PIOB), 0x400E1248 (PIOC), 0x400E1448 (PIOD), 0x400E1648 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Change Interrupt Mask**

0: Input Change Interrupt is disabled on the I/O line.

1: Input Change Interrupt is enabled on the I/O line.

### 33.7.17 PIO Interrupt Status Register

**Name:** PIO\_ISR

**Address:** 0x400E0E4C (PIOA), 0x400E104C (PIOB), 0x400E124C (PIOC), 0x400E144C (PIOD), 0x400E164C (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Change Interrupt Status**

0: No Input Change has been detected on the I/O line since PIO\_ISR was last read or since reset.

1: At least one Input Change has been detected on the I/O line since PIO\_ISR was last read or since reset.

### 33.7.18 PIO Multi-driver Enable Register

**Name:** PIO\_MDER

**Address:** 0x400E0E50 (PIOA), 0x400E1050 (PIOB), 0x400E1250 (PIOC), 0x400E1450 (PIOD), 0x400E1650 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Multi Drive Enable**

0: No effect.

1: Enables Multi Drive on the I/O line.

### 33.7.19 PIO Multi-driver Disable Register

**Name:** PIO\_MDDR

**Address:** 0x400E0E54 (PIOA), 0x400E1054 (PIOB), 0x400E1254 (PIOC), 0x400E1454 (PIOD), 0x400E1654 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Multi Drive Disable.**

0: No effect.

1: Disables Multi Drive on the I/O line.

### 33.7.20 PIO Multi-driver Status Register

**Name:** PIO\_MDSR

**Address:** 0x400E0E58 (PIOA), 0x400E1058 (PIOB), 0x400E1258 (PIOC), 0x400E1458 (PIOD), 0x400E1658 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Multi Drive Status.**

0: The Multi Drive is disabled on the I/O line. The pin is driven at high and low level.

1: The Multi Drive is enabled on the I/O line. The pin is driven at low level only.

### 33.7.21 PIO Pull Up Disable Register

**Name:** PIO\_PUDR

**Address:** 0x400E0E60 (PIOA), 0x400E1060 (PIOB), 0x400E1260 (PIOC), 0x400E1460 (PIOD), 0x400E1660 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Pull Up Disable.**

0: No effect.

1: Disables the pull up resistor on the I/O line.

### 33.7.22 PIO Pull Up Enable Register

**Name:** PIO\_PUER

**Address:** 0x400E0E64 (PIOA), 0x400E1064 (PIOB), 0x400E1264 (PIOC), 0x400E1464 (PIOD), 0x400E1664 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Pull Up Enable.**

0: No effect.

1: Enables the pull up resistor on the I/O line.

### 33.7.23 PIO Pull Up Status Register

**Name:** PIO\_PUSR

**Address:** 0x400E0E68 (PIOA), 0x400E1068 (PIOB), 0x400E1268 (PIOC), 0x400E1468 (PIOD), 0x400E1668 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Pull Up Status.**

0: Pull Up resistor is enabled on the I/O line.

1: Pull Up resistor is disabled on the I/O line.

### 33.7.24 PIO Peripheral ABCD Select Register 1

**Name:** PIO\_ABCDSR1

**Access:** Read-write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Peripheral Select.**

If the same bit is set to 0 in PIO\_ABCDSR2:

0: Assigns the I/O line to the Peripheral A function.

1: Assigns the I/O line to the Peripheral B function.

If the same bit is set to 1 in PIO\_ABCDSR2:

0: Assigns the I/O line to the Peripheral C function.

1: Assigns the I/O line to the Peripheral D function.

### 33.7.25 PIO Peripheral ABCD Select Register 2

**Name:** PIO\_ABCDSR2

**Access:** Read-write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Peripheral Select.**

If the same bit is set to 0 in PIO\_ABCDSR1:

0: Assigns the I/O line to the Peripheral A function.

1: Assigns the I/O line to the Peripheral C function.

If the same bit is set to 1 in PIO\_ABCDSR1:

0: Assigns the I/O line to the Peripheral B function.

1: Assigns the I/O line to the Peripheral D function.

### 33.7.26 PIO Input Filter Slow Clock Disable Register

**Name:** PIO\_IFSCDR

**Address:** 0x400E0E80 (PIOA), 0x400E1080 (PIOB), 0x400E1280 (PIOC), 0x400E1480 (PIOD), 0x400E1680 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: PIO Clock Glitch Filtering Select.**

0: No Effect.

1: The Glitch Filter is able to filter glitches with a duration < Tmck/2.

### 33.7.27 PIO Input Filter Slow Clock Enable Register

**Name:** PIO\_IFSCER

**Address:** 0x400E0E84 (PIOA), 0x400E1084 (PIOB), 0x400E1284 (PIOC), 0x400E1484 (PIOD), 0x400E1684 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Debouncing Filtering Select.**

0: No Effect.

1: The Debouncing Filter is able to filter pulses with a duration < Tdiv\_slclk/2.

### 33.7.28 PIO Input Filter Slow Clock Status Register

**Name:** PIO\_IFCSR

**Address:** 0x400E0E88 (PIOA), 0x400E1088 (PIOB), 0x400E1288 (PIOC), 0x400E1488 (PIOD), 0x400E1688 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Glitch or Debouncing Filter Selection Status**

0: The Glitch Filter is able to filter glitches with a duration < Tmck2.

1: The Debouncing Filter is able to filter pulses with a duration < Tdiv\_slclk/2.

### 33.7.29 PIO Slow Clock Divider Debouncing Register

**Name:** PIO\_SCDR

**Address:** 0x400E0E8C (PIOA), 0x400E108C (PIOB), 0x400E128C (PIOC), 0x400E148C (PIOD), 0x400E168C (PIOE)

**Access:** Read-write

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

- **DIV: Slow Clock Divider Selection for Debouncing**

Tdiv\_sclk = 2\*(DIV+1)\*Tslow\_clock.

### 33.7.30 PIO Pad Pull Down Disable Register

**Name:** PIO\_PPDDR

**Address:** 0x400E0E90 (PIOA), 0x400E1090 (PIOB), 0x400E1290 (PIOC), 0x400E1490 (PIOD), 0x400E1690 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Pull Down Disable**

0: No effect.

1: Disables the pull down resistor on the I/O line.

### 33.7.31 PIO Pad Pull Down Enable Register

**Name:** PIO\_PPDER

**Address:** 0x400E0E94 (PIOA), 0x400E1094 (PIOB), 0x400E1294 (PIOC), 0x400E1494 (PIOD), 0x400E1694 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Pull Down Enable**

0: No effect.

1: Enables the pull down resistor on the I/O line.

### 33.7.32 PIO Pad Pull Down Status Register

**Name:** PIO\_PPDSR

**Address:** 0x400E0E98 (PIOA), 0x400E1098 (PIOB), 0x400E1298 (PIOC), 0x400E1498 (PIOD), 0x400E1698 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Pull Down Status**

0: Pull Down resistor is enabled on the I/O line.

1: Pull Down resistor is disabled on the I/O line.

### 33.7.33 PIO Output Write Enable Register

**Name:** PIO\_OWER

**Address:** 0x400E0EA0 (PIOA), 0x400E10A0 (PIOB), 0x400E12A0 (PIOC), 0x400E14A0 (PIOD), 0x400E16A0 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Output Write Enable**

0: No effect.

1: Enables writing PIO\_ODSR for the I/O line.

### 33.7.34 PIO Output Write Disable Register

**Name:** PIO\_OWDR

**Address:** 0x400E0EA4 (PIOA), 0x400E10A4 (PIOB), 0x400E12A4 (PIOC), 0x400E14A4 (PIOD), 0x400E16A4 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **P0-P31: Output Write Disable**

0: No effect.

1: Disables writing PIO\_ODSR for the I/O line.

### 33.7.35 PIO Output Write Status Register

**Name:** PIO\_OWSR

**Address:** 0x400E0EA8 (PIOA), 0x400E10A8 (PIOB), 0x400E12A8 (PIOC), 0x400E14A8 (PIOD), 0x400E16A8 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Write Status**

0: Writing PIO\_ODSR does not affect the I/O line.

1: Writing PIO\_ODSR affects the I/O line.

### 33.7.36 PIO Additional Interrupt Modes Enable Register

**Name:** PIO\_AIMER

**Address:** 0x400E0EB0 (PIOA), 0x400E10B0 (PIOB), 0x400E12B0 (PIOC), 0x400E14B0 (PIOD), 0x400E16B0 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Additional Interrupt Modes Enable**

0: No effect.

1: The interrupt source is the event described in PIO\_ELSR and PIO\_FRLHSR.

### 33.7.37 PIO Additional Interrupt Modes Disable Register

**Name:** PIO\_AIMDR

**Address:** 0x400E0EB4 (PIOA), 0x400E10B4 (PIOB), 0x400E12B4 (PIOC), 0x400E14B4 (PIOD), 0x400E16B4 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Additional Interrupt Modes Disable**

0: No effect.

1: The interrupt mode is set to the default interrupt mode (Both Edge detection).

### 33.7.38 PIO Additional Interrupt Modes Mask Register

**Name:** PIO\_AIMMR

**Address:** 0x400E0EB8 (PIOA), 0x400E10B8 (PIOB), 0x400E12B8 (PIOC), 0x400E14B8 (PIOD), 0x400E16B8 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Peripheral CD Status**

0: The interrupt source is a Both Edge detection event.

1: The interrupt source is described by the registers PIO\_ELSR and PIO\_FRLHSR.

### 33.7.39 PIO Edge Select Register

**Name:** PIO\_ESR

**Address:** 0x400E0EC0 (PIOA), 0x400E10C0 (PIOB), 0x400E12C0 (PIOC), 0x400E14C0 (PIOD), 0x400E16C0 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Edge Interrupt Selection**

0: No effect.

1: The interrupt source is an Edge detection event.

### 33.7.40 PIO Level Select Register

**Name:** PIO\_LSR

**Address:** 0x400E0EC4 (PIOA), 0x400E10C4 (PIOB), 0x400E12C4 (PIOC), 0x400E14C4 (PIOD), 0x400E16C4 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Level Interrupt Selection**

0: No effect.

1: The interrupt source is a Level detection event.

### 33.7.41 PIO Edge/Level Status Register

**Name:** PIO\_ELSR

**Address:** 0x400E0EC8 (PIOA), 0x400E10C8 (PIOB), 0x400E12C8 (PIOC), 0x400E14C8 (PIOD), 0x400E16C8 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Edge/Level Interrupt Source Selection**

0: The interrupt source is an Edge detection event.

1: The interrupt source is a Level detection event.

### 33.7.42 PIO Falling Edge/Low Level Select Register

**Name:** PIO\_FELLSR

**Address:** 0x400E0ED0 (PIOA), 0x400E10D0 (PIOB), 0x400E12D0 (PIOC), 0x400E14D0 (PIOD), 0x400E16D0 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Falling Edge/Low Level Interrupt Selection**

0: No effect.

1: The interrupt source is set to a Falling Edge detection or Low Level detection event, depending on PIO\_ELSR.

### 33.7.43 PIO Rising Edge/High Level Select Register

**Name:** PIO\_REHLSR

**Address:** 0x400E0ED4 (PIOA), 0x400E10D4 (PIOB), 0x400E12D4 (PIOC), 0x400E14D4 (PIOD), 0x400E16D4 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Rising Edge /High Level Interrupt Selection**

0: No effect.

1: The interrupt source is set to a Rising Edge detection or High Level detection event, depending on PIO\_ELSR.

### 33.7.44 PIO Fall/Rise - Low/High Status Register

**Name:** PIO\_FRLHSR

**Address:** 0x400E0ED8 (PIOA), 0x400E10D8 (PIOB), 0x400E12D8 (PIOC), 0x400E14D8 (PIOD), 0x400E16D8 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Edge /Level Interrupt Source Selection**

0: The interrupt source is a Falling Edge detection (if PIO\_ELSR = 0) or Low Level detection event (if PIO\_ELSR = 1).

1: The interrupt source is a Rising Edge detection (if PIO\_ELSR = 0) or High Level detection event (if PIO\_ELSR = 1).

### 33.7.45 PIO Lock Status Register

**Name:** PIO\_LOCKSR

**Address:** 0x400E0EE0 (PIOA), 0x400E10E0 (PIOB), 0x400E12E0 (PIOC), 0x400E14E0 (PIOD), 0x400E16E0 (PIOE)

**Access:** Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Lock Status**

0: The I/O line is not locked.

1: The I/O line is locked.

### 33.7.46 PIO Write Protect Mode Register

**Name:** PIO\_WPMR

**Address:** 0x400E0EE4 (PIOA), 0x400E10E4 (PIOB), 0x400E12E4 (PIOC), 0x400E14E4 (PIOD), 0x400E16E4 (PIOE)

**Access:** Read-write

**Reset:** See [Table 33-3](#)

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

For more information on Write Protection Registers, refer to [Section 33.7 "Parallel Input/Output Controller \(PIO\) User Interface"](#).

- **WPEN: Write Protect Enable**

0: Disables the Write Protect if WPKEY corresponds to 0x50494F ("PIO" in ASCII).

1: Enables the Write Protect if WPKEY corresponds to 0x50494F ("PIO" in ASCII).

Protects the registers:

["PIO Enable Register" on page 679](#)

["PIO Disable Register" on page 679](#)

["PIO Output Enable Register" on page 681](#)

["PIO Output Disable Register" on page 681](#)

["PIO Input Filter Enable Register" on page 683](#)

["PIO Input Filter Disable Register" on page 683](#)

["PIO Multi-driver Enable Register" on page 691](#)

["PIO Multi-driver Disable Register" on page 691](#)

["PIO Pull Up Disable Register" on page 693](#)

["PIO Pull Up Enable Register" on page 693](#)

["PIO Peripheral ABCD Select Register 1" on page 695](#)

["PIO Peripheral ABCD Select Register 2" on page 696](#)

["PIO Output Write Enable Register" on page 702](#)

["PIO Output Write Disable Register" on page 702](#)

["PIO Pad Pull Down Disable Register" on page 700](#)

["PIO Pad Pull Down Status Register" on page 701](#)

["PIO Parallel Capture Mode Register" on page 715](#)

- **WPKEY: Write Protect KEY.**

Value	Name	Description
0x50494F	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 33.7.47 PIO Write Protect Status Register

**Name:** PIO\_WPSR

**Address:** 0x400E0EE8 (PIOA), 0x400E10E8 (PIOB), 0x400E12E8 (PIOC), 0x400E14E8 (PIOD), 0x400E16E8 (PIOE)

**Access:** Read-only

**Reset:** See [Table 33-3](#)

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

- **WPVS: Write Protect Violation Status**

0: No Write Protect Violation has occurred since the last read of the PIO\_WPSR register.

1: A Write Protect Violation has occurred since the last read of the PIO\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Note: Reading PIO\_WPSR automatically clears all fields.

### 33.7.48 PIO Schmitt Trigger Register

**Name:** PIO\_SCHMITT

**Address:** 0x400E0F00 (PIOA), 0x400E1100 (PIOB), 0x400E1300 (PIOC), 0x400E1500 (PIOD), 0x400E1700 (PIOE)

**Access:** Read-write

**Reset:** See [Table 33-3](#)

31	30	29	28	27	26	25	24
SCHMITT31	SCHMITT30	SCHMITT29	SCHMITT28	SCHMITT27	SCHMITT26	SCHMITT25	SCHMITT24
23	22	21	20	19	18	17	16
SCHMITT23	SCHMITT22	SCHMITT21	SCHMITT20	SCHMITT19	SCHMITT18	SCHMITT17	SCHMITT16
15	14	13	12	11	10	9	8
SCHMITT15	SCHMITT14	SCHMITT13	SCHMITT12	SCHMITT11	SCHMITT10	SCHMITT9	SCHMITT8
7	6	5	4	3	2	1	0
SCHMITT7	SCHMITT6	SCHMITT5	SCHMITT4	SCHMITT3	SCHMITT2	SCHMITT1	SCHMITT0

- **SCHMITTx [x=0..31]: Schmitt Trigger Control**

0: Schmitt Trigger is enabled.

1: Schmitt Trigger is disabled.

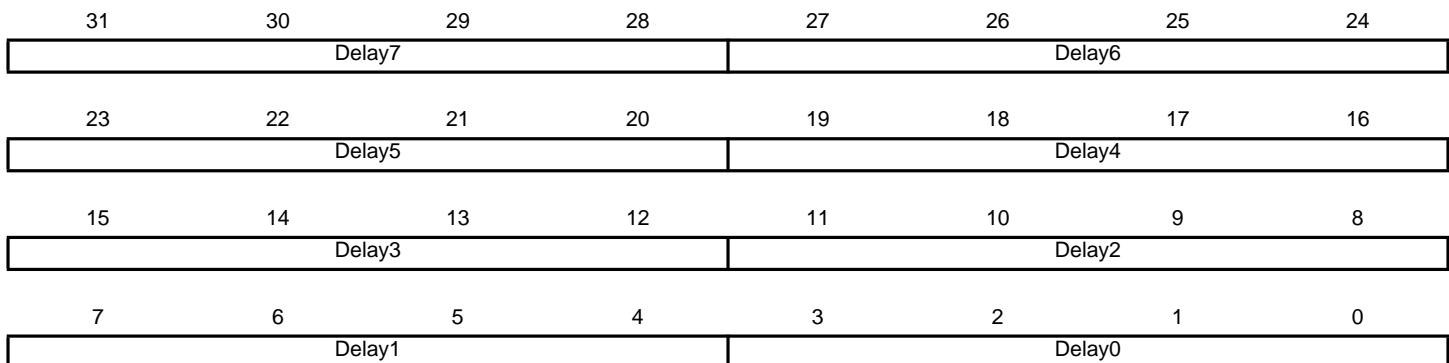
### 33.7.49 PIO I/O Delay Register

**Name:** PIO\_DELAYR

**Address:** 0x400E0F10 (PIOA), 0x400E1110 (PIOB), 0x400E1310 (PIOC), 0x400E1510 (PIOD), 0x400E1710 (PIOE)

**Access:** Read-write

**Reset:** See [Table 33-3](#)



- **Delayx [x=0..7]: Delay Control for Simultaneous Switch Reduction**

Gives the number of elements in the delay line associated to pad x.

### 33.7.50 PIO Parallel Capture Mode Register

**Name:** PIO\_PCMR

**Address:** 0x400E0F50 (PIOA), 0x400E1150 (PIOB), 0x400E1350 (PIOC), 0x400E1550 (PIOD), 0x400E1750 (PIOE)

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in “[PIO Write Protect Mode Register](#)” .

- **PCEN: Parallel Capture Mode Enable**

0: The parallel capture mode is disabled.

1: The parallel capture mode is enabled.

- **DSIZE: Parallel Capture Mode Data Size**

Value	Name	Description
0	BYTE	The reception data in the PIO_PCRHR register is a BYTE (8-bit)
1	HALF-WORD	The reception data in the PIO_PCRHR register is a HALF-WORD (16-bit)
2	WORD	The reception data in the PIO_PCRHR register is a WORD (32-bit)
3	—	Reserved

- **ALWYS: Parallel Capture Mode Always Sampling**

0: The parallel capture mode samples the data when both data enables are active.

1: The parallel capture mode samples the data whatever the data enables are.

- **HALFS: Parallel Capture Mode Half Sampling**

Independently from the ALWYS bit:

0: The parallel capture mode samples all the data.

1: The parallel capture mode samples the data only one time out of two.

- **FRSTS: Parallel Capture Mode First Sample**

This bit is useful only if the HALFS bit is set to 1. If data are numbered in the order that they are received with an index from 0 to n:

0: Only data with an even index are sampled.

1: Only data with an odd index are sampled.

### 33.7.51 PIO Parallel Capture Interrupt Enable Register

**Name:** PIO\_PCIER

**Address:** 0x400E0F54 (PIOA), 0x400E1154 (PIOB), 0x400E1354 (PIOC), 0x400E1554 (PIOD), 0x400E1754 (PIOE)

**Access:** Write-only

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
—	—	—	—	RXBUFF	ENDRX	OVRE	DRDY

- **DRDY:** Parallel Capture Mode Data Ready Interrupt Enable
- **OVRE:** Parallel Capture Mode Overrun Error Interrupt Enable
- **ENDRX:** End of Reception Transfer Interrupt Enable
- **RXBUFF:** Reception Buffer Full Interrupt Enable

### 33.7.52 PIO Parallel Capture Interrupt Disable Register

**Name:** PIO\_PCIDR

**Address:** 0x400E0F58 (PIOA), 0x400E1158 (PIOB), 0x400E1358 (PIOC), 0x400E1558 (PIOD), 0x400E1758 (PIOE)

**Access:** Write-only

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
—	—	—	—	RXBUFF	ENDRX	OVRE	DRDY

- **DRDY:** Parallel Capture Mode Data Ready Interrupt Disable
- **OVRE:** Parallel Capture Mode Overrun Error Interrupt Disable
- **ENDRX:** End of Reception Transfer Interrupt Disable
- **RXBUFF:** Reception Buffer Full Interrupt Disable

### 33.7.53 PIO Parallel Capture Interrupt Mask Register

**Name:** PIO\_PCIMR

**Address:** 0x400E0F5C (PIOA), 0x400E115C (PIOB), 0x400E135C (PIOC), 0x400E155C (PIOD), 0x400E175C (PIOE)

**Access:** Read-only

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
—	—	—	—	RXBUFF	ENDRX	OVRE	DRDY

- **DRDY:** Parallel Capture Mode Data Ready Interrupt Mask
- **OVRE:** Parallel Capture Mode Overrun Error Interrupt Mask
- **ENDRX:** End of Reception Transfer Interrupt Mask
- **RXBUFF:** Reception Buffer Full Interrupt Mask

### 33.7.54 PIO Parallel Capture Interrupt Status Register

**Name:** PIO\_PCISR

**Address:** 0x400E0F60 (PIOA), 0x400E1160 (PIOB), 0x400E1360 (PIOC), 0x400E1560 (PIOD), 0x400E1760 (PIOE)

**Access:** Read-only

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
—	—	—	—	RXBUFF	ENDRX	OVRE	DRDY

- **DRDY: Parallel Capture Mode Data Ready**

0: No new data is ready to be read since the last read of PIO\_PCRHR.

1: A new data is ready to be read since the last read of PIO\_PCRHR.

The DRDY flag is automatically reset when PIO\_PCRHR is read or when the parallel capture mode is disabled.

- **OVRE: Parallel Capture Mode Overrun Error.**

0: No overrun error occurred since the last read of this register.

1: At least one overrun error occurred since the last read of this register.

The OVRE flag is automatically reset when this register is read or when the parallel capture mode is disabled.

- **ENDRX: End of Reception Transfer.**

0: The End of Transfer signal from the Reception PDC channel is inactive.

1: The End of Transfer signal from the Reception PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The signal Buffer Full from the Reception PDC channel is inactive.

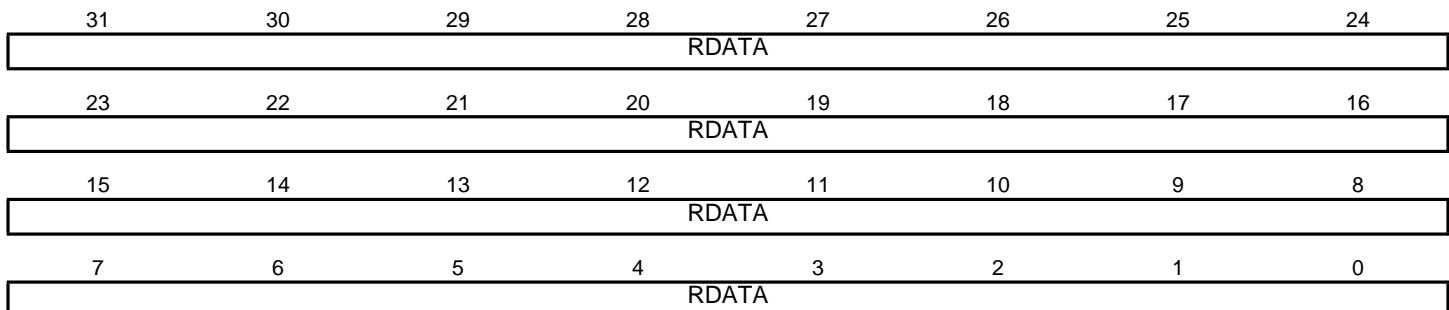
1: The signal Buffer Full from the Reception PDC channel is active.

### 33.7.55 PIO Parallel Capture Reception Holding Register

**Name:** PIO\_PCRHR

**Address:** 0x400E0F64 (PIOA), 0x400E1164 (PIOB), 0x400E1364 (PIOC), 0x400E1564 (PIOD), 0x400E1764 (PIOE)

**Access:** Read-only



- **RDATA: Parallel Capture Mode Reception Data.**

if DSIZE = 0 in PIO\_PCMR, only the 8 LSBs of RDATA are useful.

if DSIZE = 1 in PIO\_PCMR, only the 16 LSBs of RDATA are useful.

## 34. Serial Peripheral Interface (SPI)

### 34.1 Description

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave Mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master” which controls the data flow, while the other devices act as “slaves” which have data shifted into and out by the master. Different CPUs can take turn being masters (Multiple Master Protocol opposite to Single Master Protocol where one CPU is always the master while all of the others are always slaves) and one master may simultaneously shift data into multiple slaves. However, only one slave may drive its output to write data back to the master at any given time.

A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

The SPI system consists of two data lines and two control lines:

- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates; the SPCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows slaves to be turned on and off by hardware.

## 34.2 Embedded Characteristics

- Supports Communication with Serial External Devices
  - Master Mode can drive SPCK up to peripheral clock (bounded by maximum bus clock divided by 2)
  - Slave Mode operates on SPCK, asynchronously to Core and Bus Clock
  - Four Chip Selects with External Decoder Support Allow Communication with Up to 15 Peripherals
  - Serial Memories, such as DataFlash and 3-wire EEPROMs
  - Serial Peripherals, such as ADCs, DACs, LCD Controllers, CAN Controllers and Sensors
  - External Coprocessors
- Master or Slave Serial Peripheral Bus Interface
  - 8-bit to 16-bit Programmable Data Length Per Chip Select
  - Programmable Phase and Polarity Per Chip Select
  - Programmable Transfer Delay Between Consecutive Transfers and Delay before SPI Clock per Chip Select
  - Programmable Delay Between Chip Selects
  - Selectable Mode Fault Detection
- Connection to PDC Channel Capabilities Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter
- Connection to DMA Channel Capabilities Optimizes Data Transfers
  - One channel for the Receiver, One Channel for the Transmitter

### 34.3 Block Diagram

Figure 34-1. Block Diagram

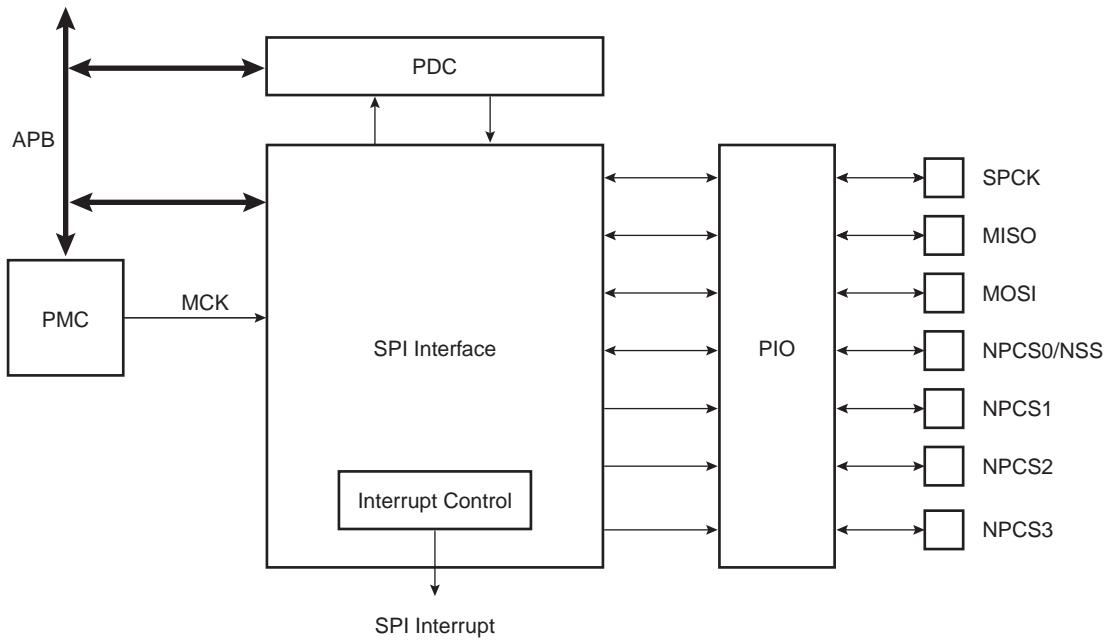
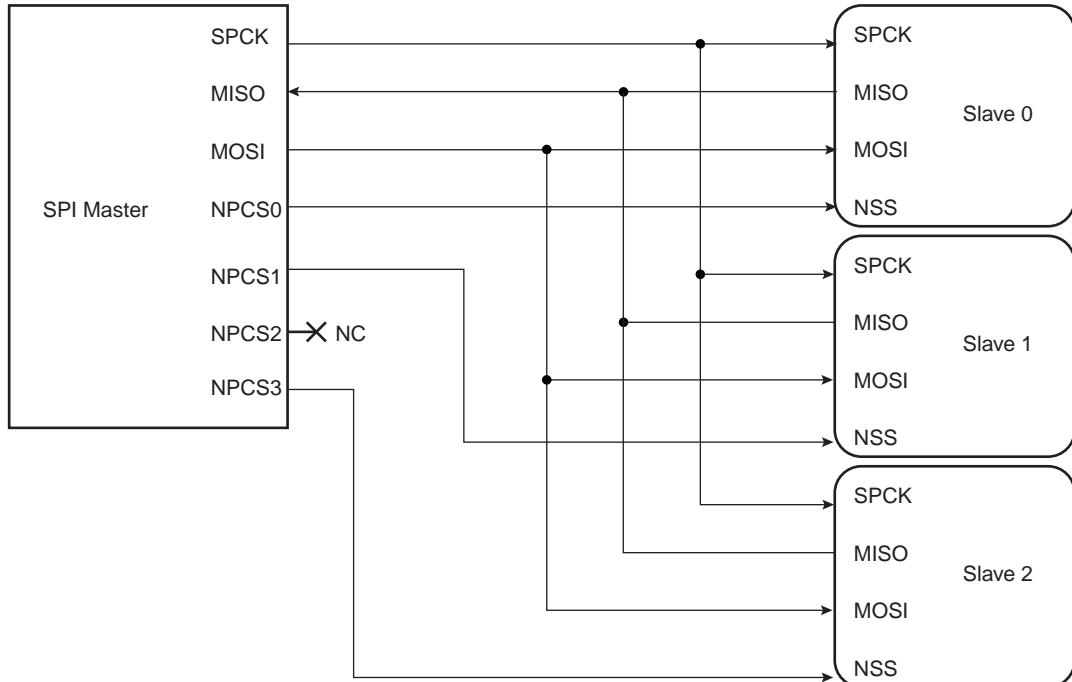


Figure 34-2. Block Diagram

### 34.4 Application Block Diagram

Figure 34-3. Application Block Diagram: Single Master/Multiple Slave Implementation



## 34.5 Signal Description

Table 34-1. Signal Description

Pin Name	Pin Description	Type	
		Master	Slave
MISO	Master In Slave Out	Input	Output
MOSI	Master Out Slave In	Output	Input
SPCK	Serial Clock	Output	Input
NPCS1-NPCS3	Peripheral Chip Selects	Output	Unused
NPCS0/NSS	Peripheral Chip Select/Slave Select	Output	Input

## 34.6 Product Dependencies

### 34.6.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the SPI pins to their peripheral functions.

Table 34-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
SPI	MISO	PA12	A
SPI	MOSI	PA13	A
SPI	NPCS0	PA11	A
SPI	NPCS1	PA9	B
SPI	NPCS1	PA31	A
SPI	NPCS1	PB14	A
SPI	NPCS1	PC4	B
SPI	NPCS2	PA10	B
SPI	NPCS2	PA30	B
SPI	NPCS2	PB2	B
SPI	NPCS3	PA3	B
SPI	NPCS3	PA5	B
SPI	NPCS3	PA22	B
SPI	SPCK	PA14	A

### 34.6.2 Power Management

The SPI may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SPI clock.

### 34.6.3 Interrupt

The SPI interface has an interrupt line connected to the Interrupt Controller. Handling the SPI interrupt requires programming the interrupt controller before configuring the SPI.

**Table 34-3. Peripheral IDs**

Instance	ID
SPI	19

### 34.6.4 Peripheral DMA Controller (PDC) or Direct Memory Access Controller (DMAC)

The SPI interface can be used in conjunction with the PDC or DMAC in order to reduce processor overhead. For a full description of the PDC or DMAC, refer to the corresponding section in the full datasheet.

## 34.7 Functional Description

### 34.7.1 Modes of Operation

The SPI operates in Master Mode or in Slave Mode.

Operation in Master Mode is programmed by writing at 1 the MSTR bit in the Mode Register. The pins NPCS0 to NPCS3 are all configured as outputs, the SPCK pin is driven, the MISO line is wired on the receiver input and the MOSI line driven as an output by the transmitter.

If the MSTR bit is written at 0, the SPI operates in Slave Mode. The MISO line is driven by the transmitter output, the MOSI line is wired on the receiver input, the SPCK pin is driven by the transmitter to synchronize the receiver. The NPCS0 pin becomes an input, and is used as a Slave Select signal (NSS). The pins NPCS1 to NPCS3 are not driven and can be used for other purposes.

The data transfers are identically programmable for both modes of operations. The baud rate generator is activated only in Master Mode.

### 34.7.2 Data Transfer

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the Chip Select Register. The clock phase is programmed with the NCPHA bit. These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

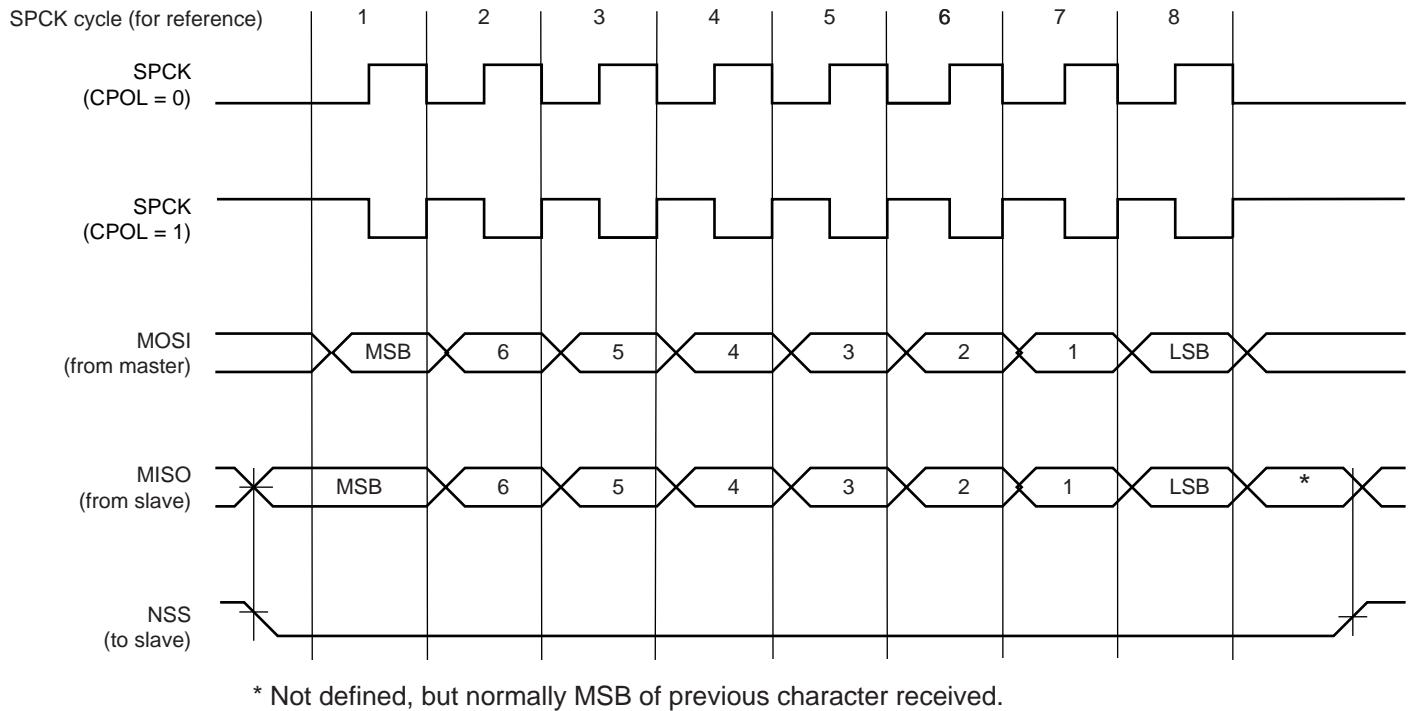
Table 34-4 shows the four modes and corresponding parameter settings.

**Table 34-4. SPI Bus Protocol Mode**

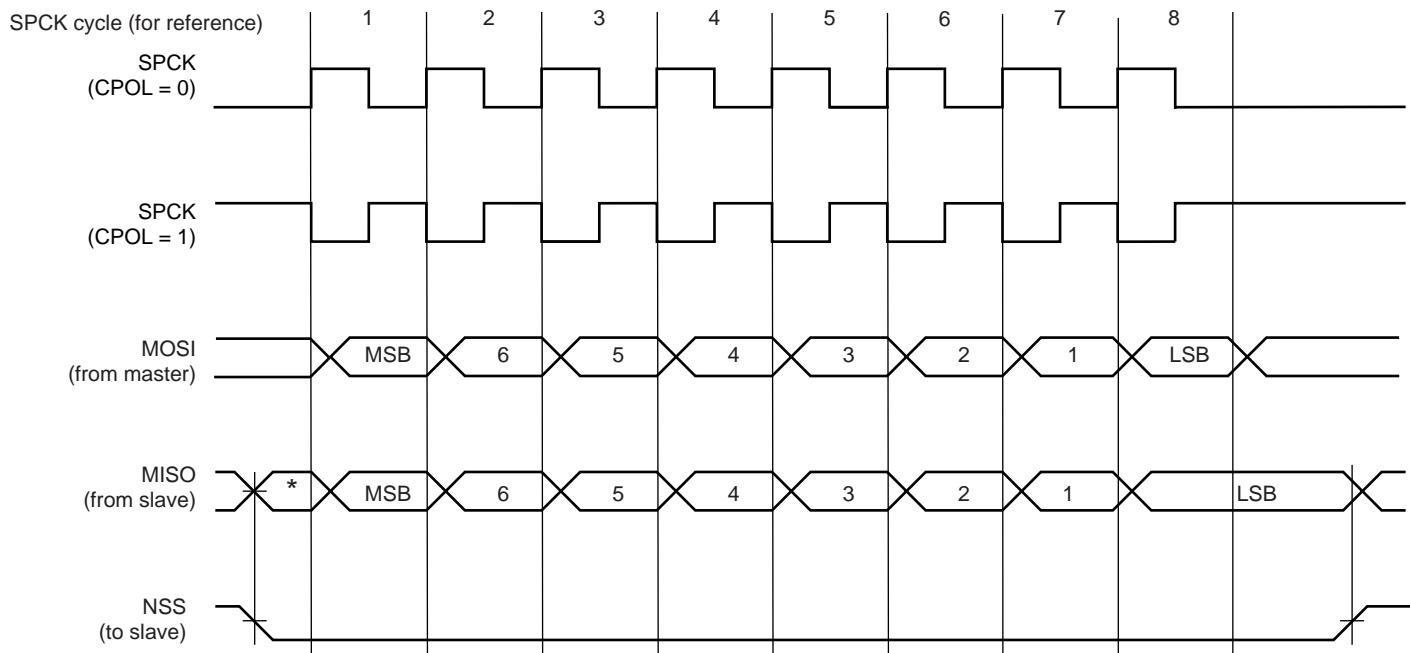
SPI Mode	CPOL	NCPHA	Shift SPCK Edge	Capture SPCK Edge	SPCK Inactive Level
0	0	1	Falling	Rising	Low
1	0	0	Rising	Falling	Low
2	1	1	Rising	Falling	High
3	1	0	Falling	Rising	High

Figure 34-4 and Figure 34-5 show examples of data transfers.

**Figure 34-4. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)**



**Figure 34-5. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)**



### 34.7.3 Master Mode Operations

When configured in Master Mode, the SPI operates on the clock generated by the internal programmable baud rate generator. It fully controls the data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select line to the slave and the serial clock signal (SPCK).

The SPI features two holding registers, the Transmit Data Register and the Receive Data Register, and a single Shift Register. The holding registers maintain the data flow at a constant rate.

After enabling the SPI, a data transfer begins when the processor writes to the SPI\_TDR (Transmit Data Register). The written data is immediately transferred in the Shift Register and transfer on the SPI bus starts. While the data in the Shift Register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift Register. Receiving data cannot occur without transmitting data. If receiving mode is not needed, for example when communicating with a slave receiver only (such as an LCD), the receive status flags in the status register can be discarded.

Before writing the TDR, the PCS field in the SPI\_MR register must be set in order to select a slave.

After enabling the SPI, a data transfer begins when the processor writes to the SPI\_TDR (Transmit Data Register). The written data is immediately transferred in the Shift Register and transfer on the SPI bus starts. While the data in the Shift Register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift Register. Transmission cannot occur without reception.

Before writing the TDR, the PCS field must be set in order to select a slave.

If new data is written in SPI\_TDR during the transfer, it stays in it until the current transfer is completed. Then, the received data is transferred from the Shift Register to SPI\_RDR, the data in SPI\_TDR is loaded in the Shift Register and a new transfer starts.

The transfer of a data written in SPI\_TDR in the Shift Register is indicated by the TDRE bit (Transmit Data Register Empty) in the Status Register (SPI\_SR). When new data is written in SPI\_TDR, this bit is cleared. The TDRE bit is used to trigger the Transmit PDC or DMA channel.

The end of transfer is indicated by the TXEMPTY flag in the SPI\_SR register. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, TXEMPTY is set after the completion of said delay. The master clock (MCK) can be switched off at this time.

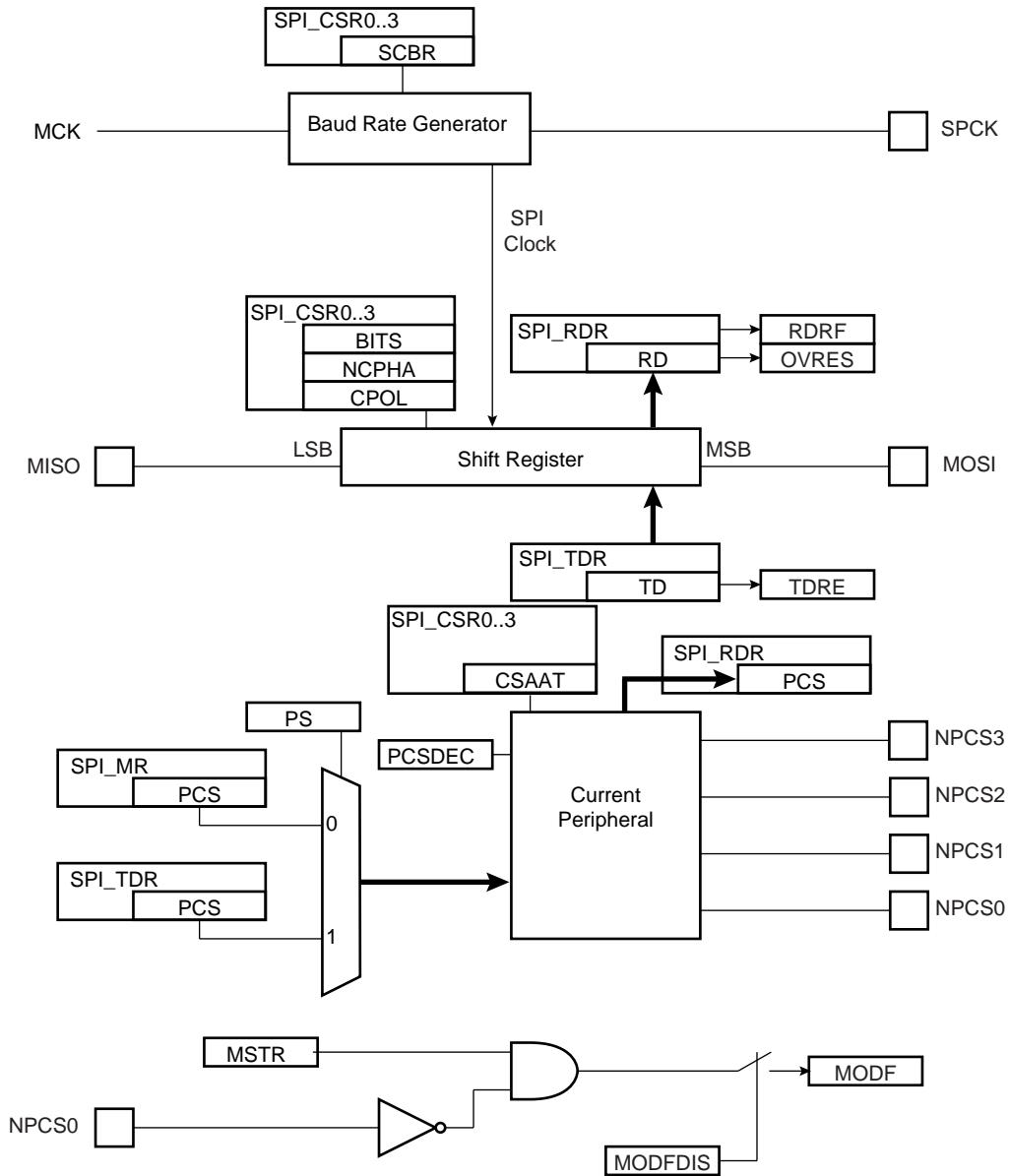
The transfer of received data from the Shift Register in SPI\_RDR is indicated by the RDRF bit (Receive Data Register Full) in the Status Register (SPI\_SR). When the received data is read, the RDRF bit is cleared.

If the SPI\_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error bit (OVRES) in SPI\_SR is set. As long as this flag is set, data is loaded in SPI\_RDR. The user has to read the status register to clear the OVRES bit.

[Figure 34-6](#), shows a block diagram of the SPI when operating in Master Mode. [Figure 34-7 on page 729](#) shows a flow chart describing how transfers are handled.

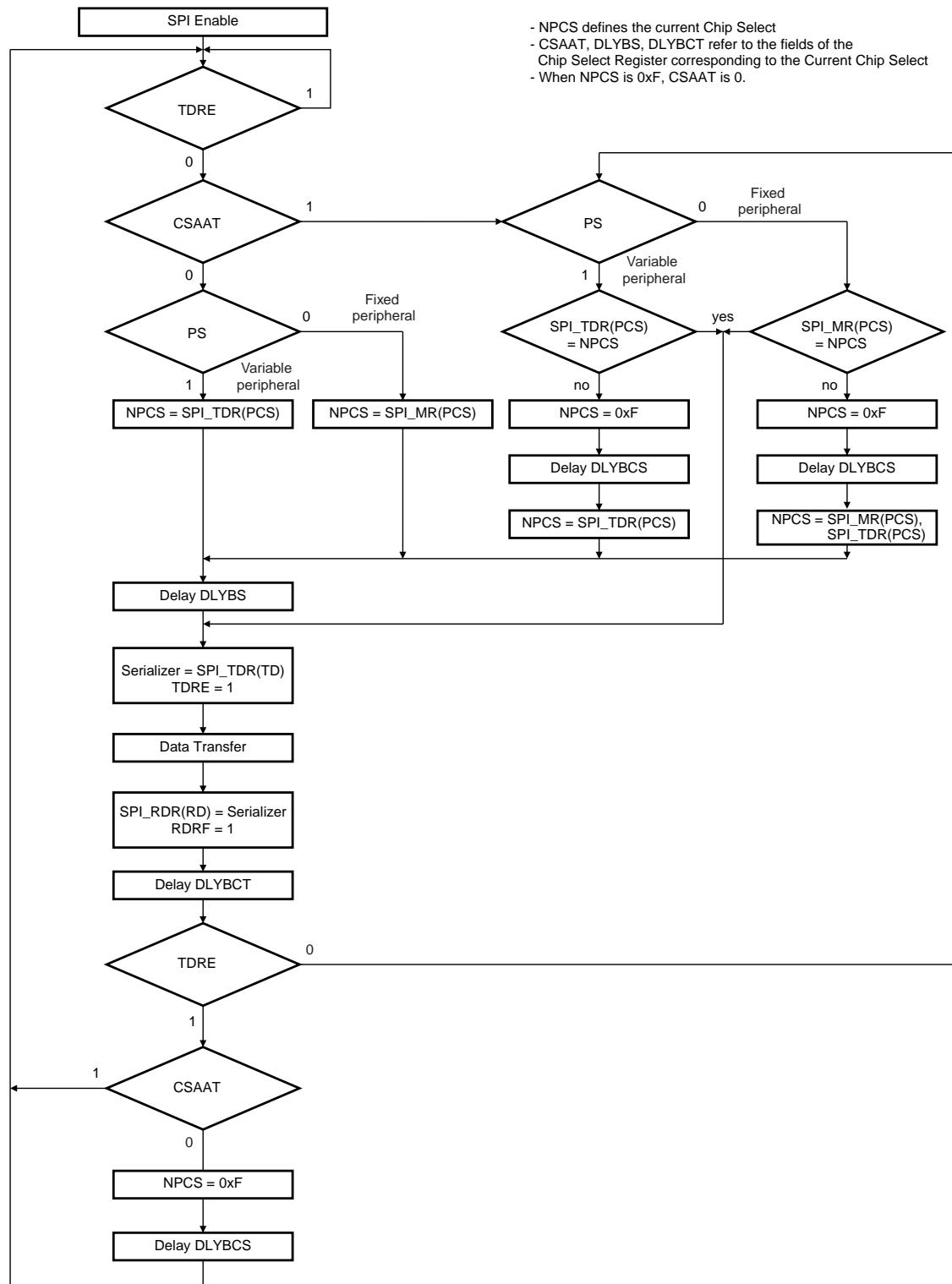
### 34.7.3.1 Master Mode Block Diagram

**Figure 34-6. Master Mode Block Diagram**



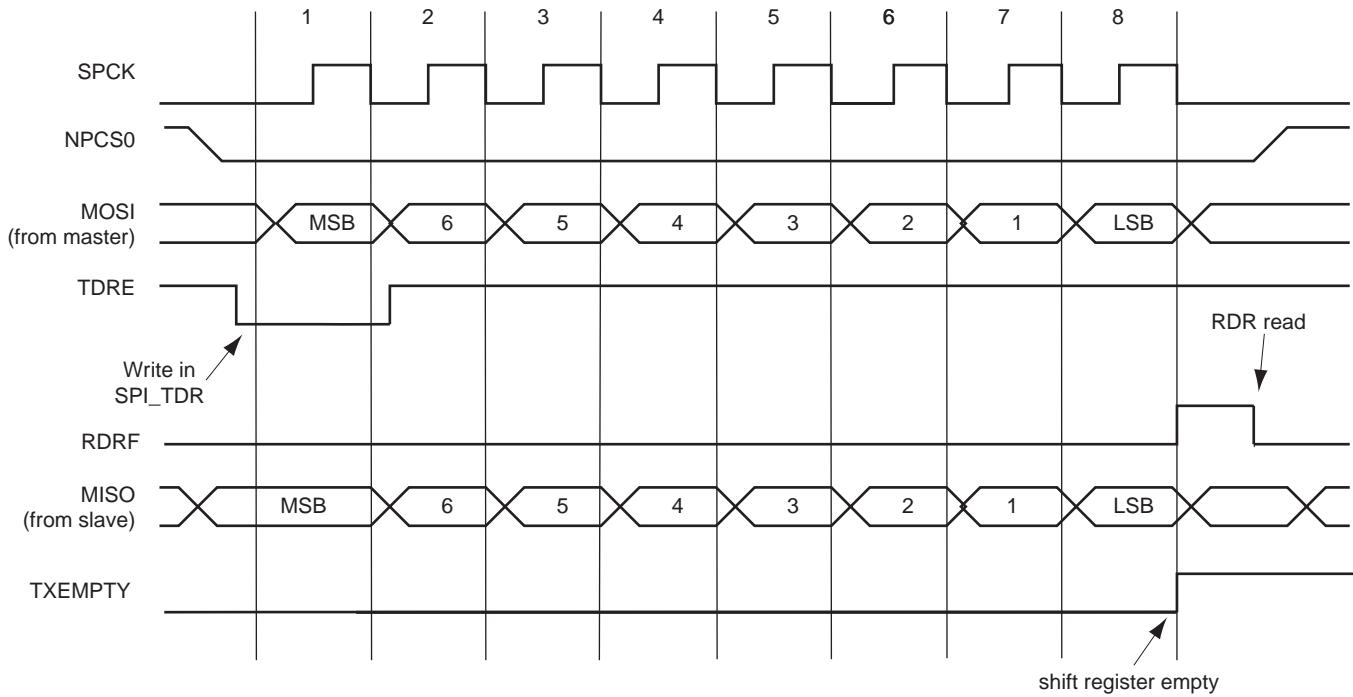
### 34.7.3.2 Master Mode Flow Diagram

**Figure 34-7. Master Mode Flow Diagram**



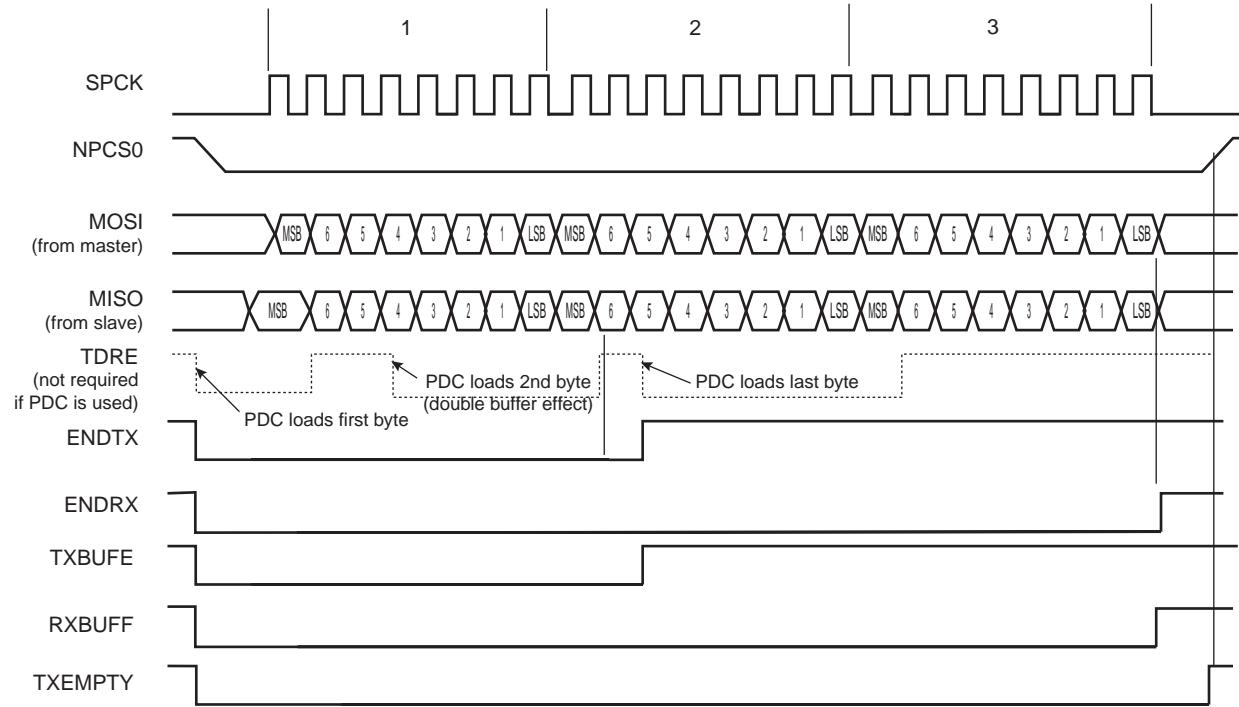
[Figure 34-8](#) shows Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags behavior within the SPI\_SR (Status Register) during an 8-bit data transfer in fixed mode and no Peripheral Data Controller involved.

**Figure 34-8. Status Register Flags Behavior**



[Figure 34-9](#) shows Transmission Register Empty (TXEMPTY), End of RX buffer (ENDRX), End of TX buffer (ENDTX), RX Buffer Full (RXBUFF) and TX Buffer Empty (TXBUFE) status flags behavior within the SPI\_SR (Status Register) during an 8-bit data transfer in fixed mode with the Peripheral Data Controller involved. The PDC is programmed to transfer and receive three data. The next pointer and counter are not used. The RDRF and TDRE are not shown because these flags are managed by the PDC when using the PDC.

**Figure 34-9. PDC Status Register Flags Behavior**



#### 34.7.3.3 Clock Generation

The SPI Baud rate clock is generated by dividing the Master Clock (MCK), by a value between 1 and 255.

This allows a maximum operating baud rate at up to Master Clock and a minimum operating baud rate of MCK divided by 255.

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

The divisor can be defined independently for each chip select, as it has to be programmed in the SCBR field of the Chip Select Registers. This allows the SPI to automatically adapt the baud rate for each interfaced peripheral without reprogramming.

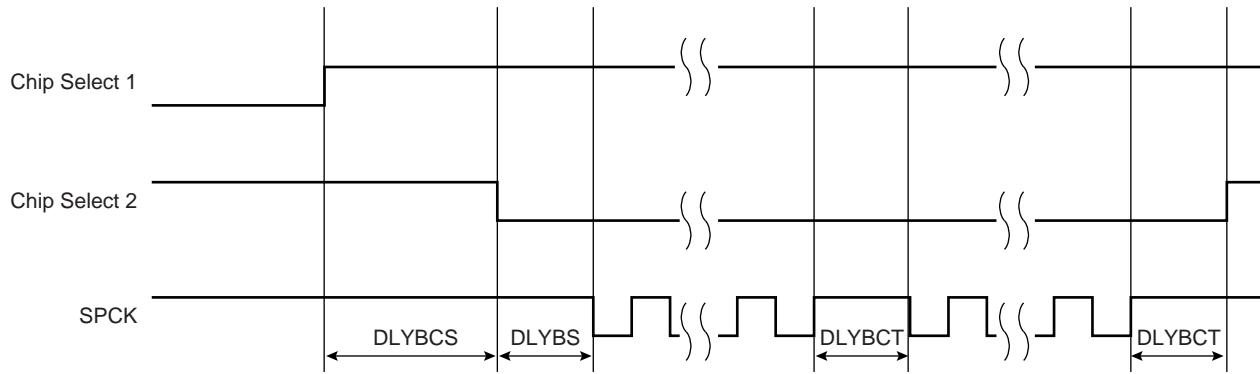
#### 34.7.3.4 Transfer Delays

Figure 34-10 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

- The delay between chip selects, programmable only once for all the chip selects by writing the DLYBCS field in the Mode Register. Allows insertion of a delay between release of one chip select and before assertion of a new one.
- The delay before SPCK, independently programmable for each chip select by writing the field DLYBS. Allows the start of SPCK to be delayed after the chip select has been asserted.
- The delay between consecutive transfers, independently programmable for each chip select by writing the DLYBCT field. Allows insertion of a delay between two transfers occurring on the same chip select

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

**Figure 34-10. Programmable Delays**



### 34.7.3.5 Peripheral Selection

The serial peripherals are selected through the assertion of the NPCS0 to NPCS3 signals. By default, all the NPCS signals are high before and after each transfer.

- Fixed Peripheral Select: SPI exchanges data with only one peripheral

Fixed Peripheral Select is activated by writing the PS bit to zero in SPI\_MR (Mode Register). In this case, the current peripheral is defined by the PCS field in SPI\_MR and the PCS field in the SPI\_TDR has no effect.

- Variable Peripheral Select: Data can be exchanged with more than one peripheral without having to reprogram the NPCS field in the SPI\_MR register.

Variable Peripheral Select is activated by setting PS bit to one. The PCS field in SPI\_TDR is used to select the current peripheral. This means that the peripheral selection can be defined for each new data. The value to write in the SPI\_TDR register as the following format.

[xxxxxxxx(7-bit) + LASTXFER(1-bit)<sup>(1)</sup> + xxxx(4-bit) + PCS (4-bit) + DATA (8 to 16-bit)] with PCS equals to the chip select to assert as defined in [Section 34.8.4](#) (SPI Transmit Data Register) and LASTXFER bit at 0 or 1 depending on CSAAT bit.

Note: 1. Optional.

CSAAT, LASTXFER and CSNAAT bits are discussed in [Section 34.7.3.10 "Peripheral Deselection with PDC"](#) and [Section 34.7.3.11 "Peripheral Deselection with DMAC"](#).

If LASTXFER is used, the command must be issued before writing the last character. Instead of LASTXFER, the user can use the SPIDIS command. After the end of the DMA or PDC transfer, wait for the TXEMPTY flag, then write SPIDIS into the SPI\_CR register (this will not change the configuration register values); the NPCS will be deactivated after the last character transfer. Then, another DMA or PDC transfer can be started if the SPIEN was previously written in the SPI\_CR register.

### 34.7.3.6 SPI Peripheral DMA Controller (PDC)

In both fixed and variable mode the Peripheral DMA Controller (PDC) can be used to reduce processor overhead.

The Fixed Peripheral Selection allows buffer transfers with a single peripheral. Using the PDC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, changing the peripheral selection requires the Mode Register to be reprogrammed.

The Variable Peripheral Selection allows buffer transfers with multiple peripherals without reprogramming the Mode Register. Data written in SPI\_TDR is 32 bits wide and defines the real data to be transmitted and the peripheral it is destined to. Using the PDC in this mode requires 32-bit wide buffers, with the data in the LSBs and the PCS and LASTXFER fields in the MSBs, however the SPI still controls the number of bits (8 to 16) to be transferred through MISO and MOSI lines with the chip select configuration registers. This is not the optimal means in term of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

Transfer Size

Depending on the data size to transmit, from 8 to 16 bits, the PDC manages automatically the type of pointer's size it has to point to. The PDC will perform the following transfer size depending on the mode and number of bits per data.

#### Fixed Mode:

- 8-bit Data:  
Byte transfer,  
PDC Pointer Address = Address + 1 byte,  
PDC Counter = Counter - 1
- 8-bit to 16-bit Data:  
2 bytes transfer. n-bit data transfer with don't care data (MSB) filled with 0's,  
PDC Pointer Address = Address + 2 bytes,  
PDC Counter = Counter - 1

#### Variable Mode:

In variable Mode, PDC Pointer Address = Address +4 bytes and PDC Counter = Counter - 1 for 8 to 16-bit transfer size. When using the PDC, the TDRE and RDRF flags are handled by the PDC, thus the user's application does not have to check those bits. Only End of RX Buffer (ENDRX), End of TX Buffer (ENDTX), Buffer Full (RXBUFF), TX Buffer Empty (TXBUFE) are significant. For further details about the Peripheral DMA Controller and user interface, refer to the PDC section of the product datasheet.

### 34.7.3.7 SPI Direct Access Memory Controller (DMAC)

In both fixed and variable mode the Direct Memory Access Controller (DMAC) can be used to reduce processor overhead.

The Fixed Peripheral Selection allows buffer transfers with a single peripheral. Using the DMAC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, changing the peripheral selection requires the Mode Register to be reprogrammed.

The Variable Peripheral Selection allows buffer transfers with multiple peripherals without reprogramming the Mode Register. Data written in SPI\_TDR is 32 bits wide and defines the real data to be transmitted and the peripheral it is destined to. Using the DMAC in this mode requires 32-bit wide buffers, with the data in the LSBs and the PCS and LASTXFER fields in the MSBs, however the SPI still controls the number of bits (8 to16) to be transferred through MISO and MOSI lines with the chip select configuration registers. This is not the optimal means in term of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

### 34.7.3.8 Peripheral Chip Select Decoding

The user can program the SPI to operate with up to 15 peripherals by decoding the four Chip Select lines, NPCS0 to NPCS3 with 1 of up to 16 decoder/demultiplexer. This can be enabled by writing the PCSDEC bit at 1 in the Mode Register (SPI\_MR).

When operating without decoding, the SPI makes sure that in any case only one chip select line is activated, i.e., one NPCS line driven low at a time. If two bits are defined low in a PCS field, only the lowest numbered chip select is driven low.

When operating with decoding, the SPI directly outputs the value defined by the PCS field on NPCS lines of either the Mode Register or the Transmit Data Register (depending on PS).

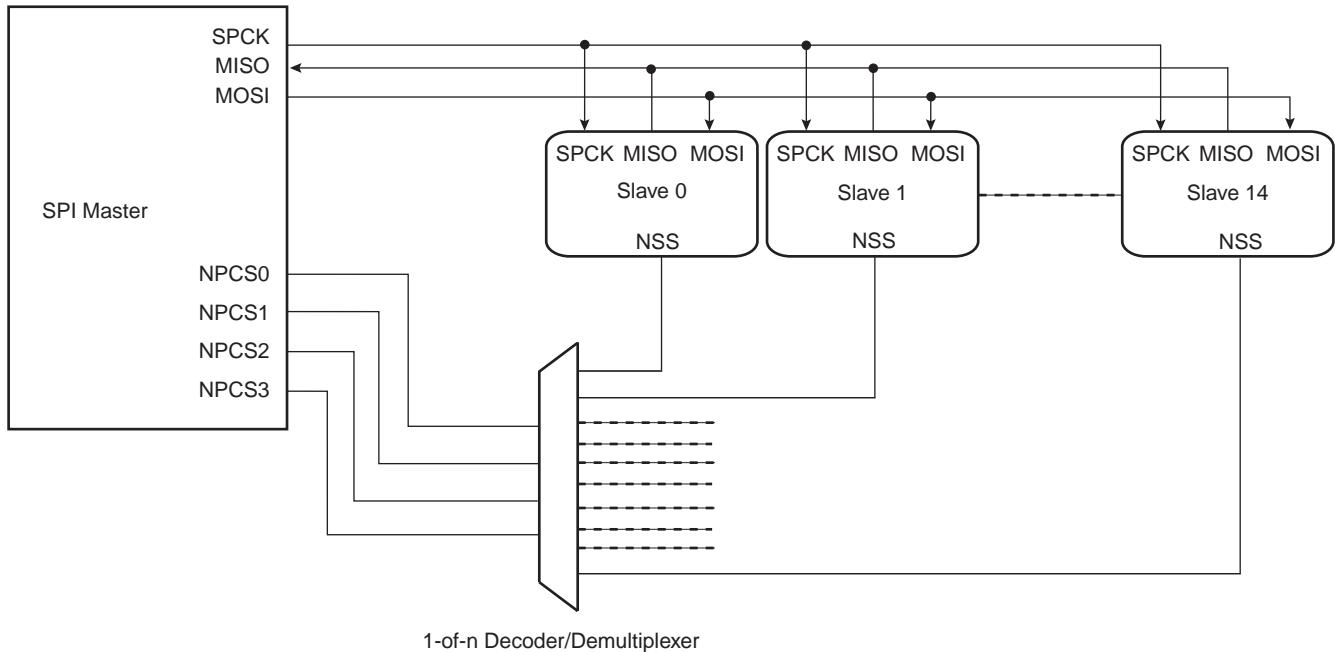
As the SPI sets a default value of 0xF on the chip select lines (i.e. all chip select lines at 1) when not processing any transfer, only 15 peripherals can be decoded.

The SPI has only four Chip Select Registers, not 15. As a result, when decoding is activated, each chip select defines the characteristics of up to four peripherals. As an example, SPI\_CRS0 defines the characteristics of the externally decoded peripherals 0 to 3, corresponding to the PCS values 0x0 to 0x3. Thus, the user has to make sure to connect compatible peripherals on the decoded chip select lines 0 to 3, 4 to 7, 8 to 11 and 12 to 14. [Figure 34-11](#) below shows such an implementation.

If the CSAAT bit is used, with or without the PDC, the Mode Fault detection for NPCS0 line must be disabled. This is not needed for all other chip select lines since Mode Fault Detection is only on NPCS0.

If the CSAAT bit is used, with or without the DMAC, the Mode Fault detection for NPCS0 line must be disabled. This is not needed for all other chip select lines since Mode Fault Detection is only on NPCS0.

**Figure 34-11. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation**



#### 34.7.3.9 Peripheral Deselection without DMA nor PDC

During a transfer of more than one data on a Chip Select without the DMA nor PDC, the SPI\_TDR is loaded by the processor, the flag TDRE rises as soon as the content of the SPI\_TDR is transferred into the internal shift register. When this flag is detected high, the SPI\_TDR can be reloaded. If this reload by the processor occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. But depending on the application software handling the SPI status register flags (by interrupt or polling method) or servicing other interrupts or other tasks, the processor may not reload the SPI\_TDR in time to keep the chip select active (low). A null Delay Between Consecutive Transfer (DLYBCT) value in the SPI\_CSR register, will give even less time for the processor to reload the SPI\_TDR. With some SPI slave peripherals, requiring the chip select line to remain active (low) during a full set of transfers might lead to communication errors.

To facilitate interfacing with such devices, the Chip Select Register [CSR0...CSR3] can be programmed with the CSAAT bit (Chip Select Active After Transfer) at 1. This allows the chip select lines to remain in their current state (low = active) until transfer to another chip select is required. Even if the SPI\_TDR is not reloaded the chip select will remain active. To have the chip select line to raise at the end of the transfer the Last transfer Bit (LASTXFER) in the SPI\_MR register must be set at 1 before writing the last data to transmit into the SPI\_TDR.

#### 34.7.3.10 Peripheral Deselection with PDC

When the Peripheral DMA Controller is used, the chip select line will remain low during the whole transfer since the TDRE flag is managed by the PDC itself. The reloading of the SPI\_TDR by the PDC is done as soon as TDRE flag is set to one. In this case the use of CSAAT bit might not be needed. However, it may happen that when other PDC channels connected to other peripherals are in use as well, the SPI PDC might be delayed by another (PDC with a higher priority on the bus). Having PDC buffers in slower memories like flash memory or SDRAM compared to fast internal SRAM, may lengthen the reload time of the SPI\_TDR by the PDC as well. This means that the SPI\_TDR might not be reloaded in time to keep the chip select line low. In this case the chip select line may toggle between data transfer and according to some SPI Slave devices, the communication might get lost. The use of the CSAAT bit might be needed.

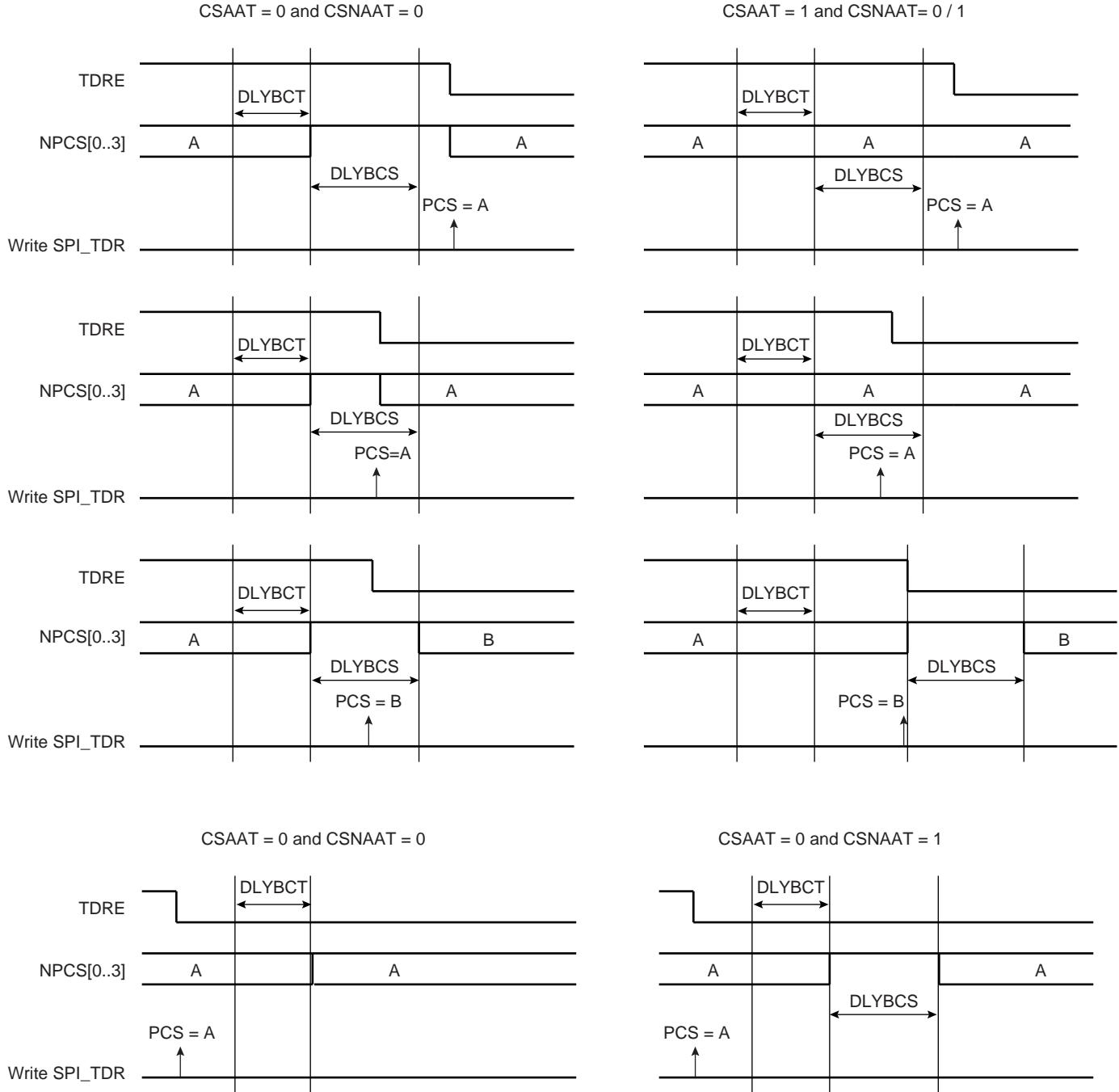
### 34.7.3.11 Peripheral Deselection with DMAC

When the Direct Memory Access Controller is used, the chip select line will remain low during the whole transfer since the TDRE flag is managed by the DMAC itself. The reloading of the SPI\_TDR by the DMAC is done as soon as TDRE flag is set to one. In this case the use of CSAAT bit might not be needed. However, it may happen that when other DMAC channels connected to other peripherals are in use as well, the SPI DMAC might be delayed by another (DMAC with a higher priority on the bus). Having DMAC buffers in slower memories like flash memory or SDRAM compared to fast internal SRAM, may lengthen the reload time of the SPI\_TDR by the DMAC as well. This means that the SPI\_TDR might not be reloaded in time to keep the chip select line low. In this case the chip select line may toggle between data transfer and according to some SPI Slave devices, the communication might get lost. The use of the CSAAT bit might be needed.

When the CSAAT bit is set at 0, the NPCS does not rise in all cases between two transfers on the same peripheral. During a transfer on a Chip Select, the flag TDRE rises as soon as the content of the SPI\_TDR is transferred into the internal shifter. When this flag is detected the SPI\_TDR can be reloaded. If this reload occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. This might lead to difficulties for interfacing with some serial peripherals requiring the chip select to be de-asserted after each transfer. To facilitate interfacing with such devices, the Chip Select Register can be programmed with the CSNAAT bit (Chip Select Not Active After Transfer) at 1. This allows to de-assert systematically the chip select lines during a time DLYBCS. (The value of the CSNAAT bit is taken into account only if the CSAAT bit is set at 0 for the same Chip Select).

Figure 34-12 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

**Figure 34-12. Peripheral Deselection**



#### 34.7.3.12 Mode Fault Detection

A mode fault is detected when the SPI is programmed in Master Mode and a low level is driven by an external master on the NPCS0/NSS signal. In this case, multi-master configuration, NPCS0, MOSI, MISO and SPCK pins must be configured in open drain (through the PIO controller). When a mode fault is detected, the MODF bit in the SPI\_SR is set until the SPI\_SR is read and the SPI is automatically disabled until re-enabled by writing the SPIEN bit in the SPI\_CR (Control Register) at 1.

By default, the Mode Fault detection circuitry is enabled. The user can disable Mode Fault detection by setting the MODFDIS bit in the SPI Mode Register (SPI\_MR).

#### 34.7.4 SPI Slave Mode

When operating in Slave Mode, the SPI processes data bits on the clock provided on the SPI clock pin (SPCK).

The SPI waits for NSS to go active before receiving the serial clock from an external master. When NSS falls, the clock is validated on the serializer, which processes the number of bits defined by the BITS field of the Chip Select Register 0 (SPI\_CSR0). These bits are processed following a phase and a polarity defined respectively by the NCPHA and CPOL bits of the SPI\_CSR0. Note that BITS, CPOL and NCPHA of the other Chip Select Registers have no effect when the SPI is programmed in Slave Mode.

The bits are shifted out on the MISO line and sampled on the MOSI line.

(For more information on BITS field, see also, the [\(Note:\)](#) below the register table; [Section 34.8.9 “SPI Chip Select Register” on page 750.](#))

When all the bits are processed, the received data is transferred in the Receive Data Register and the RDRF bit rises. If the SPI\_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error bit (OVRES) in SPI\_SR is set. As long as this flag is set, data is loaded in SPI\_RDR. The user has to read the status register to clear the OVRES bit.

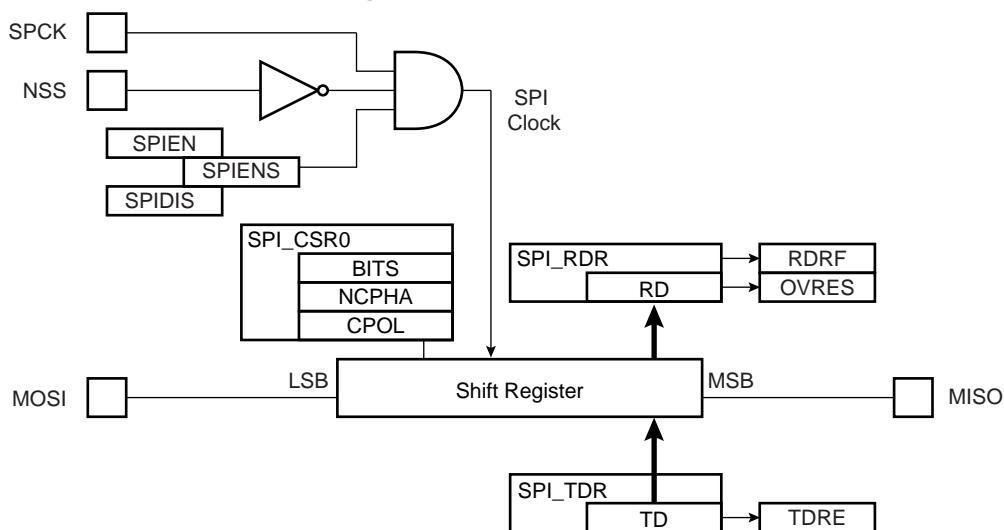
When a transfer starts, the data shifted out is the data present in the Shift Register. If no data has been written in the Transmit Data Register (SPI\_TDR), the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift Register resets at 0.

When a first data is written in SPI\_TDR, it is transferred immediately in the Shift Register and the TDRE bit rises. If new data is written, it remains in SPI\_TDR until a transfer occurs, i.e. NSS falls and there is a valid clock on the SPCK pin. When the transfer occurs, the last data written in SPI\_TDR is transferred in the Shift Register and the TDRE bit rises. This enables frequent updates of critical variables with single transfers.

Then, a new data is loaded in the Shift Register from the Transmit Data Register. In case no character is ready to be transmitted, i.e. no character has been written in SPI\_TDR since the last load from SPI\_TDR to the Shift Register, the SPI\_TDR is retransmitted. In this case the Underrun Error Status Flag (UNDES) is set in the SPI\_SR.

[Figure 34-13](#) shows a block diagram of the SPI when operating in Slave Mode.

**Figure 34-13. Slave Mode Functional Block Diagram**



### 34.7.5 Write Protected Registers

To prevent any single software error that may corrupt SPI behavior, the registers listed below can be write-protected by setting the WPEN bit in the SPI Write Protection Mode Register (SPI\_WPMR).

If a write access in a write-protected register is detected, then the WPVS flag in the SPI Write Protection Status Register (SPI\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is automatically reset after reading the SPI Write Protection Status Register (SPI\_WPSR).

List of the write-protected registers:

[Section 34.8.2 "SPI Mode Register"](#)

[Section 34.8.9 "SPI Chip Select Register"](#)

## 34.8 Serial Peripheral Interface (SPI) User Interface

Table 34-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	SPI_CR	Write-only	---
0x04	Mode Register	SPI_MR	Read-write	0x0
0x08	Receive Data Register	SPI_RDR	Read-only	0x0
0x0C	Transmit Data Register	SPI_TDR	Write-only	---
0x10	Status Register	SPI_SR	Read-only	0x000000F0
0x14	Interrupt Enable Register	SPI_IER	Write-only	---
0x18	Interrupt Disable Register	SPI_IDR	Write-only	---
0x1C	Interrupt Mask Register	SPI_IMR	Read-only	0x0
0x20 - 0x2C	Reserved			
0x30	Chip Select Register 0	SPI_CSR0	Read-write	0x0
0x34	Chip Select Register 1	SPI_CSR1	Read-write	0x0
0x38	Chip Select Register 2	SPI_CSR2	Read-write	0x0
0x3C	Chip Select Register 3	SPI_CSR3	Read-write	0x0
0x40 - 0xE0	Reserved	-	-	-
0xE4	Write Protection Control Register	SPI_WPMR	Read-write	0x0
0xE8	Write Protection Status Register	SPI_WPSR	Read-only	0x0
0x00EC - 0x00F8	Reserved	-	-	-
0x00FC	Reserved	-	-	-
0x100 - 0x124	Reserved for PDC Registers	-	-	-

### 34.8.1 SPI Control Register

**Name:** SPI\_CR

**Address:** 0x40088000

**Access:** Write-only

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

- **SPIEN: SPI Enable**

0 = No effect.

1 = Enables the SPI to transfer and receive data.

- **SPIDIS: SPI Disable**

0 = No effect.

1 = Disables the SPI.

As soon as SPIDIS is set, SPI finishes its transfer.

All pins are set in input mode and no data is received or transmitted.

If a transfer is in progress, the transfer is finished before the SPI is disabled.

If both SPIEN and SPIDIS are equal to one when the control register is written, the SPI is disabled.

- **SWRST: SPI Software Reset**

0 = No effect.

1 = Reset the SPI. A software-triggered hardware reset of the SPI interface is performed.

The SPI is in slave mode after software reset.

PDC channels are not affected by software reset.

- **LASTXFER: Last Transfer**

0 = No effect.

1 = The current NPCS will be deasserted after the character written in TD has been transferred. When CSAAT is set, this allows to close the communication with the current serial peripheral by raising the corresponding NPCS line as soon as TD transfer has completed.

Refer to [Section 34.7.3.5 "Peripheral Selection"](#) for more details.

### 34.8.2 SPI Mode Register

**Name:** SPI\_MR  
**Address:** 0x40088004  
**Access:** Read-write

31	30	29	28	27	26	25	24
DLYBCS							
23	22	21	20	19	18	17	16
—	—	—	—	—	PCS	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	—
7	6	5	4	3	2	1	0
LLB	—	WDRBT	MODFDIS	—	PCSDEC	PS	MSTR

This register can only be written if the WPEN bit is cleared in "SPI Write Protection Mode Register".

- **MSTR: Master/Slave Mode**

0 = SPI is in Slave mode.

1 = SPI is in Master mode.

- **PS: Peripheral Select**

0 = Fixed Peripheral Select.

1 = Variable Peripheral Select.

- **PCSDEC: Chip Select Decode**

0 = The chip selects are directly connected to a peripheral device.

1 = The four chip select lines are connected to a 4- to 16-bit decoder.

When PCSDEC equals one, up to 15 Chip Select signals can be generated with the four lines using an external 4- to 16-bit decoder. The Chip Select Registers define the characteristics of the 15 chip selects according to the following rules:

SPI\_CSR0 defines peripheral chip select signals 0 to 3.

SPI\_CSR1 defines peripheral chip select signals 4 to 7.

SPI\_CSR2 defines peripheral chip select signals 8 to 11.

SPI\_CSR3 defines peripheral chip select signals 12 to 14.

- **MODFDIS: Mode Fault Detection**

0 = Mode fault detection is enabled.

1 = Mode fault detection is disabled.

- **WDRBT: Wait Data Read Before Transfer**

0 = No Effect. In master mode, a transfer can be initiated whatever the state of the Receive Data Register is.

1 = In Master Mode, a transfer can start only if the Receive Data Register is empty, i.e. does not contain any unread data. This mode prevents overrun error in reception.

- **LLB: Local Loopback Enable**

0 = Local loopback path disabled.

1 = Local loopback path enabled

LLB controls the local loopback on the data serializer for testing in Master Mode only. (MISO is internally connected on MOSI.)

- **PCS: Peripheral Chip Select**

This field is only used if Fixed Peripheral Select is active (PS = 0).

If PCSDEC = 0:

PCS = xxx0	NPCS[3:0] = 1110
PCS = xx01	NPCS[3:0] = 1101
PCS = x011	NPCS[3:0] = 1011
PCS = 0111	NPCS[3:0] = 0111
PCS = 1111	forbidden (no peripheral is selected)

(x = don't care)

If PCSDEC = 1:

NPCS[3:0] output signals = PCS.

- **DLYBCS: Delay Between Chip Selects**

This field defines the delay from NPCS inactive to the activation of another NPCS. The DLYBCS time guarantees non-overlapping chip selects and solves bus contentions in case of peripherals having long data float times.

If DLYBCS is less than or equal to six, six MCK periods will be inserted by default.

Otherwise, the following equation determines the delay:

$$\text{Delay Between Chip Selects} = \frac{DLYBCS}{MCK}$$

### 34.8.3 SPI Receive Data Register

**Name:** SPI\_RDR

**Address:** 0x40088008

**Access:** Read-only

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

- **RD: Receive Data**

Data received by the SPI Interface is stored in this register right-justified. Unused bits read zero.

- **PCS: Peripheral Chip Select**

In Master Mode only, these bits indicate the value on the NPCS pins at the end of a transfer. Otherwise, these bits read zero.

Note: When using variable peripheral select mode (PS = 1 in SPI\_MR) it is mandatory to also set the WDRBT field to 1 if the SPI\_RDR PCS field is to be processed.

#### 34.8.4 SPI Transmit Data Register

**Name:** SPI\_TDR

**Address:** 0x4008800C

**Access:** Write-only

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

- **TD: Transmit Data**

Data to be transmitted by the SPI Interface is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

- **PCS: Peripheral Chip Select**

This field is only used if Variable Peripheral Select is active (PS = 1).

If PCSDEC = 0:

PCS = xxx0	NPCS[3:0] = 1110
PCS = xx01	NPCS[3:0] = 1101
PCS = x011	NPCS[3:0] = 1011
PCS = 0111	NPCS[3:0] = 0111
PCS = 1111	forbidden (no peripheral is selected)
(x = don't care)	

If PCSDEC = 1:

NPCS[3:0] output signals = PCS

- **LASTXFER: Last Transfer**

0 = No effect.

1 = The current NPCS will be deasserted after the character written in TD has been transferred. When CSAAT is set, this allows to close the communication with the current serial peripheral by raising the corresponding NPCS line as soon as TD transfer has completed.

This field is only used if Variable Peripheral Select is active (PS = 1).

### 34.8.5 SPI Status Register

**Name:** SPI\_SR  
**Address:** 0x40088010  
**Access:** Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	SPIENS
15	14	13	12	11	10	9	8
—	—	—	—	—	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full**

0 = No data has been received since the last read of SPI\_RDR

1 = Data has been received and the received data has been transferred from the serializer to SPI\_RDR since the last read of SPI\_RDR.

- **TDRE: Transmit Data Register Empty**

0 = Data has been written to SPI\_TDR and not yet transferred to the serializer.

1 = The last data written in the Transmit Data Register has been transferred to the serializer.

TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to one.

- **MODF: Mode Fault Error**

0 = No Mode Fault has been detected since the last read of SPI\_SR.

1 = A Mode Fault occurred since the last read of the SPI\_SR.

- **OVRES: Overrun Error Status**

0 = No overrun has been detected since the last read of SPI\_SR.

1 = An overrun has occurred since the last read of SPI\_SR.

An overrun occurs when SPI\_RDR is loaded at least twice from the serializer since the last read of the SPI\_RDR.

- **ENDRX: End of RX buffer**

0 = The Receive Counter Register has not reached 0 since the last write in SPI\_RCR<sup>(1)</sup> or SPI\_RNCR<sup>(1)</sup>.

1 = The Receive Counter Register has reached 0 since the last write in SPI\_RCR<sup>(1)</sup> or SPI\_RNCR<sup>(1)</sup>.

- **ENDTX: End of TX buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in SPI\_TCR<sup>(1)</sup> or SPI\_TNCR<sup>(1)</sup>.

1 = The Transmit Counter Register has reached 0 since the last write in SPI\_TCR<sup>(1)</sup> or SPI\_TNCR<sup>(1)</sup>.

- **RXBUFF: RX Buffer Full**

0 = SPI\_RCR<sup>(1)</sup> or SPI\_RNCR<sup>(1)</sup> has a value other than 0.

1 = Both SPI\_RCR<sup>(1)</sup> and SPI\_RNCR<sup>(1)</sup> have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = SPI\_TCR<sup>(1)</sup> or SPI\_TNCR<sup>(1)</sup> has a value other than 0.

1 = Both SPI\_TCR<sup>(1)</sup> and SPI\_TNCR<sup>(1)</sup> have a value of 0.

- **NSSR: NSS Rising**

0 = No rising edge detected on NSS pin since last read.

1 = A rising edge occurred on NSS pin since last read.

- **TXEMPTY: Transmission Registers Empty**

0 = As soon as data is written in SPI\_TDR.

1 = SPI\_TDR and internal shifter are empty. If a transfer delay has been defined, TXEMPTY is set after the completion of such delay.

- **UNDES: Underrun Error Status (Slave Mode Only)**

0 = No underrun has been detected since the last read of SPI\_SR.

1 = A transfer begins whereas no data has been loaded in the Transmit Data Register.

- **SPIENS: SPI Enable Status**

0 = SPI is disabled.

1 = SPI is enabled.

Note: 1. SPI\_RCR, SPI\_RNCR, SPI\_TCR, SPI\_TNCR are physically located in the PDC.

### 34.8.6 SPI Interrupt Enable Register

**Name:** SPI\_IER

**Address:** 0x40088014

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

0 = No effect.

1 = Enables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Enable**
- **MODF: Mode Fault Error Interrupt Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **NSSR: NSS Rising Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **UNDES: Underrun Error Interrupt Enable**

### 34.8.7 SPI Interrupt Disable Register

**Name:** SPI\_IDR

**Address:** 0x40088018

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

0 = No effect.

1 = Disables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Disable**
- **MODF: Mode Fault Error Interrupt Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **NSSR: NSS Rising Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **UNDES: Underrun Error Interrupt Disable**

### 34.8.8 SPI Interrupt Mask Register

**Name:** SPI\_IMR  
**Address:** 0x4008801C  
**Access:** Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

- **RDRF:** Receive Data Register Full Interrupt Mask
- **TDRE:** SPI Transmit Data Register Empty Interrupt Mask
- **MODF:** Mode Fault Error Interrupt Mask
- **OVRES:** Overrun Error Interrupt Mask
- **ENDRX:** End of Receive Buffer Interrupt Mask
- **ENDTX:** End of Transmit Buffer Interrupt Mask
- **RXBUFF:** Receive Buffer Full Interrupt Mask
- **TXBUFE:** Transmit Buffer Empty Interrupt Mask
- **NSSR:** NSS Rising Interrupt Mask
- **TXEMPTY:** Transmission Registers Empty Mask
- **UNDES:** Underrun Error Interrupt Mask

### 34.8.9 SPI Chip Select Register

**Name:** SPI\_CSRx[x=0..3]

**Address:** 0x40088030

**Access:** Read/Write

31	30	29	28	27	26	25	24
DLYBCT							
23	22	21	20	19	18	17	16
DLYBS							
15	14	13	12	11	10	9	8
SCBR							
7	6	5	4	3	2	1	0
BITS				CSAAT	CSNAAT	NCPHA	CPOL

This register can only be written if the WPEN bit is cleared in "[SPI Write Protection Mode Register](#)".

Note: SPI\_CSRx registers must be written even if the user wants to use the defaults. The BITS field will not be updated with the translated value unless the register is written.

- **CPOL: Clock Polarity**

0 = The inactive state value of SPCK is logic level zero.

1 = The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with NCPHA to produce the required clock/data relationship between master and slave devices.

- **NCPHA: Clock Phase**

0 = Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1 = Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

NCPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. NCPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CSAAT: Chip Select Not Active After Transfer (Ignored if CSAAT = 1)**

0 = The Peripheral Chip Select does not rise between two transfers if the SPI\_TDR is reloaded before the end of the first transfer and if the two transfers occur on the same Chip Select.

1 = The Peripheral Chip Select rises systematically after each transfer performed on the same slave. It remains active after the end of transfer for a minimal duration of:

$$-\frac{DLYBCT}{MCK} \text{ (if DLYBCT field is different from 0)}$$

$$-\frac{DLYBCT + 1}{MCK} \text{ (if DLYBCT field equals 0)}$$

- **CSAAT: Chip Select Active After Transfer**

0 = The Peripheral Chip Select Line rises as soon as the last transfer is achieved.

1 = The Peripheral Chip Select does not rise after the last transfer is achieved. It remains active until a new transfer is requested on a different chip select.

- **BITS: Bits Per Transfer**

(See the [\(Note\)](#) below the register table; [Section 34.8.9 “SPI Chip Select Register” on page 750](#).)

The BITS field determines the number of data bits transferred. Reserved values should not be used.

Value	Name	Description
0	8_BIT	8 bits for transfer
1	9_BIT	9 bits for transfer
2	10_BIT	10 bits for transfer
3	11_BIT	11 bits for transfer
4	12_BIT	12 bits for transfer
5	13_BIT	13 bits for transfer
6	14_BIT	14 bits for transfer
7	15_BIT	15 bits for transfer
8	16_BIT	16 bits for transfer
9	–	Reserved
10	–	Reserved
11	–	Reserved
12	–	Reserved
13	–	Reserved
14	–	Reserved
15	–	Reserved

- **SCBR: Serial Clock Baud Rate**

In Master Mode, the SPI Interface uses a modulus counter to derive the SPCK baud rate from the Master Clock MCK. The Baud rate is selected by writing a value from 1 to 255 in the SCBR field. The following equations determine the SPCK baud rate:

$$\text{SPCK Baudrate} = \frac{MCK}{SCBR}$$

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

Note: If one of the SCBR fields in SPI\_CSRx is set to 1, the other SCBR fields in SPI\_CSRx must be set to 1 as well, if they are required to process transfers. If they are not used to transfer data, they can be set at any value.

- **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS valid to the first valid SPCK transition.

When DLYBS equals zero, the NPCS valid to SPCK transition is 1/2 the SPCK clock period.

Otherwise, the following equations determine the delay:

$$\text{Delay Before SPCK} = \frac{DLYBS}{MCK}$$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT equals zero, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers.

Otherwise, the following equation determines the delay:

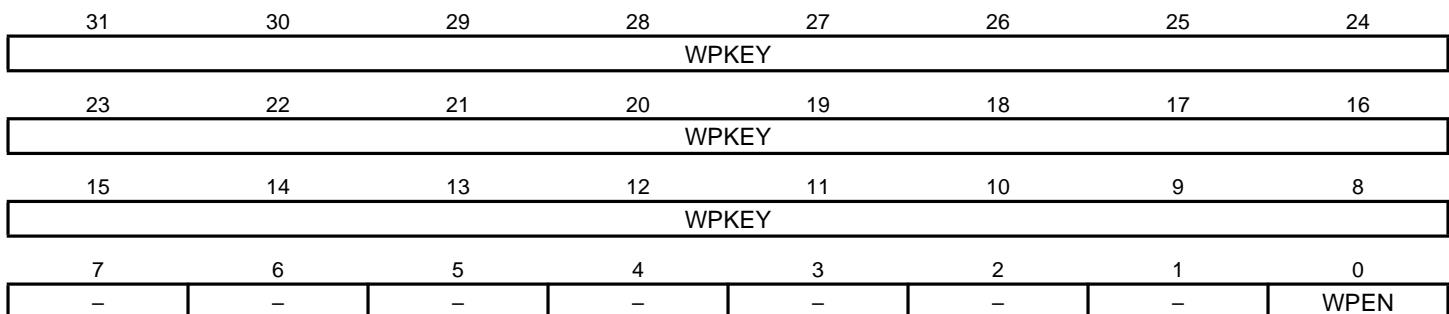
$$\text{Delay Between Consecutive Transfers} = \frac{32 \times DLYBCT}{MCK}$$

### 34.8.10 SPI Write Protection Mode Register

**Name:** SPI\_WPMR

**Address:** 0x400880E4

**Access:** Read-write



- **WPEN: Write Protect Enable**

0: Disables the Write Protect if WPKEY corresponds to 0x535049 ("SPI" in ASCII).

1: Enables the Write Protect if WPKEY corresponds to 0x535049 ("SPI" in ASCII).

Protects the registers:

- [Section 34.8.2 "SPI Mode Register"](#)
- [Section 34.8.9 "SPI Chip Select Register"](#)

- **WPKEY: Write Protect Key**

Value	Name	Description
0x535049	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 34.8.11 SPI Write Protection Status Register

**Name:** SPI\_WPSR

**Address:** 0x400880E8

**Access:** Read-only

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

- **WPVS: Write Protection Violation Status**

0 = No Write Protect Violation has occurred since the last read of the SPI\_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the SPI\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

This Field indicates the APB Offset of the register concerned by the violation (SPI\_MR or SPI\_CSRx)



## 35. Two-wire Interface (TWI)

### 35.1 Description

The Atmel Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 Kbits per second, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and I<sup>2</sup>C compatible device such as Real Time Clock (RTC), Dot Matrix/Graphic LCD Controllers and Temperature Sensor, to name but a few. The TWI is programmable as a master or a slave with sequential or single-byte access. Multiple master capability is supported.

Arbitration of the bus is performed internally and puts the TWI in slave mode automatically if the bus arbitration is lost.

A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

Below, [Table 35-1](#) lists the compatibility level of the Atmel Two-wire Interface in Master Mode and a full I<sup>2</sup>C compatible device.

**Table 35-1. Atmel TWI compatibility with I<sup>2</sup>C Standard**

I <sup>2</sup> C Standard	Atmel TWI
Standard Mode Speed (100 KHz)	Supported
Fast Mode Speed (400 KHz)	Supported
7 or 10 bits Slave Addressing	Supported
START BYTE <sup>(1)</sup>	Not Supported
Repeated Start (Sr) Condition	Supported
ACK and NACK Management	Supported
Slope control and input filtering (Fast mode)	Not Supported
Clock stretching	Supported
Multi Master Capability	Supported

Note: 1. START + b000000001 + Ack + Sr

### 35.2 Embedded Characteristics

- 2 TWIs
- Compatible with Atmel Two-wire Interface Serial Memory and I<sup>2</sup>C Compatible Devices<sup>(1)</sup>
- One, Two or Three Bytes for Slave Address
- Sequential Read-write Operations
- Master, Multi-master and Slave Mode Operation
- Bit Rate: Up to 400 Kbits
- General Call Supported in Slave mode
- SMBUS Quick Command Supported in Master Mode
- Connection to Peripheral DMA Controller (PDC) Channel Capabilities Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter

Note: 1. See [Table 35-1](#) for details on compatibility with I<sup>2</sup>C Standard.

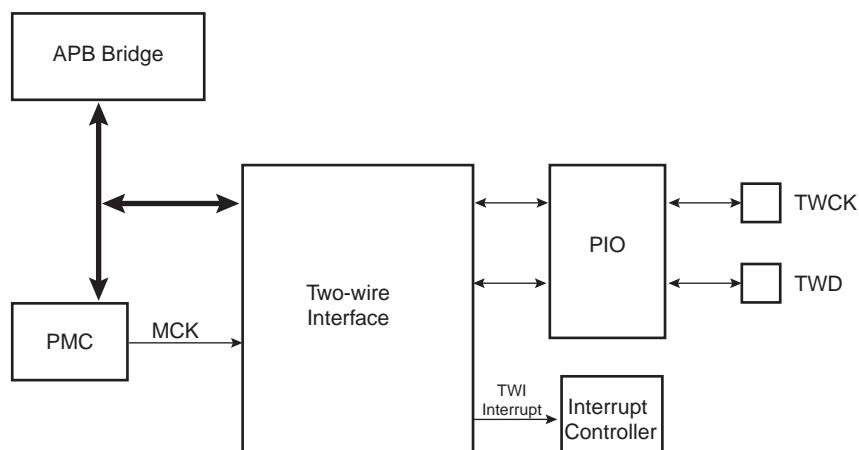
### 35.3 List of Abbreviations

Table 35-2. Abbreviations

Abbreviation	Description
TWI	Two-wire Interface
A	Acknowledge
NA	Non Acknowledge
P	Stop
S	Start
Sr	Repeated Start
SADR	Slave Address
ADR	Any address except SADR
R	Read
W	Write

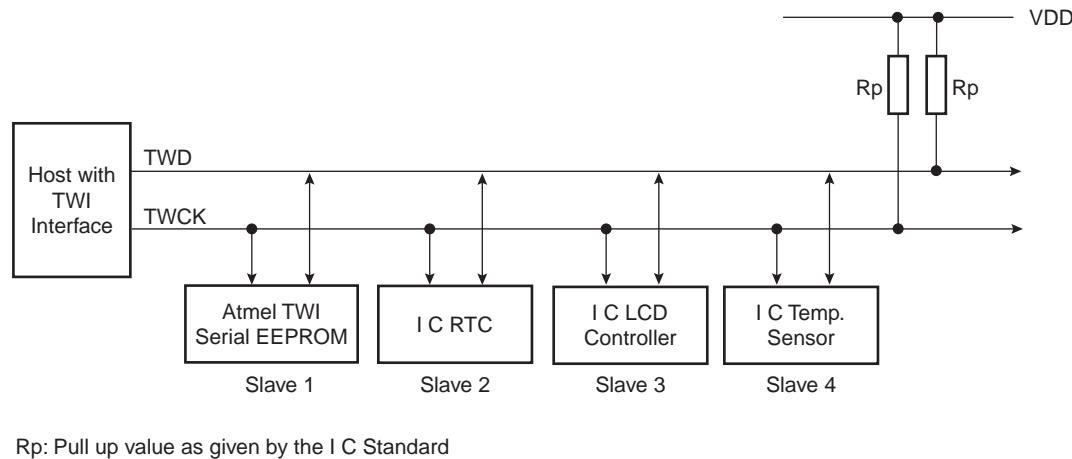
### 35.4 Block Diagram

Figure 35-1. Block Diagram



## 35.5 Application Block Diagram

Figure 35-2. Application Block Diagram



### 35.5.1 I/O Lines Description

Table 35-3. I/O Lines Description

Pin Name	Pin Description	Type
TWD	Two-wire Serial Data	Input/Output
TWCK	Two-wire Serial Clock	Input/Output

## 35.6 Product Dependencies

### 35.6.1 I/O Lines

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor (see [Figure 35-2 on page 757](#)). When the bus is free, both lines are high. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function.

TWD and TWCK pins may be multiplexed with PIO lines. To enable the TWI, the programmer must perform the following step:

- Program the PIO controller to dedicate TWD and TWCK as peripheral lines.

The user must not program TWD and TWCK as open-drain. It is already done by the hardware.

Table 35-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TWI0	TWCK0	PA4	A
TWI0	TWD0	PA3	A
TWI1	TWCK1	PB5	A
TWI1	TWD1	PB4	A

### 35.6.2 Power Management

- Enable the peripheral clock.

The TWI interface may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the TWI clock.

### 35.6.3 Interrupt

The TWI interface has an interrupt line connected to the Interrupt Controller. In order to handle interrupts, the Interrupt Controller must be programmed before configuring the TWI.

**Table 35-5. Peripheral IDs**

Instance	ID
TWI0	17
TWI1	18

## 35.7 Functional Description

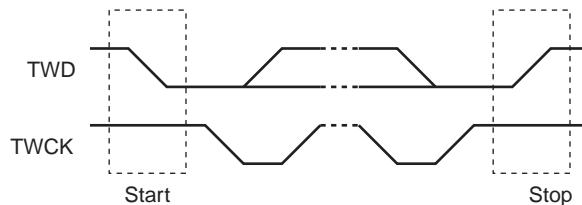
### 35.7.1 Transfer Format

The data put on the TWD line must be 8 bits long. Data is transferred MSB first; each byte must be followed by an acknowledgement. The number of bytes per transfer is unlimited (see [Figure 35-4](#)).

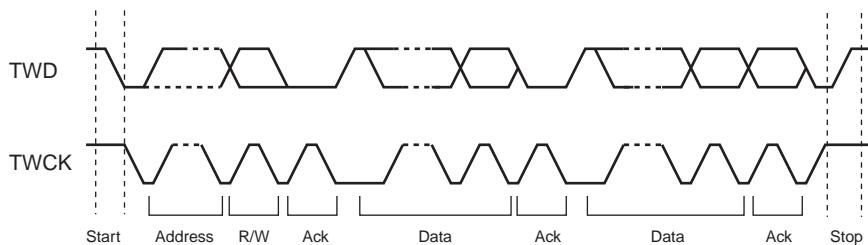
Each transfer begins with a START condition and terminates with a STOP condition (see [Figure 35-3](#)).

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines a STOP condition.

**Figure 35-3. START and STOP Conditions**



**Figure 35-4. Transfer Format**



### 35.7.2 Modes of Operation

The TWI has different modes of operations:

- Master transmitter mode
- Master receiver mode
- Multi-master transmitter mode
- Multi-master receiver mode
- Slave transmitter mode
- Slave receiver mode

These modes are described in the following chapters.

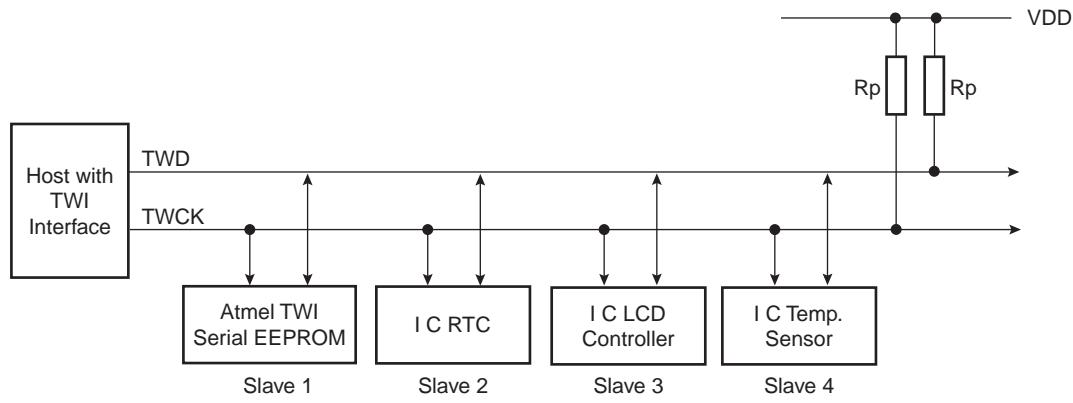
## 35.8 Master Mode

### 35.8.1 Definition

The Master is the device that starts a transfer, generates a clock and stops it.

### 35.8.2 Application Block Diagram

Figure 35-5. Master Mode Typical Application Block Diagram



Rp: Pull up value as given by the I C Standard

### 35.8.3 Programming Master Mode

The following registers have to be programmed before entering Master mode:

1. DADR (+ IADRSZ + IADR if a 10 bit device is addressed): The device address is used to access slave devices in read or write mode.
2. CKDIV + CHDIV + CLDIV: Clock Waveform.
3. SVDIS: Disable the slave mode.
4. MSEN: Enable the master mode.

### 35.8.4 Master Transmitter Mode

After the master initiates a Start condition when writing into the Transmit Holding Register, TWI\_THR, it sends a 7-bit slave address, configured in the Master Mode register (DADR in TWI\_MMR), to notify the slave device. The bit following the slave address indicates the transfer direction, 0 in this case (MREAD = 0 in TWI\_MMR).

The TWI transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the Not Acknowledge bit (NACK) in the status register if the slave does not acknowledge the byte. As with the other status bits, an interrupt can be generated if enabled in the interrupt enable register (TWI\_IER). If the slave acknowledges the byte, the data written in the TWI\_THR, is then shifted in the internal shifter and transferred. When an acknowledge is detected, the TXRDY bit is set until a new write in the TWI\_THR.

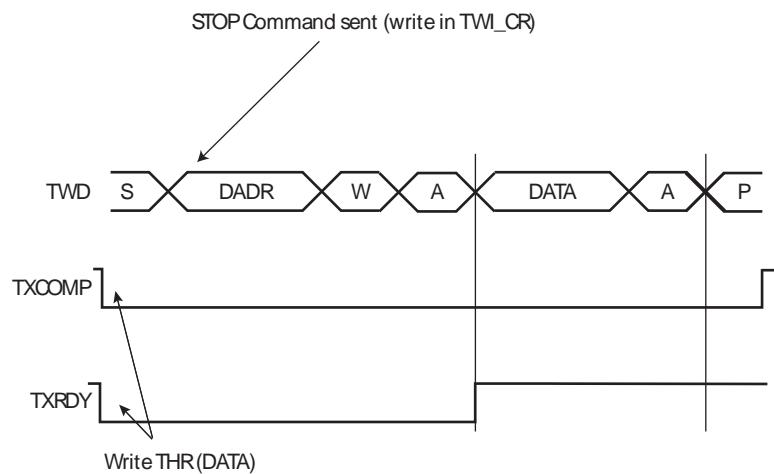
TXRDY is used as Transmit Ready for the PDC transmit channel.

While no new data is written in the TWI\_THR, the Serial Clock Line is tied low. When new data is written in the TWI\_THR, the SCL is released and the data is sent. To generate a STOP event, the STOP command must be performed by writing in the STOP field of TWI\_CR.

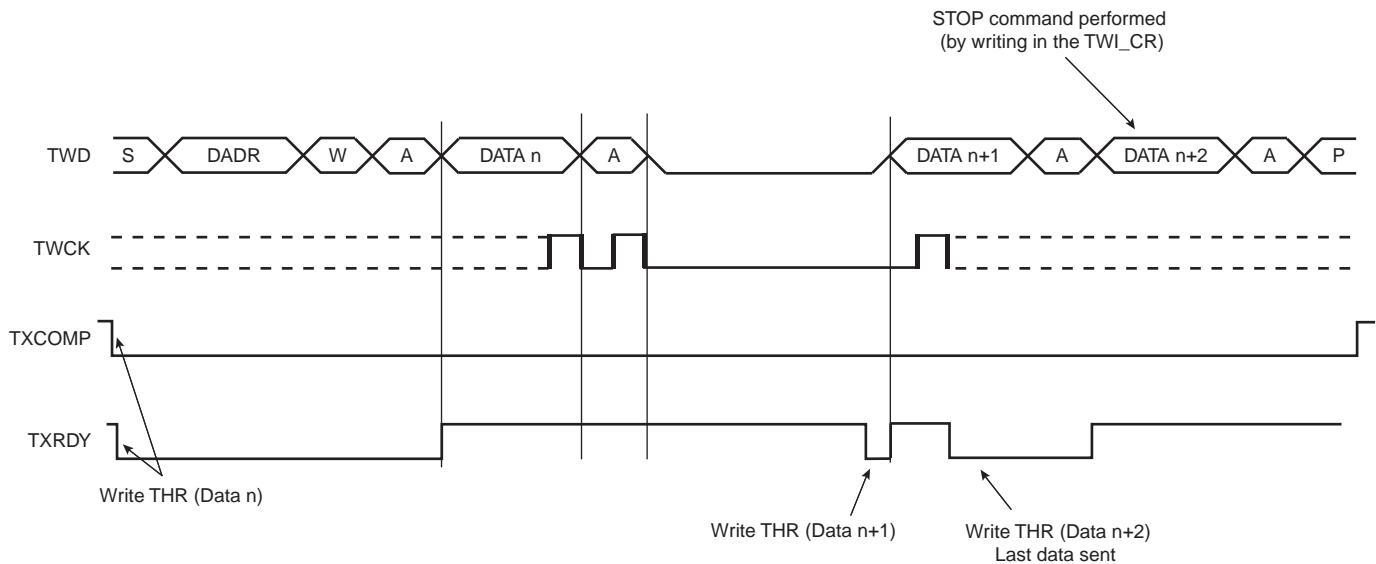
After a Master Write transfer, the Serial Clock line is stretched (tied low) while no new data is written in the TWI\_THR or until a STOP command is performed.

See [Figure 35-6](#), [Figure 35-7](#), and [Figure 35-8](#).

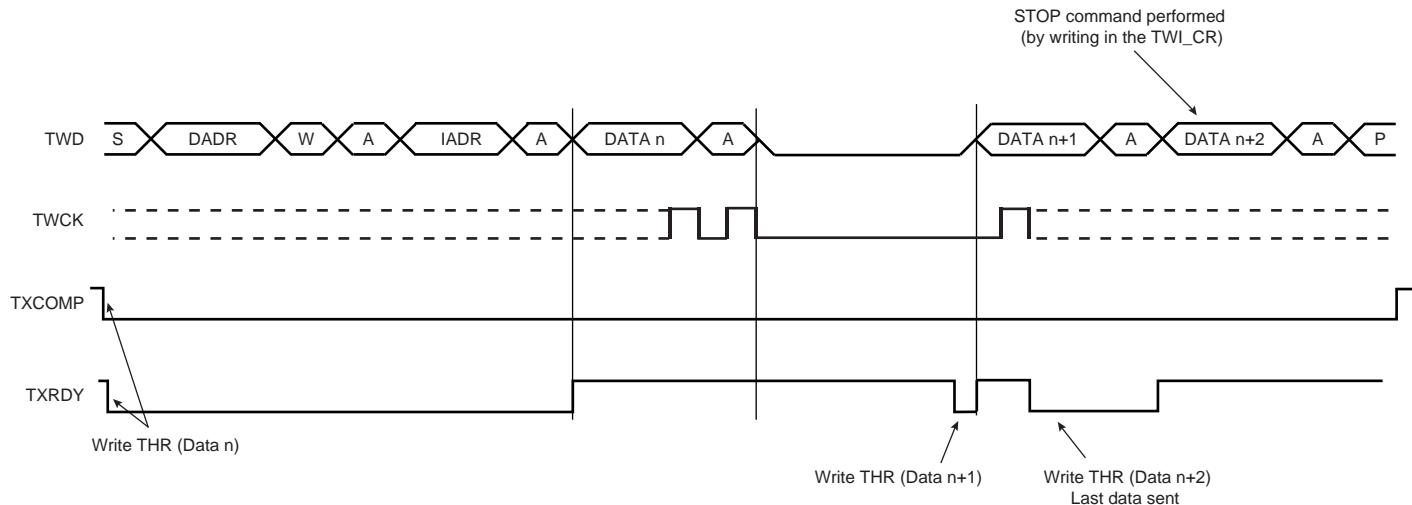
**Figure 35-6. Master Write with One Data Byte**



**Figure 35-7. Master Write with Multiple Data Bytes**



**Figure 35-8. Master Write with One Byte Internal Address and Multiple Data Bytes**



### 35.8.5 Master Receiver Mode

The read sequence begins by setting the START bit. After the start condition has been sent, the master sends a 7-bit slave address to notify the slave device. The bit following the slave address indicates the transfer direction, 1 in this case (MREAD = 1 in TWI\_MMR). During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the NACK bit in the status register if the slave does not acknowledge the byte.

If an acknowledge is received, the master is then ready to receive data from the slave. After data has been received, the master sends an acknowledge condition to notify the slave that the data has been received except for the last data, after the stop condition. See [Figure 35-9](#). When the RXRDY bit is set in the status register, a character has been received in the receive-holding register (TWI\_RHR). The RXRDY bit is reset when reading the TWI\_RHR.

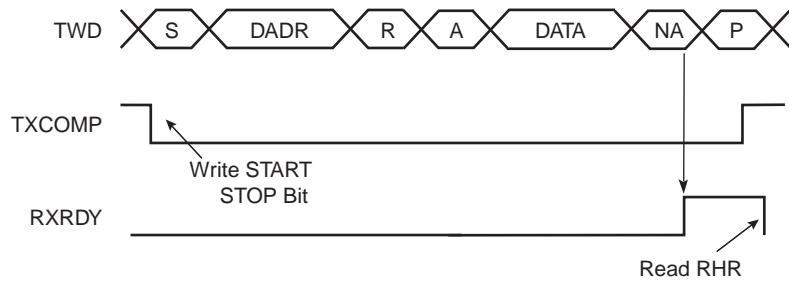
When a single data byte read is performed, with or without internal address (IADR), the START and STOP bits must be set at the same time. See [Figure 35-9](#). When a multiple data byte read is performed, with or without internal address (IADR), the STOP bit must be set after the next-to-last data received. See [Figure 35-10](#). For Internal Address usage see [Section 35.8.6](#).

If the receive holding register (TWI\_RHR) is full (RXRDY high) and the master is receiving data, the Serial Clock Line will be tied low before receiving the last bit of the data and until the TWI\_RHR register is read. Once the TWI\_RHR register is read, the master will stop stretching the Serial Clock Line and end the data reception. See [Figure 35-11](#).

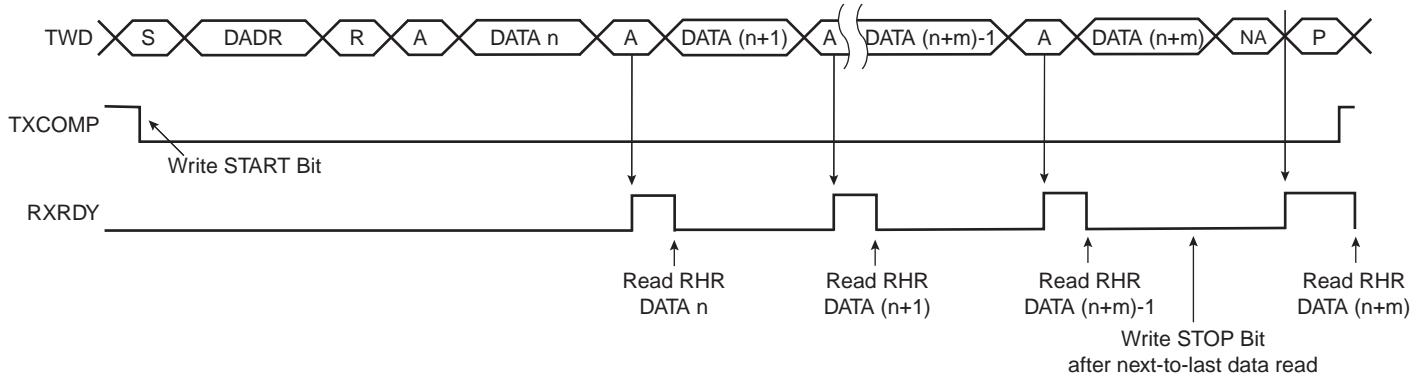
**Warning:** When receiving multiple bytes in master read mode, if the next-to-last access is not read (the RXRDY flag remains high), the last access will not be completed until TWI\_RHR is read. The last access stops on the next-to-last bit (clock stretching). When the TWI\_RHR register is read there is only half a bit period to send the stop bit command, else another read access might occur (spurious access).

A possible workaround is to raise the STOP BIT command before reading the TWI\_RHR on the next-to-last access (within IT handler).

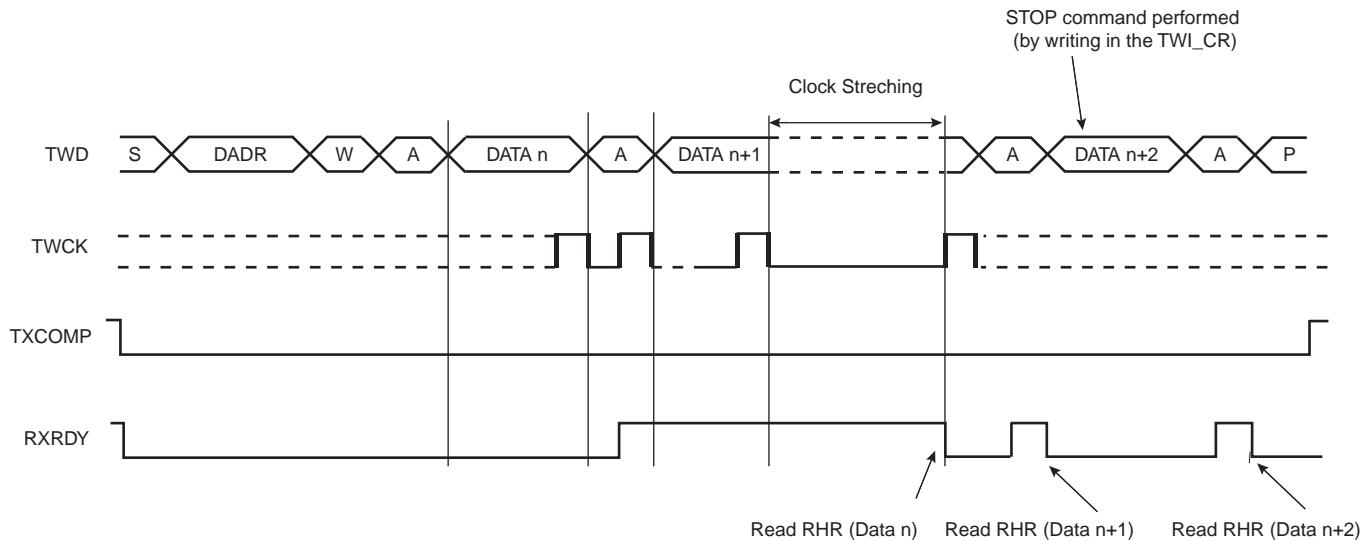
**Figure 35-9. Master Read with One Data Byte**



**Figure 35-10. Master Read with Multiple Data Bytes**



**Figure 35-11. Master Read Clock Stretching with Multiple Data Bytes**



RXRDY is used as Receive Ready for the PDC receive channel.

### 35.8.6 Internal Address

The TWI interface can perform various transfer formats: Transfers with 7-bit slave address devices and 10-bit slave address devices.

### 35.8.6.1 7-bit Slave Addressing

When Addressing 7-bit slave devices, the internal address bytes are used to perform random address (read or write) accesses to reach one or more data bytes, within a memory page location in a serial memory, for example. When performing read operations with an internal address, the TWI performs a write operation to set the internal address into the slave device, and then switch to Master Receiver mode. Note that the second start condition (after sending the IADR) is sometimes called “repeated start” (Sr) in I<sup>2</sup>C fully-compatible devices. See [Figure 35-13](#). See [Figure 35-12](#) and [Figure 35-14](#) for Master Write operation with internal address.

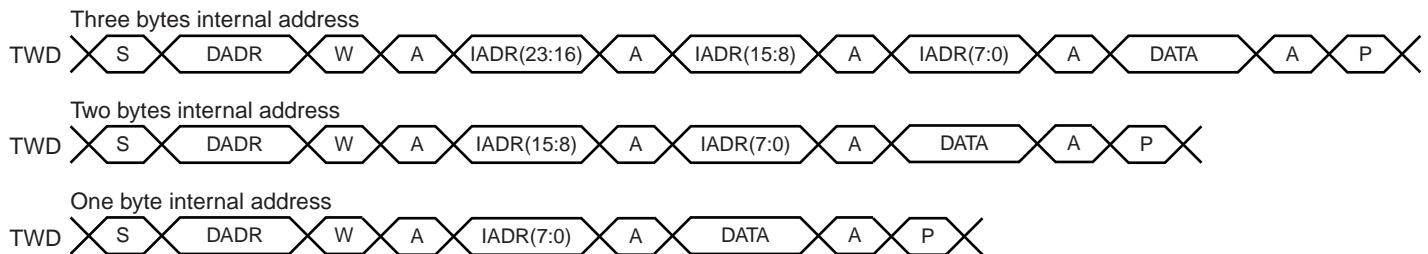
The three internal address bytes are configurable through the Master Mode register (TWI\_MMR).

If the slave device supports only a 7-bit address, i.e. no internal address, IADRSZ must be set to 0.

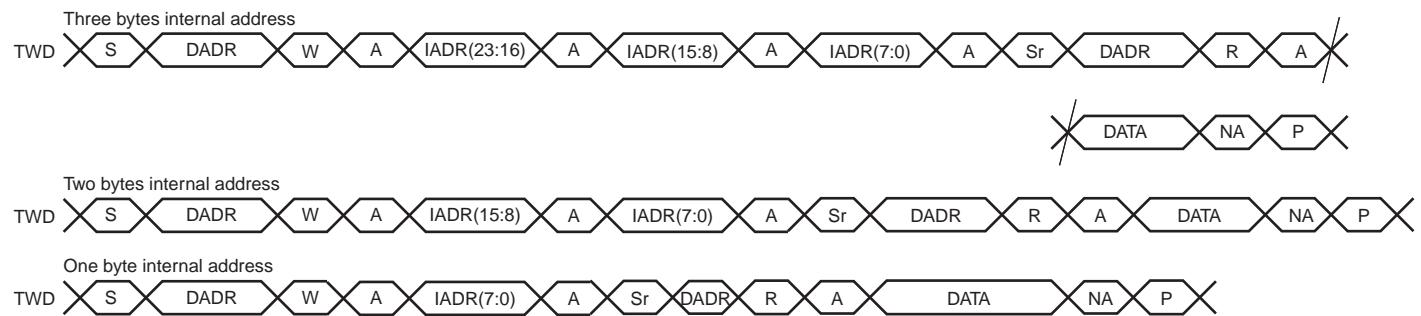
In the figures below the following abbreviations are used:

- S Start
- Sr Repeated Start
- P Stop
- W Write
- R Read
- A Acknowledge
- NA Not Acknowledge
- DADR Device Address
- IADR Internal Address

**Figure 35-12. Master Write with One, Two or Three Bytes Internal Address and One Data Byte**



**Figure 35-13. Master Read with One, Two or Three Bytes Internal Address and One Data Byte**



### 35.8.6.2 10-bit Slave Addressing

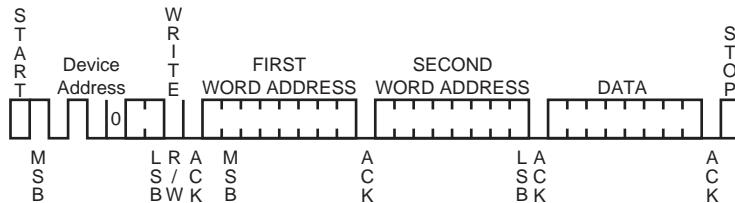
For a slave address higher than 7 bits, the user must configure the address size (IADRSZ) and set the other slave address bits in the internal address register (TWI\_IADR). The two remaining Internal address bytes, IADR[15:8] and IADR[23:16] can be used the same as in 7-bit Slave Addressing.

**Example:** Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1,
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program TWI\_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

[Figure 35-14](#) below shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

**Figure 35-14. Internal Address Usage**



### 35.8.7 Using the Peripheral DMA Controller (PDC)

The use of the PDC significantly reduces the CPU load.

To assure correct implementation, respect the following programming sequences:

#### 35.8.7.1 Data Transmit with the PDC

1. Initialize the transmit PDC (memory pointers, transfer size - 1).
2. Configure the master (DADR, CKDIV, etc.) or slave mode.
3. Start the transfer by setting the PDC TXTEN bit.
4. Wait for the PDC ENDTX Flag either by using the polling method or ENDTX interrupt.
5. Disable the PDC by setting the PDC TXTDIS bit.
6. Wait for the TXRDY flag in TWI\_SR register
7. Set the STOP command in TWI\_CR.
8. Write the last character in TWI\_THR
9. (Optional) Wait for the TXCOMP flag in TWI\_SR register before disabling the peripheral clock if required

#### 35.8.7.2 Data Receive with the PDC

The PDC transfer size must be defined with the buffer size minus 2. The two remaining characters must be managed without PDC to ensure that the exact number of bytes are received whatever the system bus latency conditions encountered during the end of buffer transfer period.

In slave mode, the number of characters to receive must be known in order to configure the PDC.

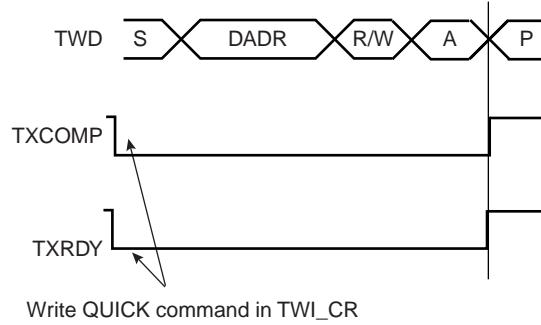
1. Initialize the receive PDC (memory pointers, transfer size - 2).
2. Configure the master (DADR, CKDIV, etc.) or slave mode.
3. Set the PDC RXTEEN bit.
4. (Master Only) Write the START bit in the TWI\_CR register to start the transfer
5. Wait for the PDC ENDRX Flag either by using polling method or ENDRX interrupt.
6. Disable the PDC by setting the PDC RXTDIS bit.
7. Wait for the RXRDY flag in TWI\_SR register
8. Set the STOP command in TWI\_CR
9. Read the penultimate character in TWI\_RHR
10. Wait for the RXRDY flag in TWI\_SR register
11. Read the last character in TWI\_RHR
12. (Optional) Wait for the TXCOMP flag in TWI\_SR register before disabling the peripheral clock if required

### 35.8.8 SMBUS Quick Command (Master Mode Only)

The TWI interface can perform a Quick Command:

1. Configure the master mode (DADR, CKDIV, etc.).
2. Write the MREAD bit in the TWI\_MMR register at the value of the one-bit command to be sent.
3. Start the transfer by setting the QUICK bit in the TWI\_CR.

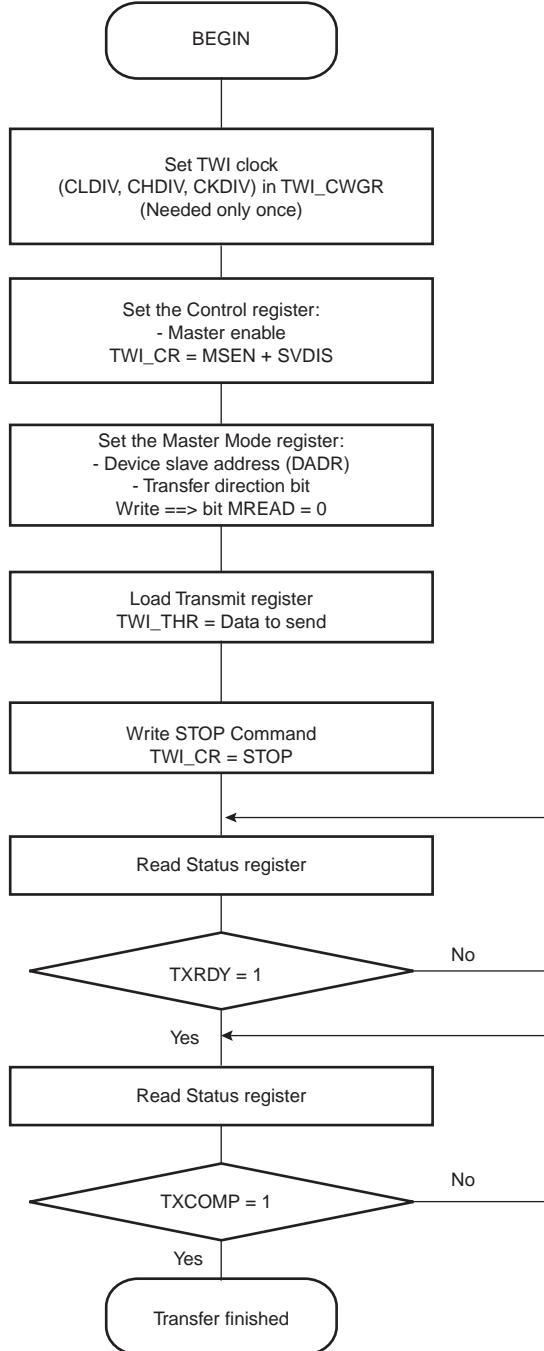
**Figure 35-15. SMBUS Quick Command**



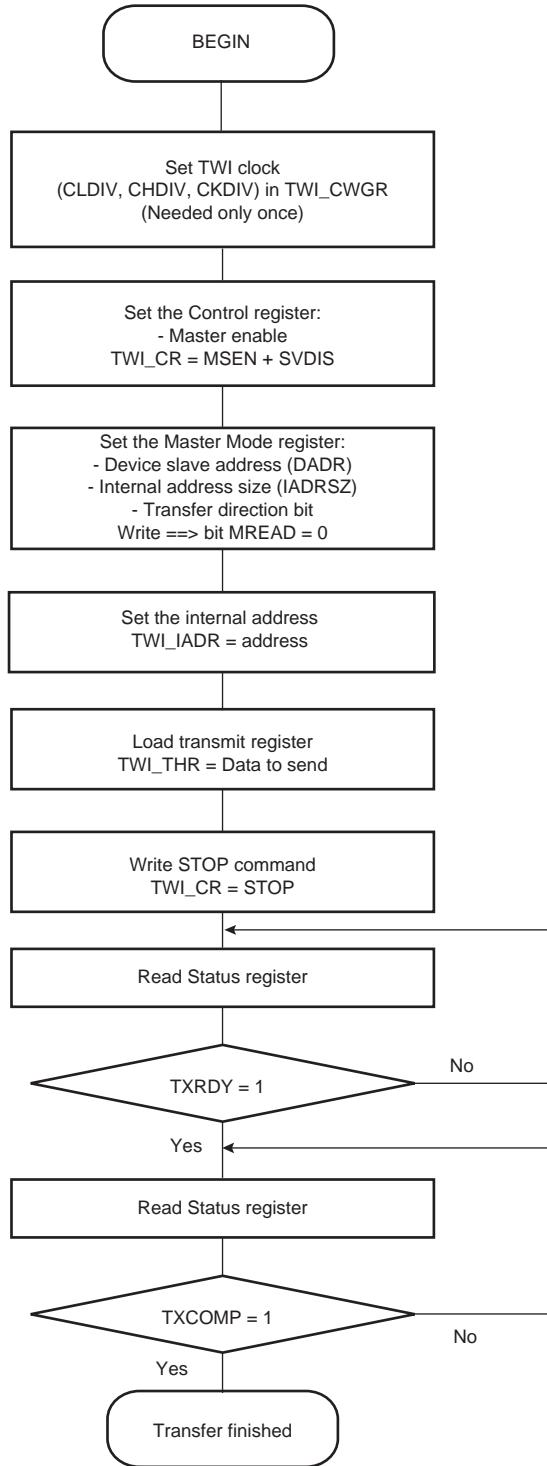
### 35.8.9 Read-write Flowcharts

The following flowcharts shown in [Figure 35-17 on page 767](#), [Figure 35-18 on page 768](#), [Figure 35-19 on page 769](#), [Figure 35-20 on page 770](#) and [Figure 35-21 on page 771](#) give examples for read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the interrupt enable register (TWI\_IER) be configured first.

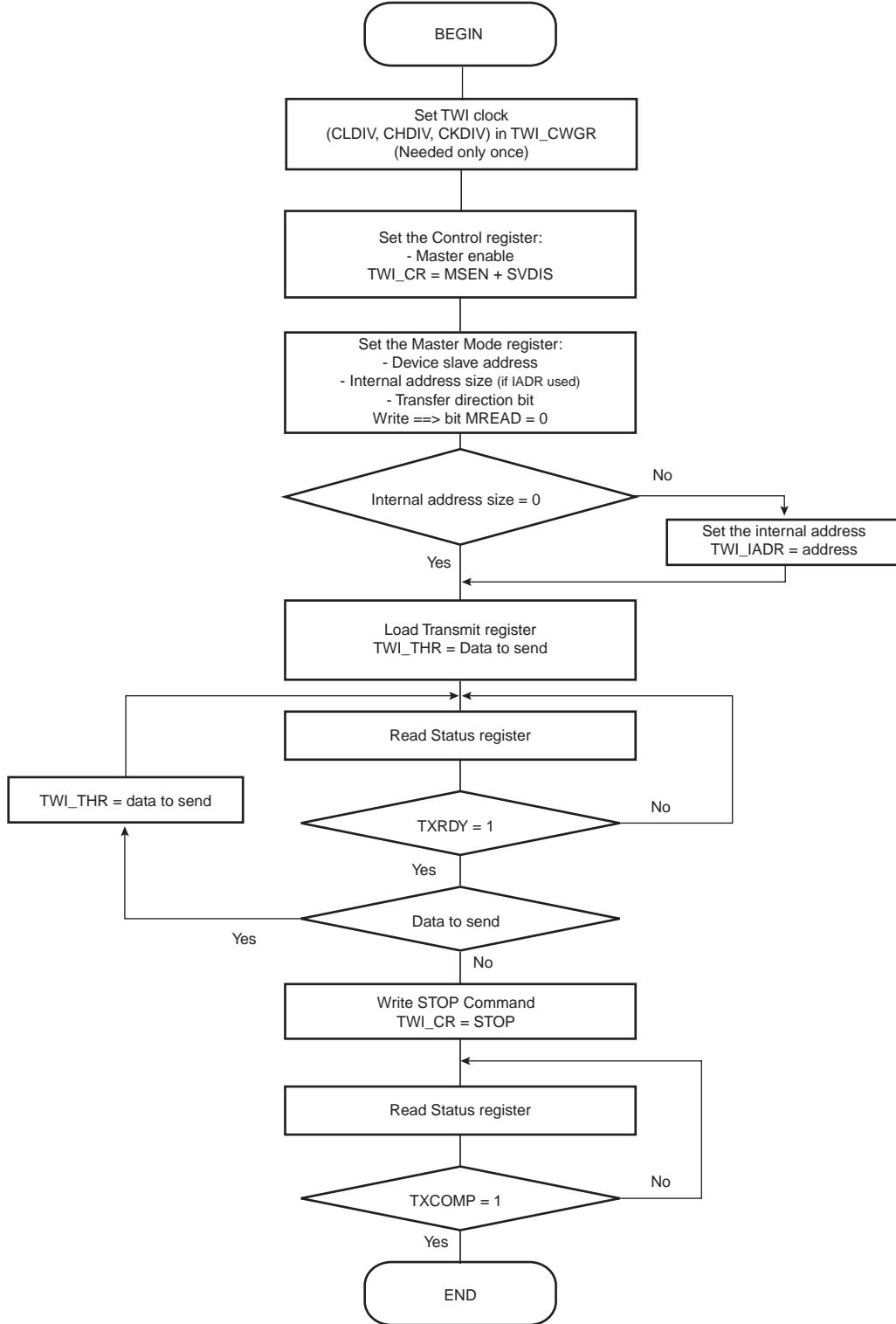
**Figure 35-16. TWI Write Operation with Single Data Byte without Internal Address**



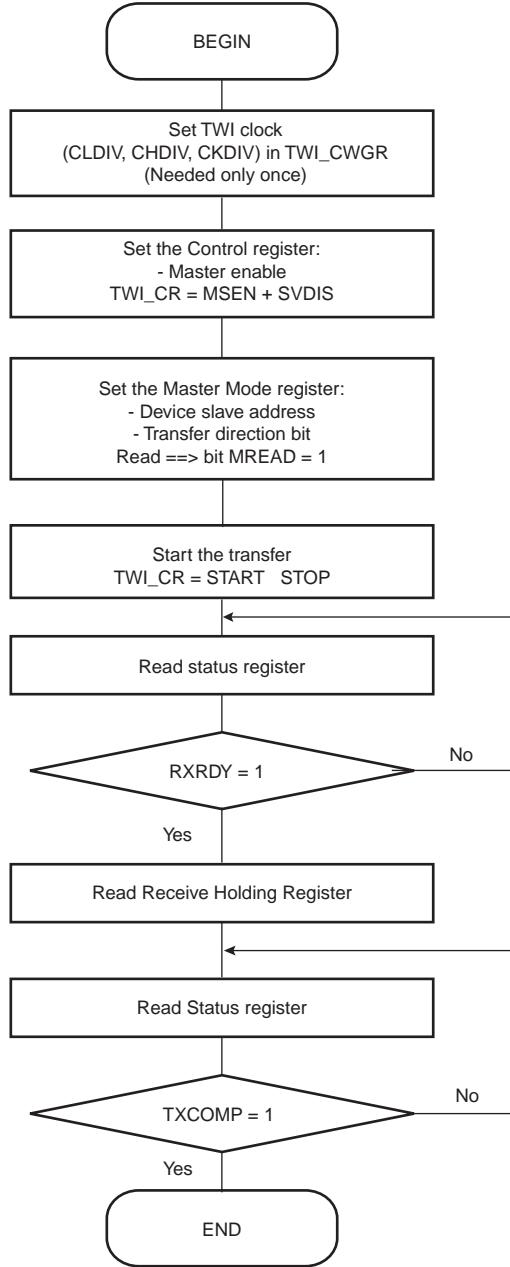
**Figure 35-17. TWI Write Operation with Single Data Byte and Internal Address**



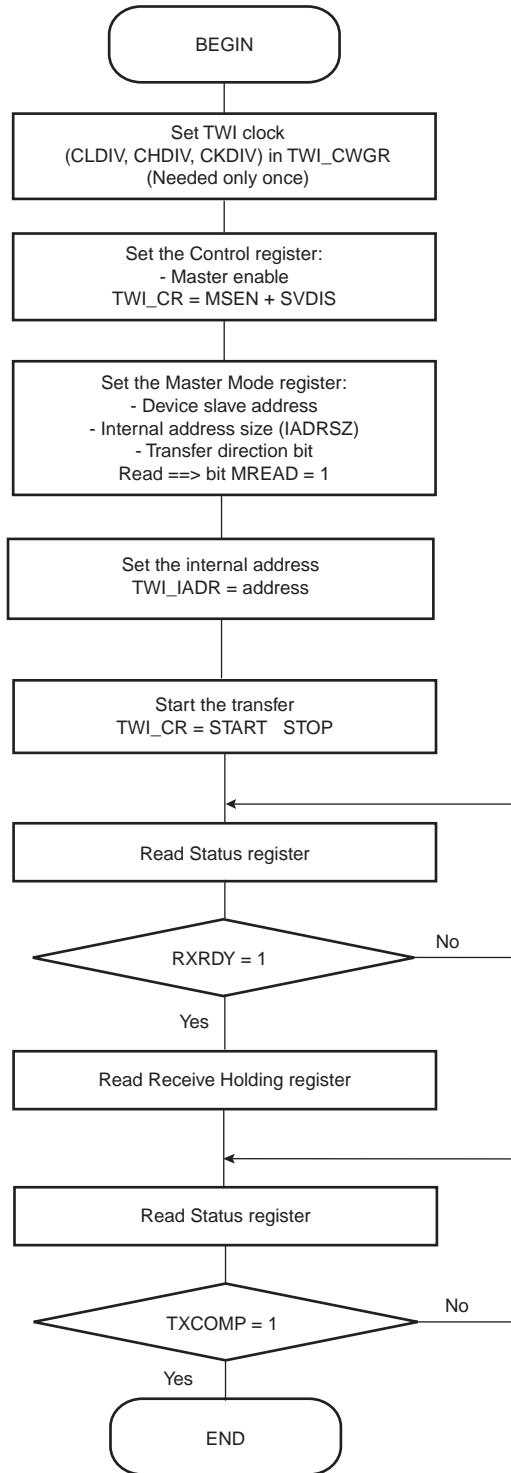
**Figure 35-18. TWI Write Operation with Multiple Data Bytes with or without Internal Address**



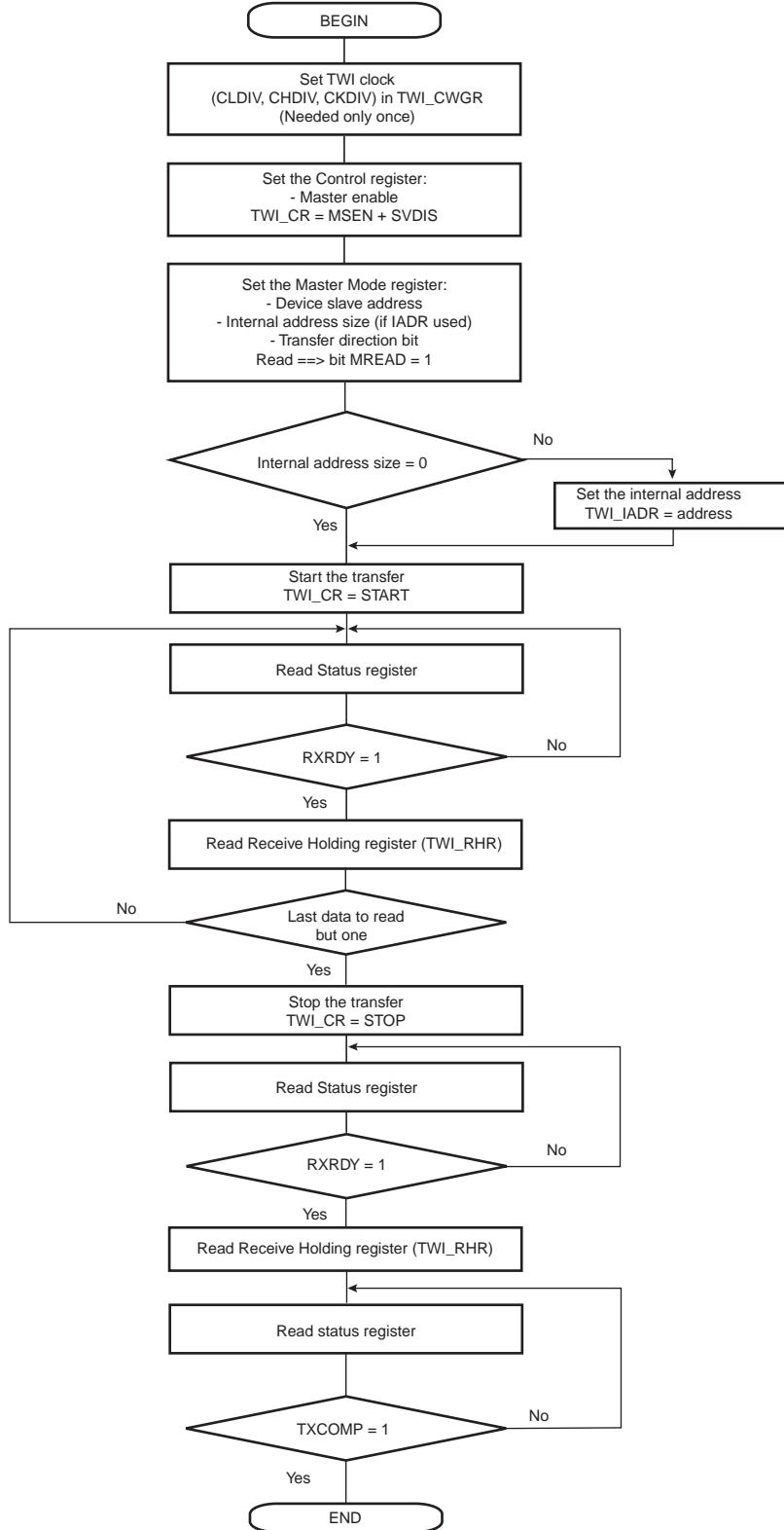
**Figure 35-19. TWI Read Operation with Single Data Byte without Internal Address**



**Figure 35-20. TWI Read Operation with Single Data Byte and Internal Address**



**Figure 35-21. TWI Read Operation with Multiple Data Bytes with or without Internal Address**



## 35.9 Multi-master Mode

### 35.9.1 Definition

More than one master may handle the bus at the same time without data corruption by using arbitration.

Arbitration starts as soon as two or more masters place information on the bus at the same time, and stops (arbitration is lost) for the master that intends to send a logical one while the other master sends a logical zero.

As soon as arbitration is lost by a master, it stops sending data and listens to the bus in order to detect a stop. When the stop is detected, the master who has lost arbitration may put its data on the bus by respecting arbitration.

Arbitration is illustrated in [Figure 35-23 on page 773](#).

### 35.9.2 Different Multi-master Modes

Two multi-master modes may be distinguished:

1. TWI is considered as a Master only and will never be addressed.
2. TWI may be either a Master or a Slave and may be addressed.

Note: In both Multi-master modes arbitration is supported.

#### 35.9.2.1 TWI as Master Only

In this mode, TWI is considered as a Master only (MSEN is always at one) and must be driven like a Master with the ARBLST (ARBitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the programmer must reinitiate the data transfer.

If the user starts a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWI automatically waits for a STOP condition on the bus to initiate the transfer (see [Figure 35-22 on page 773](#)).

Note: The state of the bus (busy or free) is not indicated in the user interface.

#### 35.9.2.2 TWI as Master or Slave

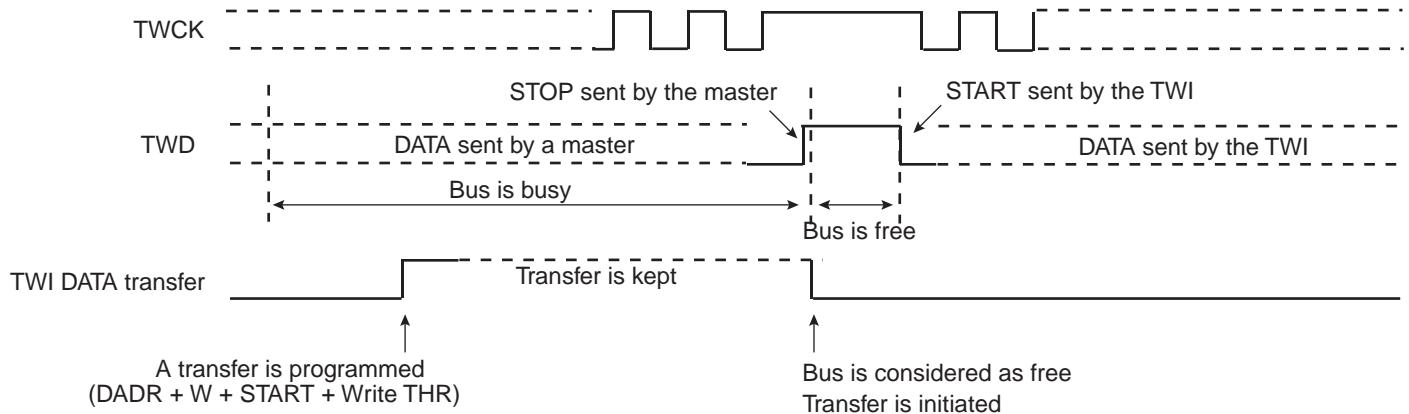
The automatic reversal from Master to Slave is not supported in case of a lost arbitration.

Then, in the case where TWI may be either a Master or a Slave, the programmer must manage the pseudo Multi-master mode described in the steps below.

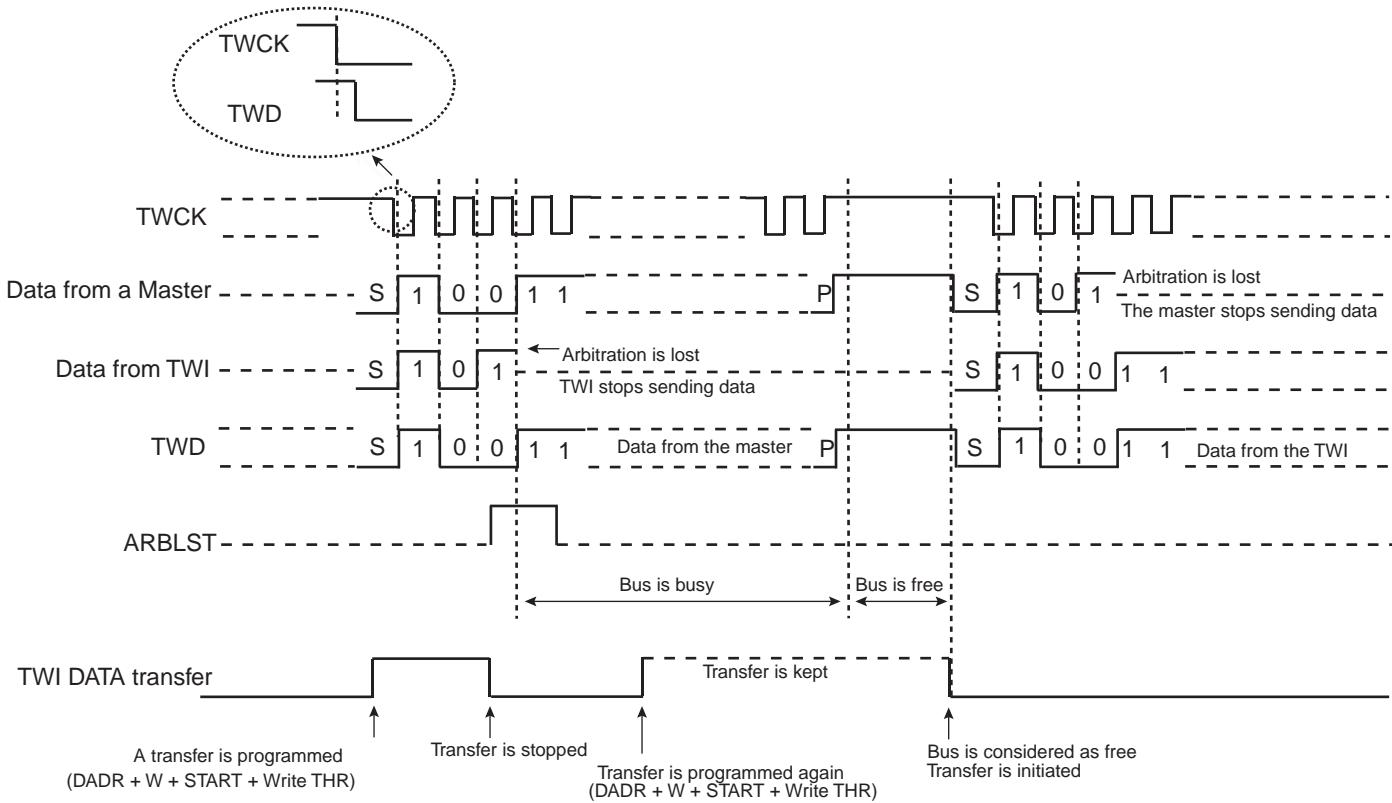
1. Program TWI in Slave mode (SADR + MSDIS + SVEN) and perform Slave Access (if TWI is addressed).
2. If TWI has to be set in Master mode, wait until TXCOMP flag is at 1.
3. Program Master mode (DADR + SVDIS + MSEN) and start the transfer (ex: START + Write in THR).
4. As soon as the Master mode is enabled, TWI scans the bus in order to detect if it is busy or free. When the bus is considered as free, TWI initiates the transfer.
5. As soon as the transfer is initiated and until a STOP condition is sent, the arbitration becomes relevant and the user must monitor the ARBLST flag.
6. If the arbitration is lost (ARBLST is set to 1), the user must program the TWI in Slave mode in the case where the Master that won the arbitration wanted to access the TWI.
7. If TWI has to be set in Slave mode, wait until TXCOMP flag is at 1 and then program the Slave mode.

Note: In the case where the arbitration is lost and TWI is addressed, TWI will not acknowledge even if it is programmed in Slave mode as soon as ARBLST is set to 1. Then, the Master must repeat SADR.

**Figure 35-22. Programmer Sends Data While the Bus is Busy**

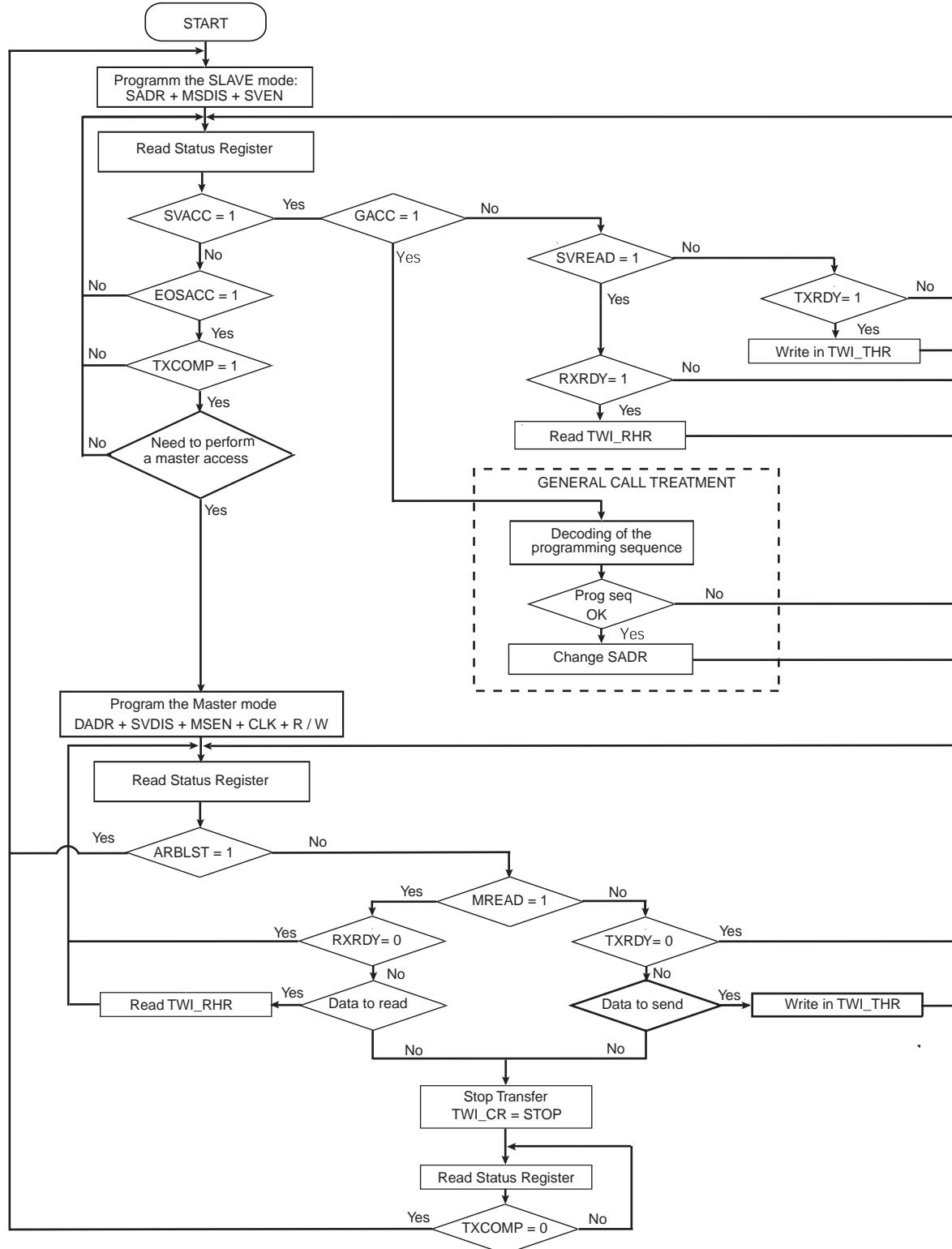


**Figure 35-23. Arbitration Cases**



The flowchart shown in [Figure 35-24 on page 774](#) gives an example of read and write operations in Multi-master mode.

**Figure 35-24. Multi-master Flowchart**



## 35.10 Slave Mode

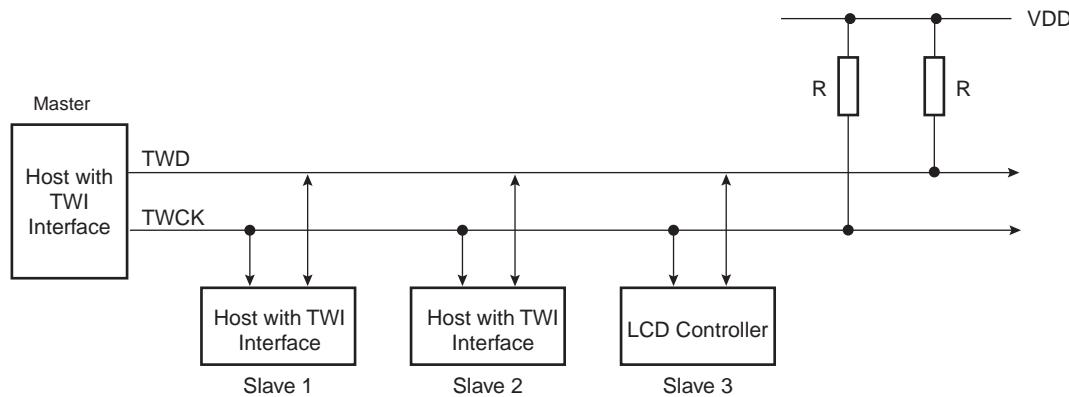
### 35.10.1 Definition

The Slave Mode is defined as a mode where the device receives the clock and the address from another device called the master.

In this mode, the device never initiates and never completes the transmission (START, REPEATED\_START and STOP conditions are always provided by the master).

### 35.10.2 Application Block Diagram

Figure 35-25. Slave Mode Typical Application Block Diagram



### 35.10.3 Programming Slave Mode

The following fields must be programmed before entering Slave mode:

1. SADR (TWI\_SMR): The slave device address is used in order to be accessed by master devices in read or write mode.
2. MSDIS (TWI\_CR): Disable the master mode.
3. SVEN (TWI\_CR): Enable the slave mode.

As the device receives the clock, values written in TWI\_CWGR are not taken into account.

### 35.10.4 Receiving Data

After a Start or Repeated Start condition is detected and if the address sent by the Master matches with the Slave address programmed in the SADR (Slave ADdress) field, SVACC (Slave ACCess) flag is set and SVREAD (Slave READ) indicates the direction of the transfer.

SVACC remains high until a STOP condition or a repeated START is detected. When such a condition is detected, EOSACC (End Of Slave ACCess) flag is set.

#### 35.10.4.1 Read Sequence

In the case of a Read sequence (SVREAD is high), TWI transfers data written in the TWI\_THR (TWI Transmit Holding Register) until a STOP condition or a REPEATED\_START + an address different from SADR is detected. Note that at the end of the read sequence TXCOMP (Transmission Complete) flag is set and SVACC reset.

As soon as data is written in the TWI\_THR, TXRDY (Transmit Holding Register Ready) flag is reset, and it is set when the shift register is empty and the sent data acknowledged or not. If the data is not acknowledged, the NACK flag is set.

Note that a STOP or a repeated START always follows a NACK.

See [Figure 35-26 on page 776](#).

### 35.10.4.2 Write Sequence

In the case of a Write sequence (SVREAD is low), the RXRDY (Receive Holding Register Ready) flag is set as soon as a character has been received in the TWI\_RHR (TWI Receive Holding Register). RXRDY is reset when reading the TWI\_RHR.

TWI continues receiving data until a STOP condition or a REPEATED\_START + an address different from SADR is detected. Note that at the end of the write sequence TXCOMP flag is set and SVACC reset.

See [Figure 35-27 on page 777](#).

### 35.10.4.3 Clock Synchronization Sequence

In the case where TWI\_THR or TWI\_RHR is not written/read in time, TWI performs a clock synchronization.

Clock stretching information is given by the SCLWS (Clock Wait state) bit.

See [Figure 35-29 on page 778](#) and [Figure 35-30 on page 779](#).

### 35.10.4.4 General Call

In the case where a GENERAL CALL is performed, GACC (General Call ACCess) flag is set.

After GACC is set, it is up to the programmer to interpret the meaning of the GENERAL CALL and to decode the new address programming sequence.

See [Figure 35-28 on page 777](#).

## 35.10.5 Data Transfer

### 35.10.5.1 Read Operation

The read mode is defined as a data requirement from the master.

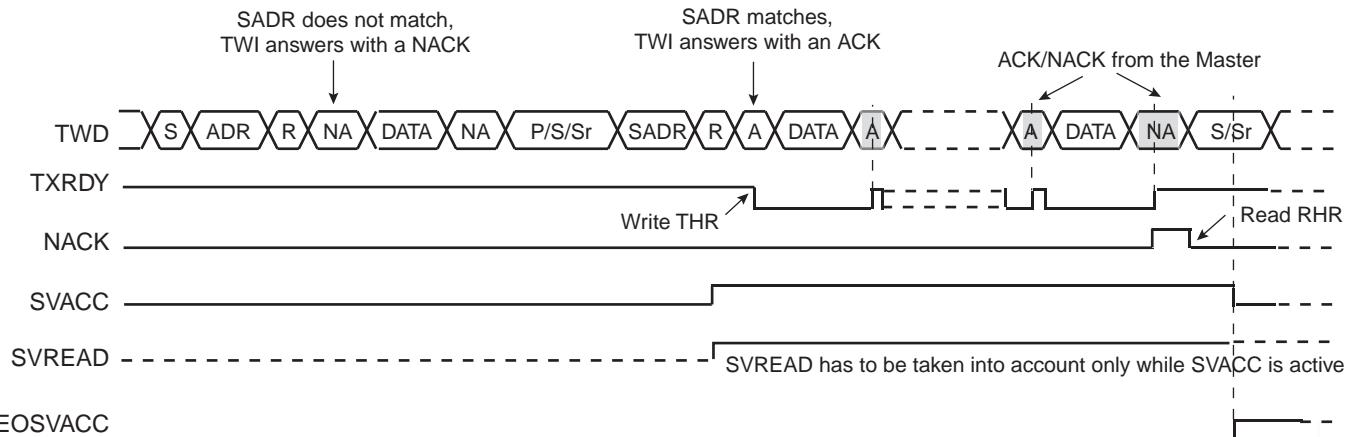
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

Until a STOP or REPEATED START condition is detected, TWI continues sending data loaded in the TWI\_THR register.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

[Figure 35-26 on page 776](#) describes the write operation.

**Figure 35-26. Read Access Ordered by a MASTER**



Notes: 1. When SVACC is low, the state of SVREAD becomes irrelevant.

2. TXRDY is reset when data has been transmitted from TWI\_THR to the shift register and set when this data has been acknowledged or non acknowledged.

### 35.10.5.2 Write Operation

The write mode is defined as a data transmission from the master.

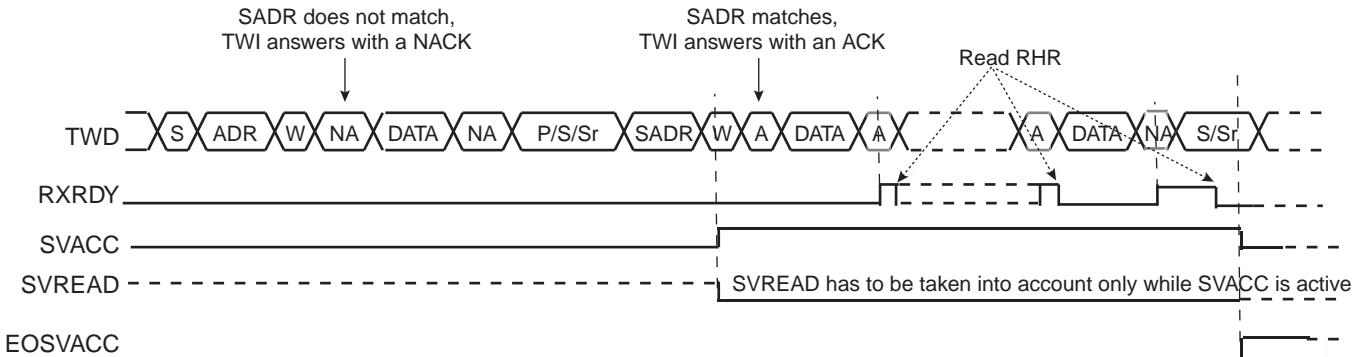
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

Until a STOP or REPEATED START condition is detected, TWI stores the received data in the TWI\_RHR register.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

[Figure 35-27 on page 777](#) describes the Write operation.

**Figure 35-27. Write Access Ordered by a Master**



Notes: 1. When SVACC is low, the state of SVREAD becomes irrelevant.

2. RXRDY is set when data has been transmitted from the shift register to the TWI\_RHR and reset when this data is read.

### 35.10.5.3 General Call

The general call is performed in order to change the address of the slave.

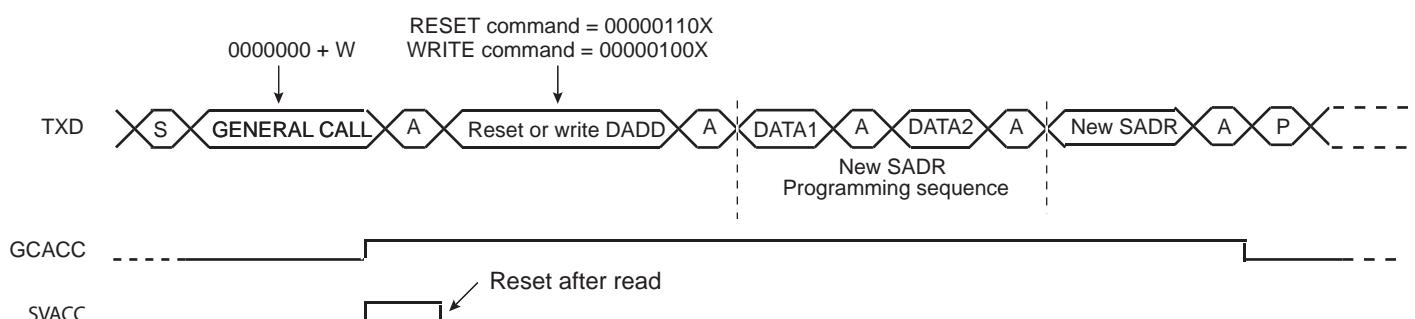
If a GENERAL CALL is detected, GACC is set.

After the detection of General Call, it is up to the programmer to decode the commands which come afterwards.

In case of a WRITE command, the programmer has to decode the programming sequence and program a new SADR if the programming sequence matches.

[Figure 35-28 on page 777](#) describes the General Call access.

**Figure 35-28. Master Performs a General Call**



Note: This method allows the user to create an own programming sequence by choosing the programming bytes and the number of them. The programming sequence has to be provided to the master.

### 35.10.5.4 Clock Synchronization

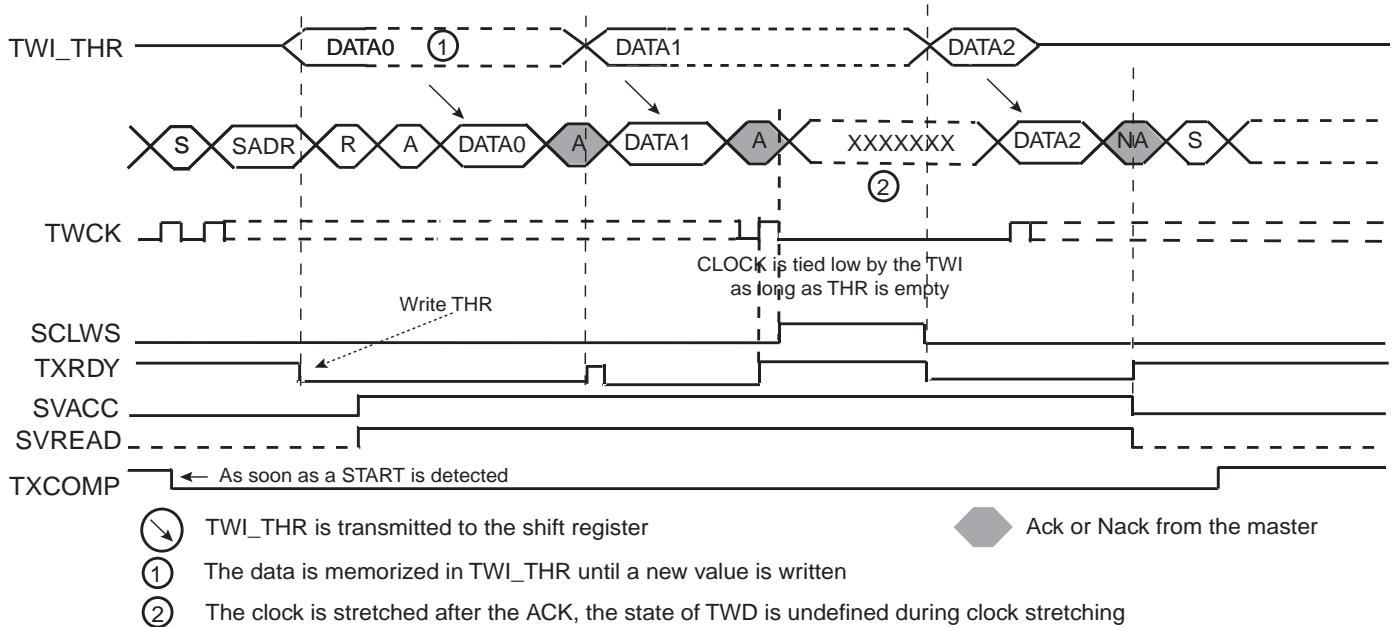
In both read and write modes, it may happen that TWI\_THR/TWI\_RHR buffer is not filled /emptied before the emission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching mechanism is implemented.

#### Clock Synchronization in Read Mode

The clock is tied low if the shift register is empty and if a STOP or REPEATED START condition was not detected. It is tied low until the shift register is loaded.

Figure 35-29 on page 778 describes the clock synchronization in Read mode.

**Figure 35-29. Clock Synchronization in Read Mode**



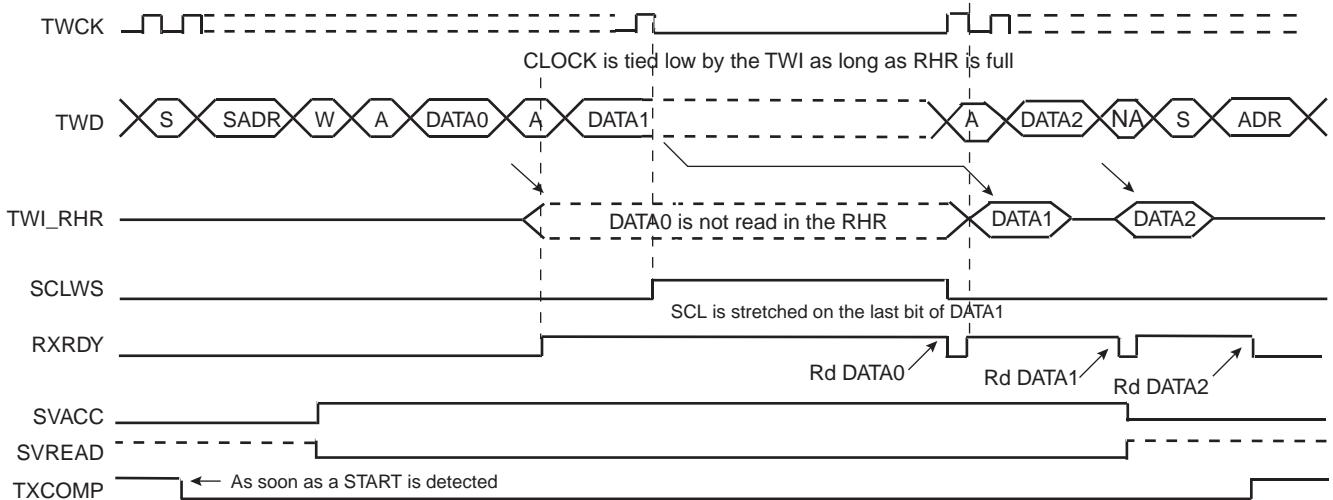
- Notes:
1. TXRDY is reset when data has been written in the TWI\_THR to the shift register and set when this data has been acknowledged or non acknowledged.
  2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED\_START + an address different from SADR.
  3. SCLWS is automatically set when the clock synchronization mechanism is started.

### Clock Synchronization in Write Mode

The clock is tied low if the shift register and the TWI\_RHR is full. If a STOP or REPEATED\_START condition was not detected, it is tied low until TWI\_RHR is read.

Figure 35-30 on page 779 describes the clock synchronization in Read mode.

**Figure 35-30. Clock Synchronization in Write Mode**



- Notes:
1. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED\_START + an address different from SADR.
  2. SCLWS is automatically set when the clock synchronization mechanism is started and automatically reset when the mechanism is finished.

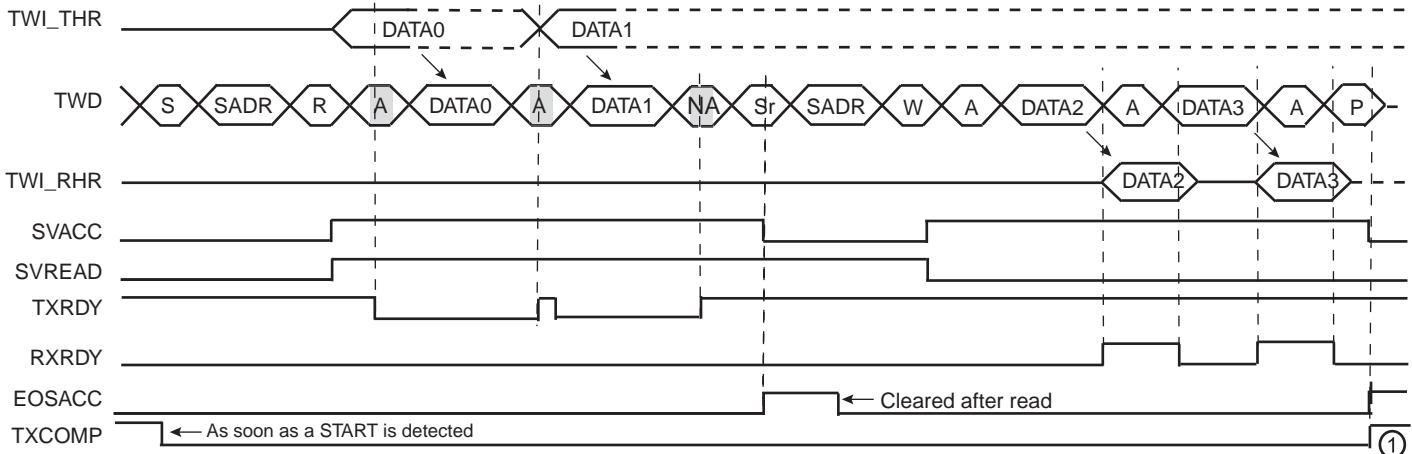
### 35.10.5.5 Reversal after a Repeated Start

#### Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

[Figure 35-31 on page 780](#) describes the repeated start + reversal from Read to Write mode.

**Figure 35-31. Repeated Start + Reversal from Read to Write Mode**

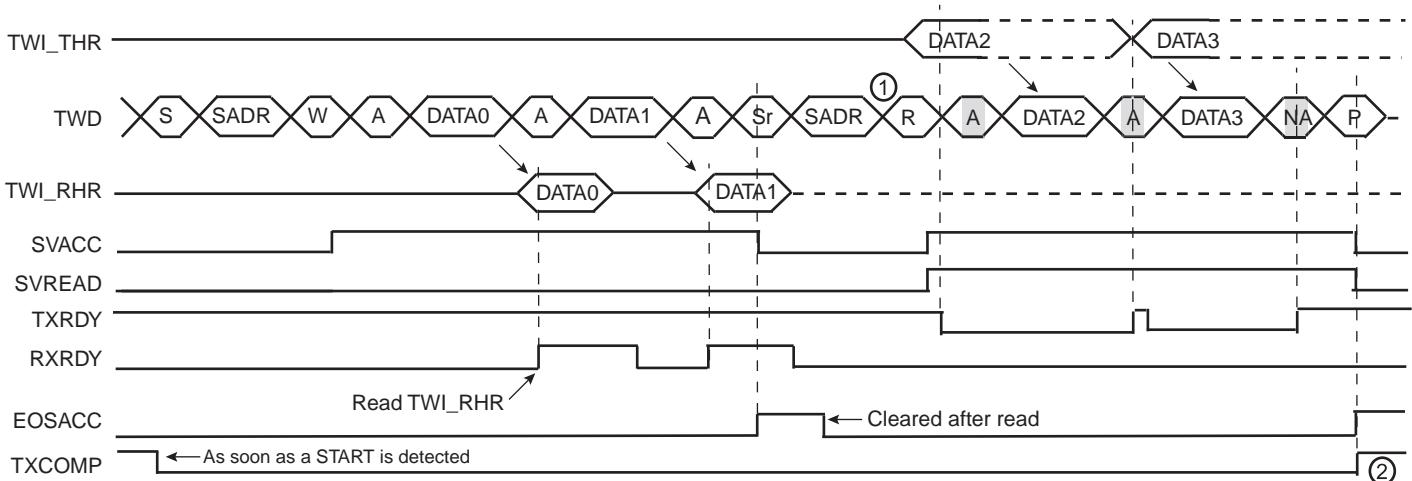


Note: 1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

#### Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command. [Figure 35-32 on page 780](#) describes the repeated start + reversal from Write to Read mode.

**Figure 35-32. Repeated Start + Reversal from Write to Read Mode**



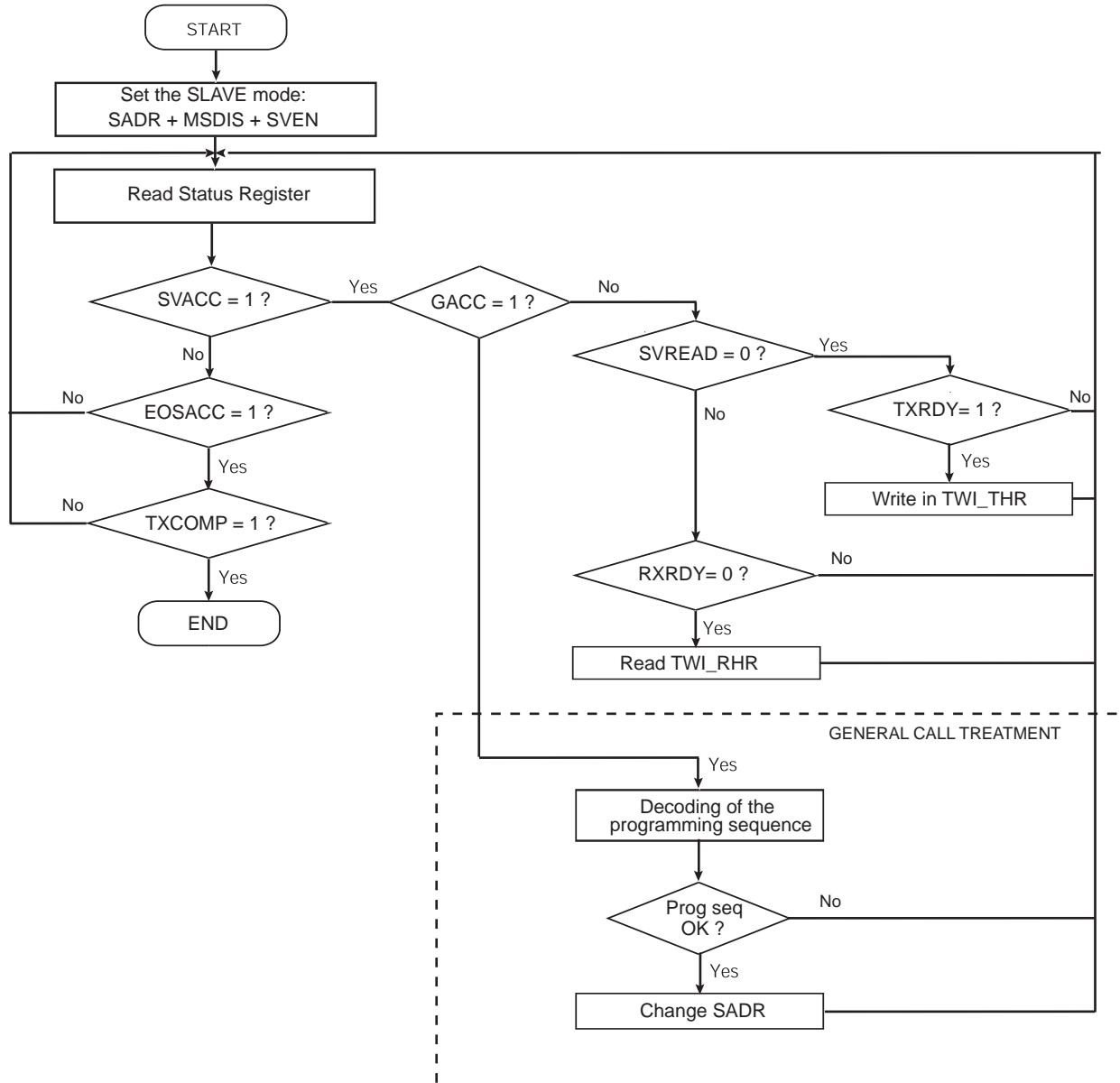
Notes: 1. In this case, if TWI\_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.

2. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

### 35.10.6 Read Write Flowcharts

The flowchart shown in [Figure 35-33 on page 781](#) gives an example of read and write operations in Slave mode. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the interrupt enable register (TWI\_IER) be configured first.

**Figure 35-33. Read Write Flowchart in Slave Mode**



## 35.11 Write Protection System

In order to bring security to the TWI, a write protection system has been implemented.

The write protection mode prevents the write of "[TWI Clock Waveform Generator Register](#)" and "[TWI Slave Mode Register](#)". When this mode is enabled and one of the protected registers is written, an error is generated in the "[TWI Write Protection Status Register](#)" and the register write request is canceled. When a write protection error occurs the WPVS flag is set and the address of the corresponding canceled register write is available in the WPVSR field of the "[TWI Write Protection Status Register](#)" register.

Due to the nature of the write protection feature, enabling and disabling the write protection mode requires the use of a security code. Thus when enabling or disabling the write protection mode the WPKEY field of the "[TWI Write Protection Mode Register](#)" must be filled with the "TWI" ASCII code (0x545749) otherwise the register write will be canceled.

## 35.12 Two-wire Interface (TWI) User Interface

Table 35-6. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	TWI_CR	Write-only	N / A
0x04	Master Mode Register	TWI_MMR	Read-write	0x00000000
0x08	Slave Mode Register	TWI_SMR	Read-write	0x00000000
0x0C	Internal Address Register	TWI_IADR	Read-write	0x00000000
0x10	Clock Waveform Generator Register	TWI_CWGR	Read-write	0x00000000
0x14 - 0x1C	Reserved	—	—	—
0x20	Status Register	TWI_SR	Read-only	0x0000F009
0x24	Interrupt Enable Register	TWI_IER	Write-only	N / A
0x28	Interrupt Disable Register	TWI_IDR	Write-only	N / A
0x2C	Interrupt Mask Register	TWI_IMR	Read-only	0x00000000
0x30	Receive Holding Register	TWI_RHR	Read-only	0x00000000
0x34	Transmit Holding Register	TWI_THR	Write-only	0x00000000
0x38-0xE0	Reserved	—	—	—
0xE4	Protection Mode Register	TWI_WPMR	Read-write	0x00000000
0xE8	Protection Status Register	TWI_WPSR	Read-only	0x00000000
0xEC - 0xFC <sup>(1)</sup>	Reserved	—	—	—
0x100 - 0x128	Reserved for PDC registers	—	—	—

Note: 1. All unlisted offset values are considered as “reserved”.

### 35.12.1 TWI Control Register

**Name:** TWI\_CR

**Address:** 0x400A8000 (0), 0x400AC000 (1)

**Access:** Write-only

**Reset:** 0x00000000

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
SWRST	QUICK	SVDIS	SVEN	MSDIS	MSEN	STOP	START

- **START: Send a START Condition**

0 = No effect.

1 = A frame beginning with a START bit is transmitted according to the features defined in the mode register.

This action is necessary when the TWI peripheral wants to read data from a slave. When configured in Master Mode with a write operation, a frame is sent as soon as the user writes a character in the Transmit Holding Register (TWI\_THR).

- **STOP: Send a STOP Condition**

0 = No effect.

1 = STOP Condition is sent just after completing the current byte transmission in master read mode.

- In single data byte master read, the START and STOP must both be set.
- In multiple data bytes master read, the STOP must be set after the last data received but one.
- In master read mode, if a NACK bit is received, the STOP is automatically performed.
- In master data write operation, a STOP condition will be sent after the transmission of the current data is finished.

- **MSEN: TWI Master Mode Enabled**

0 = No effect.

1 = If MSDIS = 0, the master mode is enabled.

Note: Switching from Slave to Master mode is only permitted when TXCOMP = 1.

- **MSDIS: TWI Master Mode Disabled**

0 = No effect.

1 = The master mode is disabled, all pending data is transmitted. The shifter and holding characters (if it contains data) are transmitted in case of write operation. In read operation, the character being transferred must be completely received before disabling.

- **SVEN: TWI Slave Mode Enabled**

0 = No effect.

1 = If SVDIS = 0, the slave mode is enabled.

Note: Switching from Master to Slave mode is only permitted when TXCOMP = 1.

- **SVDIS: TWI Slave Mode Disabled**

0 = No effect.

1 = The slave mode is disabled. The shifter and holding characters (if it contains data) are transmitted in case of read operation. In write operation, the character being transferred must be completely received before disabling.

- **QUICK: SMBUS Quick Command**

0 = No effect.

1 = If Master mode is enabled, a SMBUS Quick Command is sent.

- **SWRST: Software Reset**

0 = No effect.

1 = Equivalent to a system reset.

### 35.12.2 TWI Master Mode Register

**Name:** TWI\_MMR

**Address:** 0x400A8004 (0), 0x400AC004 (1)

**Access:** Read-write

**Reset:** 0x00000000

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

- **IADRSZ: Internal Device Address Size**

Value	Name	Description
0	NONE	No internal device address
1	1_BYT	One-byte internal device address
2	2_BYT	Two-byte internal device address
3	3_BYT	Three-byte internal device address

- **MREAD: Master Read Direction**

0 = Master write direction.

1 = Master read direction.

- **DADR: Device Address**

The device address is used to access slave devices in read or write mode. Those bits are only used in Master mode.

### 35.12.3 TWI Slave Mode Register

Name: TWI\_SMR

Address: 0x400A8008 (0), 0x400AC008 (1)

Access: Read-write

Reset: 0x00000000

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

This register can only be written if the WPEN bit is cleared in the "TWI Write Protection Mode Register".

- **SADR: Slave Address**

The slave device address is used in Slave mode in order to be accessed by master devices in read or write mode.

SADR must be programmed before enabling the Slave mode or after a general call. Writes at other times have no effect.

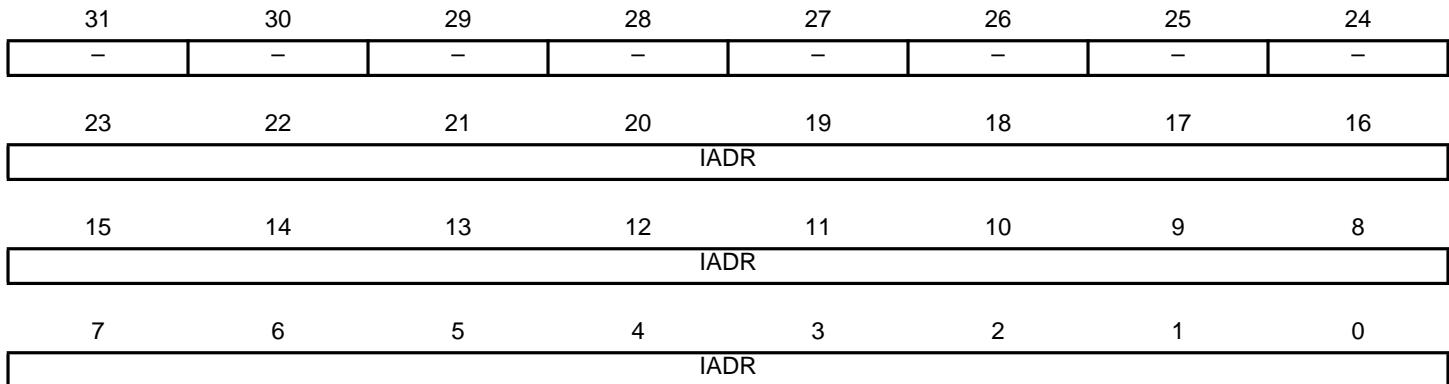
#### **35.12.4 TWI Internal Address Register**

Name: TWI\_IADR

**Address:** 0x400A800C (0), 0x400AC00C (1)

Access: Read-write

**Reset:** 0x00000000



- IADR: Internal Address

0, 1, 2 or 3 bytes depending on IADRSZ.

### 35.12.5 TWI Clock Waveform Generator Register

**Name:** TWI\_CWGR

**Address:** 0x400A8010 (0), 0x400AC010 (1)

**Access:** Read-write

**Reset:** 0x00000000

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

This register can only be written if the WPEN bit is cleared in the "TWI Write Protection Mode Register".

TWI\_CWGR is only used in Master mode.

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$T_{low} = ((CLDIV \times 2^{CKDIV}) + 4) \times T_{MCK}$$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$T_{high} = ((CHDIV \times 2^{CKDIV}) + 4) \times T_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

### 35.12.6 TWI Status Register

**Name:** TWI\_SR

**Address:** 0x400A8020 (0), 0x400AC020 (1)

**Access:** Read-only

**Reset:** 0x0000F009

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCLWS	ARBLST	NACK
7	6	5	4	3	2	1	0
-	OVRE	GACC	SVACC	SVREAD	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed (automatically set / reset)**

TXCOMP used in Master mode:

0 = During the length of the current frame.

1 = When both holding and shifter registers are empty and STOP condition has been sent.

*TXCOMP behavior in Master mode* can be seen in [Figure 35-8 on page 761](#) and in [Figure 35-10 on page 762](#).

TXCOMP used in Slave mode:

0 = As soon as a Start is detected.

1 = After a Stop or a Repeated Start + an address different from SADR is detected.

*TXCOMP behavior in Slave mode* can be seen in [Figure 35-29 on page 778](#), [Figure 35-30 on page 779](#), [Figure 35-31 on page 780](#) and [Figure 35-32 on page 780](#).

- **RXRDY: Receive Holding Register Ready (automatically set / reset)**

0 = No character has been received since the last TWI\_RHR read operation.

1 = A byte has been received in the TWI\_RHR since the last read.

*RXRDY behavior in Master mode* can be seen in [Figure 35-10 on page 762](#).

*RXRDY behavior in Slave mode* can be seen in [Figure 35-27 on page 777](#), [Figure 35-30 on page 779](#), [Figure 35-31 on page 780](#) and [Figure 35-32 on page 780](#).

- **TXRDY: Transmit Holding Register Ready (automatically set / reset)**

TXRDY used in Master mode:

0 = The transmit holding register has not been transferred into shift register. Set to 0 when writing into TWI\_THR register.

1 = As soon as a data byte is transferred from TWI\_THR to internal shifter or if a NACK error is detected, TXRDY is set at the same time as TXCOMP and NACK. TXRDY is also set when MSEN is set (enable TWI).

*TXRDY behavior in Master mode* can be seen in [Figure 35.8.4 on page 759](#).

TXRDY used in Slave mode:

0 = As soon as data is written in the TWI\_THR, until this data has been transmitted and acknowledged (ACK or NACK).

1 = It indicates that the TWI\_THR is empty and that data has been transmitted and acknowledged.

If TXRDY is high and if a NACK has been detected, the transmission will be stopped. Thus when TRDY = NACK = 1, the programmer must not fill TWI\_THR to avoid losing it.

*TXRDY behavior in Slave mode* can be seen in [Figure 35-26 on page 776](#), [Figure 35-29 on page 778](#), [Figure 35-31 on page 780](#) and [Figure 35-32 on page 780](#).

- **SVREAD: Slave Read (automatically set / reset)**

This bit is only used in Slave mode. When SVACC is low (no Slave access has been detected) SVREAD is irrelevant.

0 = Indicates that a write access is performed by a Master.

1 = Indicates that a read access is performed by a Master.

*SVREAD behavior* can be seen in [Figure 35-26 on page 776](#), [Figure 35-27 on page 777](#), [Figure 35-31 on page 780](#) and [Figure 35-32 on page 780](#).

- **SVACC: Slave Access (automatically set / reset)**

This bit is only used in Slave mode.

0 = TWI is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1 = Indicates that the address decoding sequence has matched (A Master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

*SVACC behavior* can be seen in [Figure 35-26 on page 776](#), [Figure 35-27 on page 777](#), [Figure 35-31 on page 780](#) and [Figure 35-32 on page 780](#).

- **GACC: General Call Access (clear on read)**

This bit is only used in Slave mode.

0 = No General Call has been detected.

1 = A General Call has been detected. After the detection of General Call, if need be, the programmer may acknowledge this access and decode the following bytes and respond according to the value of the bytes.

*GACC behavior* can be seen in [Figure 35-28 on page 777](#).

- **OVRE: Overrun Error (clear on read)**

This bit is only used in Master mode.

0 = TWI\_RHR has not been loaded while RXRDY was set

1 = TWI\_RHR has been loaded while RXRDY was set. Reset by read in TWI\_SR when TXCOMP is set.

- **NACK: Not Acknowledged (clear on read)**

NACK used in Master mode:

0 = Each data byte has been correctly received by the far-end side TWI slave component.

1 = A data byte has not been acknowledged by the slave component. Set at the same time as TXCOMP.

NACK used in Slave Read mode:

0 = Each data byte has been correctly received by the Master.

1 = In read mode, a data byte has not been acknowledged by the Master. When NACK is set the programmer must not fill TWI\_THR even if TXRDY is set, because it means that the Master will stop the data transfer or re initiate it.

Note that in Slave Write mode all data are acknowledged by the TWI.

- **ARBLST: Arbitration Lost (clear on read)**

This bit is only used in Master mode.

0: Arbitration won.

1: Arbitration lost. Another master of the TWI bus has won the multi-master arbitration. TXCOMP is set at the same time.

- **SCLWS: Clock Wait State (automatically set / reset)**

This bit is only used in Slave mode.

0 = The clock is not stretched.

1 = The clock is stretched. TWI\_THR / TWI\_RHR buffer is not filled / emptied before the emission / reception of a new character.

*SCLWS* behavior can be seen in [Figure 35-29 on page 778](#) and [Figure 35-30 on page 779](#).

- **EOSACC: End Of Slave Access (clear on read)**

This bit is only used in Slave mode.

0 = A slave access is being performing.

1 = The Slave Access is finished. End Of Slave Access is automatically set as soon as SVACC is reset.

*EOSACC* behavior can be seen in [Figure 35-31 on page 780](#) and [Figure 35-32 on page 780](#)

- **ENDRX: End of RX buffer**

0 = The Receive Counter Register has not reached 0 since the last write in TWI\_RCR or TWI\_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in TWI\_RCR or TWI\_RNCR.

- **ENDTX: End of TX buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in TWI\_TCR or TWI\_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in TWI\_TCR or TWI\_TNCR.

- **RXBUFF: RX Buffer Full**

0 = TWI\_RCR or TWI\_RNCR have a value other than 0.

1 = Both TWI\_RCR and TWI\_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = TWI\_TCR or TWI\_TNCR have a value other than 0.

1 = Both TWI\_TCR and TWI\_TNCR have a value of 0.

### 35.12.7 TWI Interrupt Enable Register

Name: TWI\_IER

Address: 0x400A8024 (0), 0x400AC024 (1)

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK
7	6	5	4	3	2	1	0
-	OVRE	GACC	SVACC	-	TXRDY	RXRDY	TXCOMP

- TXCOMP: Transmission Completed Interrupt Enable
- RXRDY: Receive Holding Register Ready Interrupt Enable
- TXRDY: Transmit Holding Register Ready Interrupt Enable
- SVACC: Slave Access Interrupt Enable
- GACC: General Call Access Interrupt Enable
- OVRE: Overrun Error Interrupt Enable
- NACK: Not Acknowledge Interrupt Enable
- ARBLST: Arbitration Lost Interrupt Enable
- SCL\_WS: Clock Wait State Interrupt Enable
- EOSACC: End Of Slave Access Interrupt Enable
- ENDRX: End of Receive Buffer Interrupt Enable
- ENDTX: End of Transmit Buffer Interrupt Enable
- RXBUFF: Receive Buffer Full Interrupt Enable
- TXBUFE: Transmit Buffer Empty Interrupt Enable

0 = No effect.

1 = Enables the corresponding interrupt.

### 35.12.8 TWI Interrupt Disable Register

**Name:** TWI\_IDR

**Address:** 0x400A8028 (0), 0x400AC028 (1)

**Access:** Write-only

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK
7	6	5	4	3	2	1	0
-	OVRE	GACC	SVACC	-	TXRDY	RXRDY	TXCOMP

- **TXCOMP:** Transmission Completed Interrupt Disable
- **RXRDY:** Receive Holding Register Ready Interrupt Disable
- **TXRDY:** Transmit Holding Register Ready Interrupt Disable
- **SVACC:** Slave Access Interrupt Disable
- **GACC:** General Call Access Interrupt Disable
- **OVRE:** Overrun Error Interrupt Disable
- **NACK:** Not Acknowledge Interrupt Disable
- **ARBLST:** Arbitration Lost Interrupt Disable
- **SCL\_WS:** Clock Wait State Interrupt Disable
- **EOSACC:** End Of Slave Access Interrupt Disable
- **ENDRX:** End of Receive Buffer Interrupt Disable
- **ENDTX:** End of Transmit Buffer Interrupt Disable
- **RXBUFF:** Receive Buffer Full Interrupt Disable
- **TXBUFE:** Transmit Buffer Empty Interrupt Disable

0 = No effect.

1 = Disables the corresponding interrupt.

### 35.12.9 TWI Interrupt Mask Register

**Name:** TWI\_IMR

**Address:** 0x400A802C (0), 0x400AC02C (1)

**Access:** Read-only

**Reset:** 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK
7	6	5	4	3	2	1	0
-	OVRE	GACC	SVACC	-	TXRDY	RXRDY	TXCOMP

- **TXCOMP:** Transmission Completed Interrupt Mask
- **RXRDY:** Receive Holding Register Ready Interrupt Mask
- **TXRDY:** Transmit Holding Register Ready Interrupt Mask
- **SVACC:** Slave Access Interrupt Mask
- **GACC:** General Call Access Interrupt Mask
- **OVRE:** Overrun Error Interrupt Mask
- **NACK:** Not Acknowledge Interrupt Mask
- **ARBLST:** Arbitration Lost Interrupt Mask
- **SCL\_WS:** Clock Wait State Interrupt Mask
- **EOSACC:** End Of Slave Access Interrupt Mask
- **ENDRX:** End of Receive Buffer Interrupt Mask
- **ENDTX:** End of Transmit Buffer Interrupt Mask
- **RXBUFF:** Receive Buffer Full Interrupt Mask
- **TXBUFE:** Transmit Buffer Empty Interrupt Mask

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

### 35.12.10TWI Receive Holding Register

Name: TWI\_RHR

Address: 0x400A8030 (0), 0x400AC030 (1)

Access: Read-only

Reset: 0x00000000

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
RXDATA							

- RXDATA: Master or Slave Receive Holding Data

### 35.12.11TWI Transmit Holding Register

Name: TWI\_THR

Address: 0x400A8034 (0), 0x400AC034 (1)

Access: Read-write

Reset: 0x00000000

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
TXDATA							

- TXDATA: Master or Slave Transmit Holding Data

### 35.12.12TWI Write Protection Mode Register

**Name:** TWI\_WPMR

**Address:** 0x400A80E4 (0), 0x400AC0E4 (1)

**Access:** Read-write

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x545749 ("TWI" in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x545749 ("TWI" in ASCII).

The write protected registers are:

["TWI Clock Waveform Generator Register"](#)

["TWI Slave Mode Register"](#)

- **WPKEY: Write Protect Key**

Value	Name	Description
0x545749	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0

### 35.12.13TWI Write Protection Status Register

**Name:** TWI\_WPSR

**Address:** 0x400A80E8 (0), 0x400AC0E8 (1)

**Access:** Read-only

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

- **WPVS: Write Protect Violation Status**

0: No Write Protect Violation has occurred since the last read of the TWI\_WPSR register.

1: A Write Protect Violation has occurred since the last read of the TWI\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Note: Reading TWI\_WPSR automatically clears all fields.



## 36. Universal Asynchronous Receiver Transmitter (UART)

### 36.1 Description

The Universal Asynchronous Receiver Transmitter (UART) features a two-pin UART that can be used for communication and trace purposes and offers an ideal medium for in-situ programming solutions.

Moreover, the association with peripheral DMA controller (PDC) permits packet handling for these tasks with processor time reduced to a minimum.

### 36.2 Embedded Characteristics

- Two-pin UART
  - Independent receiver and transmitter with a common programmable Baud Rate Generator
  - Even, Odd, Mark or Space Parity Generation
  - Parity, Framing and Overrun Error Detection
  - Automatic Echo, Local Loopback and Remote Loopback Channel Modes
  - Support for two PDC channels with connection to receiver and transmitter

### 36.3 Block Diagram

Figure 36-1. UART Functional Block Diagram

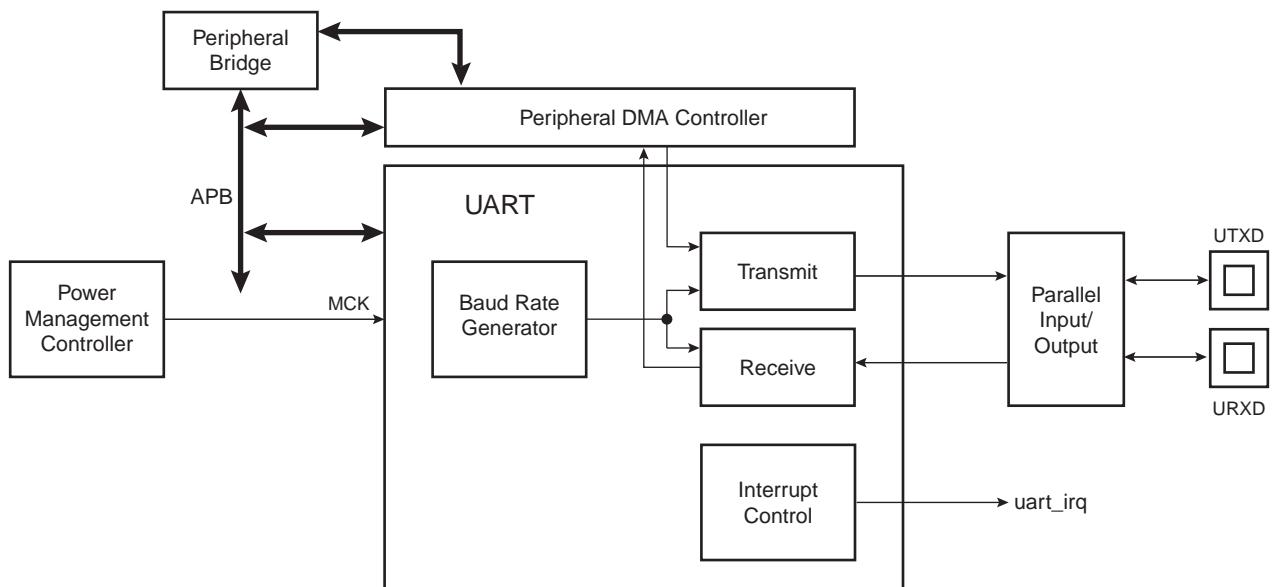


Table 36-1. UART Pin Description

Pin Name	Description	Type
URXD	UART Receive Data	Input
UTXD	UART Transmit Data	Output

## 36.4 Product Dependencies

### 36.4.1 I/O Lines

The UART pins are multiplexed with PIO lines. The programmer must first configure the corresponding PIO Controller to enable I/O line operations of the UART.

Table 36-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
UART0	URXD0	PA9	A
UART0	UTXD0	PA10	A
UART1	URXD1	PA5	C
UART1	UTXD1	PA6	C

### 36.4.2 Power Management

The UART clock is controllable through the Power Management Controller. In this case, the programmer must first configure the PMC to enable the UART clock. Usually, the peripheral identifier used for this purpose is 1.

### 36.4.3 Interrupt Source

The UART interrupt line is connected to one of the interrupt sources of the Interrupt Controller. Interrupt handling requires programming of the Interrupt Controller before configuring the UART.

## 36.5 UART Operations

The UART operates in asynchronous mode only and supports only 8-bit character handling (with parity). It has no clock pin.

The UART is made up of a receiver and a transmitter that operate independently, and a common baud rate generator. Receiver timeout and transmitter time guard are not implemented. However, all the implemented features are compatible with those of a standard USART.

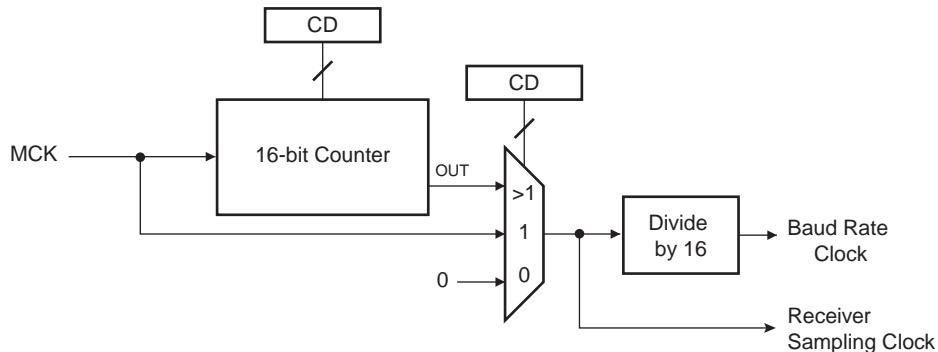
### 36.5.1 Baud Rate Generator

The baud rate generator provides the bit period clock named baud rate clock to both the receiver and the transmitter.

The baud rate clock is the master clock divided by 16 times the value (CD) written in UART\_BRGR (Baud Rate Generator Register). If UART\_BRGR is set to 0, the baud rate clock is disabled and the UART remains inactive. The maximum allowable baud rate is Master Clock divided by 16. The minimum allowable baud rate is Master Clock divided by (16 x 65536).

$$\text{Baud Rate} = \frac{\text{MCK}}{16 \times \text{CD}}$$

Figure 36-2. Baud Rate Generator



### 36.5.2 Receiver

#### 36.5.2.1 Receiver Reset, Enable and Disable

After device reset, the UART receiver is disabled and must be enabled before being used. The receiver can be enabled by writing the control register UART\_CR with the bit RXEN at 1. At this command, the receiver starts looking for a start bit.

The programmer can disable the receiver by writing UART\_CR with the bit RXDIS at 1. If the receiver is waiting for a start bit, it is immediately stopped. However, if the receiver has already detected a start bit and is receiving the data, it waits for the stop bit before actually stopping its operation.

The programmer can also put the receiver in its reset state by writing UART\_CR with the bit RSTRX at 1. In doing so, the receiver immediately stops its current operations and is disabled, whatever its current state. If RSTRX is applied when data is being processed, this data is lost.

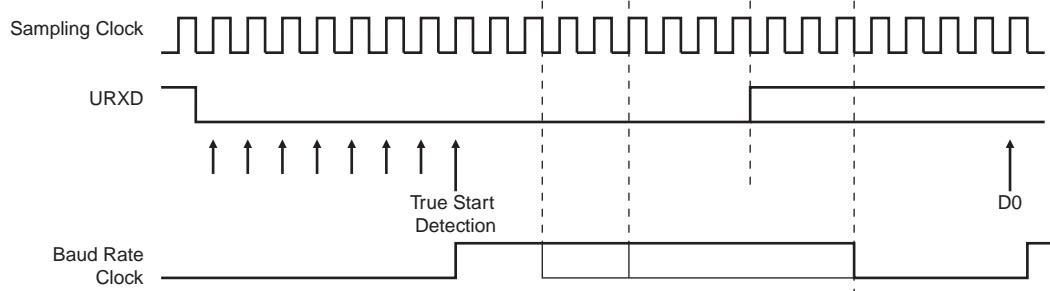
#### 36.5.2.2 Start Detection and Data Sampling

The UART only supports asynchronous operations, and this affects only its receiver. The UART receiver detects the start of a received character by sampling the URXD signal until it detects a valid start bit. A low level (space) on URXD is interpreted as a valid start bit if it is detected for more than 7 cycles of the sampling clock, which is 16 times the baud rate. Hence, a space that is longer than 7/16 of the bit period is detected as a valid start bit. A space which is 7/16 of a bit period or shorter is ignored and the receiver continues to wait for a valid start bit.

When a valid start bit has been detected, the receiver samples the URXD at the theoretical midpoint of each bit. It is assumed that each bit lasts 16 cycles of the sampling clock (1-bit period) so the bit sampling point is eight cycles (0.5-bit period) after the start of the bit. The first sampling point is therefore 24 cycles (1.5-bit periods) after the falling edge of the start bit was detected.

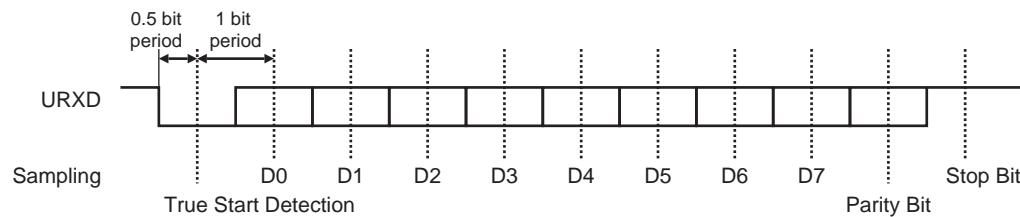
Each subsequent bit is sampled 16 cycles (1-bit period) after the previous one.

**Figure 36-3. Start Bit Detection**



**Figure 36-4. Character Reception**

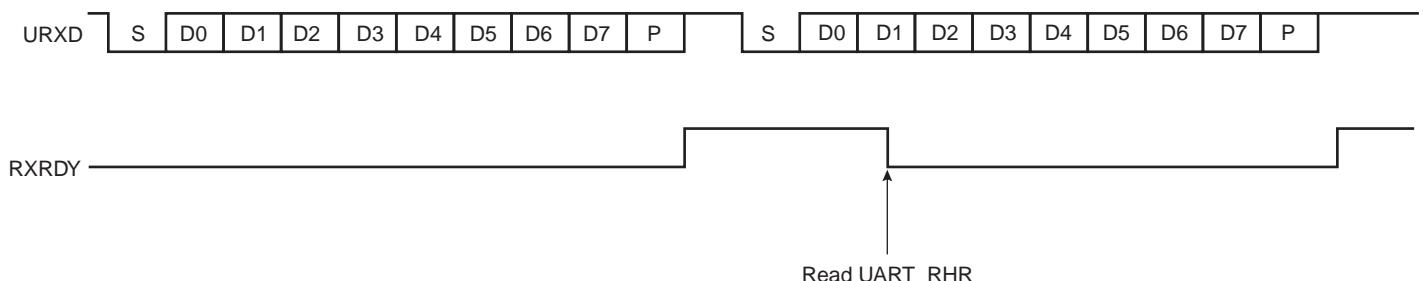
Example: 8-bit, parity enabled 1 stop



### 36.5.2.3 Receiver Ready

When a complete character is received, it is transferred to the UART\_RHR and the RXRDY status bit in UART\_SR (Status Register) is set. The bit RXRDY is automatically cleared when the receive holding register UART\_RHR is read.

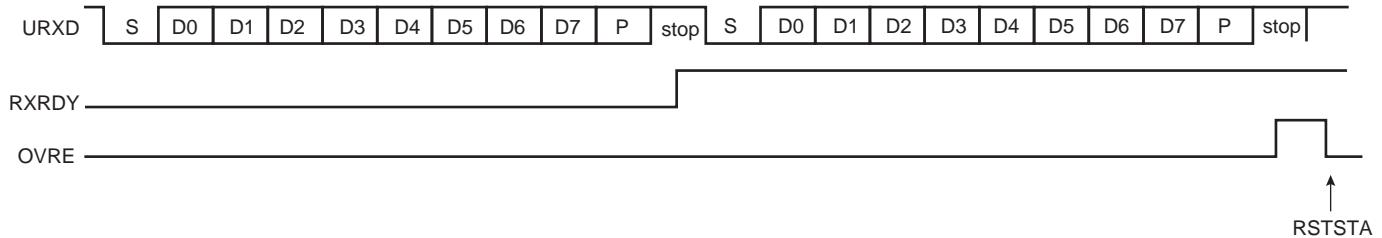
**Figure 36-5. Receiver Ready**



### 36.5.2.4 Receiver Overrun

If UART\_RHR has not been read by the software (or the Peripheral Data Controller or DMA Controller) since the last transfer, the RXRDY bit is still set and a new character is received, the OVRE status bit in UART\_SR is set. OVRE is cleared when the software writes the control register UART\_CR with the bit RSTSTA (Reset Status) at 1.

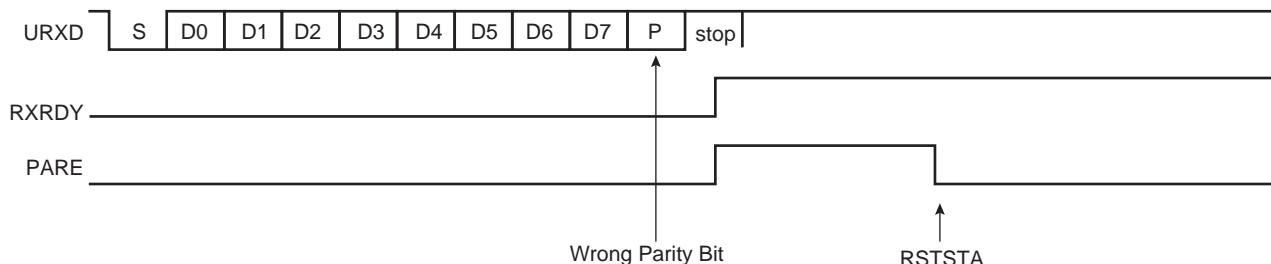
**Figure 36-6. Receiver Overrun**



### 36.5.2.5 Parity Error

Each time a character is received, the receiver calculates the parity of the received data bits, in accordance with the field PAR in UART\_MR. It then compares the result with the received parity bit. If different, the parity error bit PARE in UART\_SR is set at the same time the RXRDY is set. The parity bit is cleared when the control register UART\_CR is written with the bit RSTSTA (Reset Status) at 1. If a new character is received before the reset status command is written, the PARE bit remains at 1.

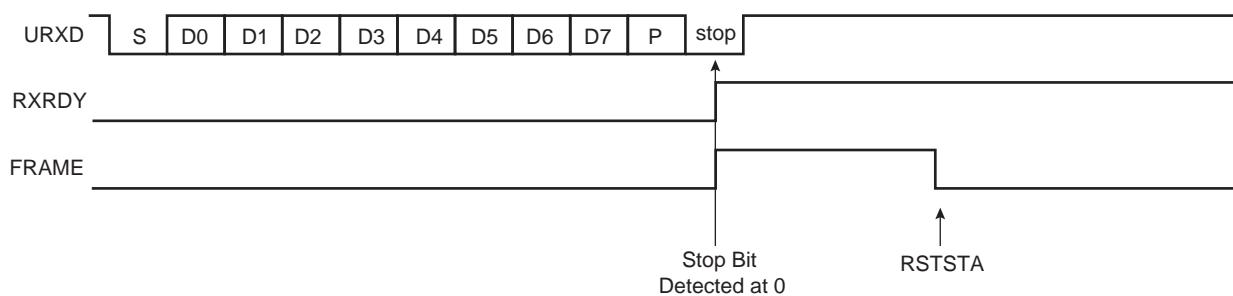
**Figure 36-7. Parity Error**



### 36.5.2.6 Receiver Framing Error

When a start bit is detected, it generates a character reception when all the data bits have been sampled. The stop bit is also sampled and when it is detected at 0, the FRAME (Framing Error) bit in UART\_SR is set at the same time the RXRDY bit is set. The FRAME bit remains high until the control register UART\_CR is written with the bit RSTSTA at 1.

**Figure 36-8. Receiver Framing Error**



### 36.5.3 Transmitter

#### 36.5.3.1 Transmitter Reset, Enable and Disable

After device reset, the UART transmitter is disabled and it must be enabled before being used. The transmitter is enabled by writing the control register `UART_CR` with the bit `TXEN` at 1. From this command, the transmitter waits for a character to be written in the Transmit Holding Register (`UART_THR`) before actually starting the transmission.

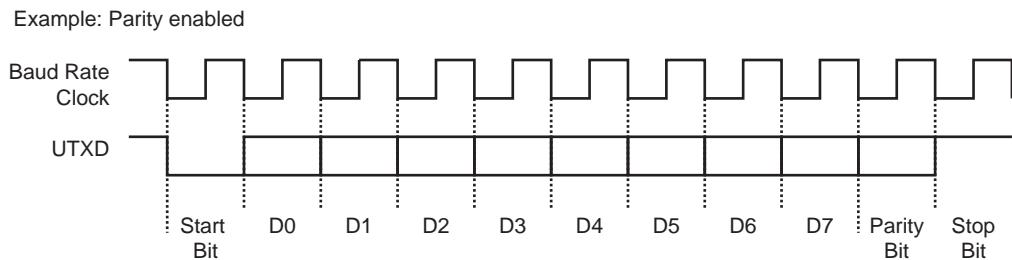
The programmer can disable the transmitter by writing `UART_CR` with the bit `TXDIS` at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the Shift Register and/or a character has been written in the Transmit Holding Register, the characters are completed before the transmitter is actually stopped.

The programmer can also put the transmitter in its reset state by writing the `UART_CR` with the bit `RSTTX` at 1. This immediately stops the transmitter, whether or not it is processing characters.

#### 36.5.3.2 Transmit Format

The UART transmitter drives the pin `UTXD` at the baud rate clock speed. The line is driven depending on the format defined in the Mode Register and the data stored in the Shift Register. One start bit at level 0, then the 8 data bits, from the lowest to the highest bit, one optional parity bit and one stop bit at 1 are consecutively shifted out as shown in the following figure. The field `PARE` in the mode register `UART_MR` defines whether or not a parity bit is shifted out. When a parity bit is enabled, it can be selected between an odd parity, an even parity, or a fixed space or mark bit.

**Figure 36-9. Character Transmission**

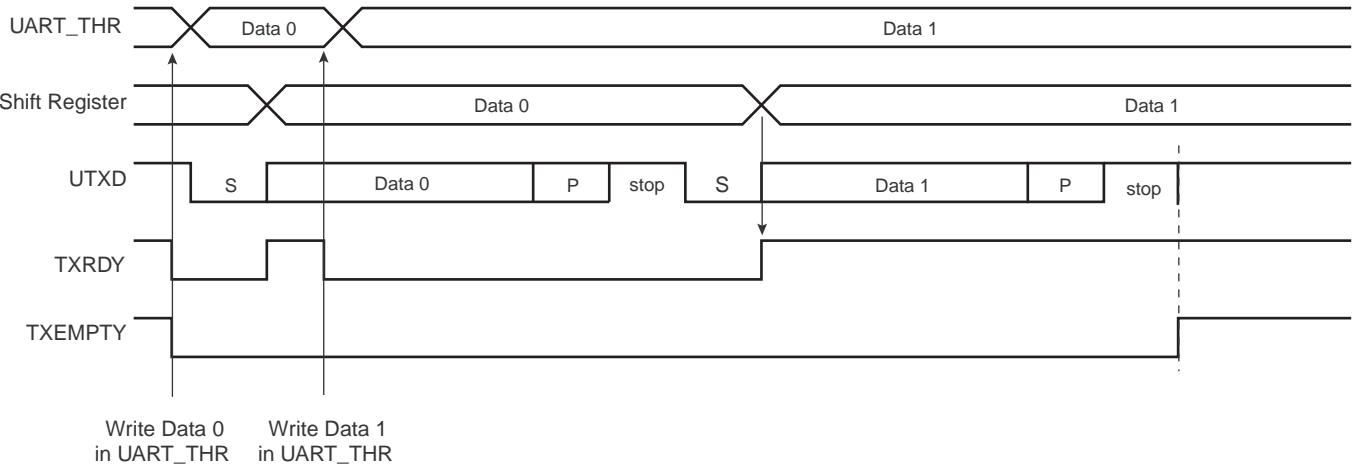


#### 36.5.3.3 Transmitter Control

When the transmitter is enabled, the bit `TXRDY` (Transmitter Ready) is set in the status register `UART_SR`. The transmission starts when the programmer writes in the Transmit Holding Register (`UART_THR`), and after the written character is transferred from `UART_THR` to the Shift Register. The `TXRDY` bit remains high until a second character is written in `UART_THR`. As soon as the first character is completed, the last character written in `UART_THR` is transferred into the shift register and `TXRDY` rises again, showing that the holding register is empty.

When both the Shift Register and `UART_THR` are empty, i.e., all the characters written in `UART_THR` have been processed, the `TXEMPTY` bit rises after the last stop bit has been completed.

**Figure 36-10. Transmitter Control**



#### 36.5.4 Peripheral DMA Controller

Both the receiver and the transmitter of the UART are connected to a Peripheral DMA Controller (PDC) channel.

The peripheral data controller channels are programmed via registers that are mapped within the UART user interface from the offset 0x100. The status bits are reported in the UART status register (UART\_SR) and can generate an interrupt.

The RXRDY bit triggers the PDC channel data transfer of the receiver. This results in a read of the data in UART\_RHR. The TXRDY bit triggers the PDC channel data transfer of the transmitter. This results in a write of data in UART THR.

#### 36.5.5 Test Modes

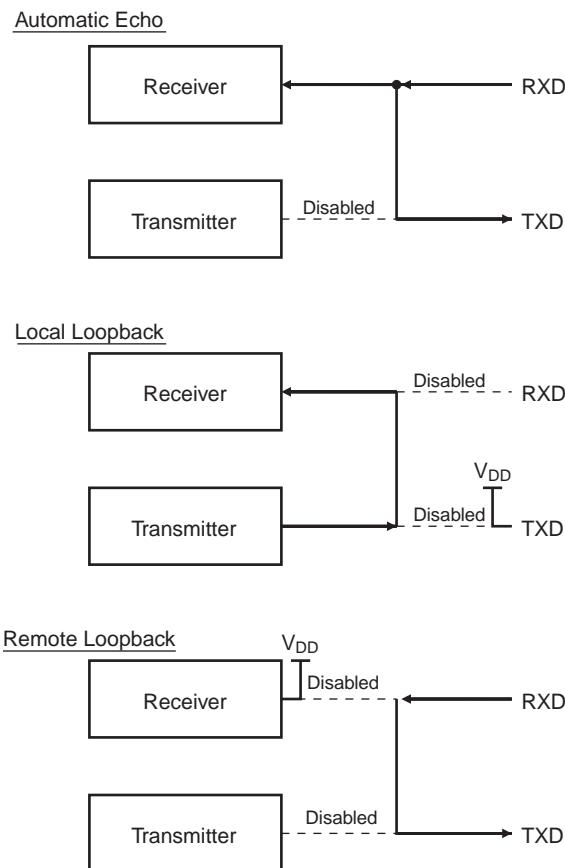
The UART supports three test modes. These modes of operation are programmed by using the field CHMODE (Channel Mode) in the mode register (UART\_MR).

The Automatic Echo mode allows bit-by-bit retransmission. When a bit is received on the URXD line, it is sent to the UTXD line. The transmitter operates normally, but has no effect on the UTXD line.

The Local Loopback mode allows the transmitted characters to be received. UTXD and URXD pins are not used and the output of the transmitter is internally connected to the input of the receiver. The URXD pin level has no effect and the UTXD line is held high, as in idle state.

The Remote Loopback mode directly connects the URXD pin to the UTXD line. The transmitter and the receiver are disabled and have no effect. This mode allows a bit-by-bit retransmission.

**Figure 36-11. Test Modes**



## 36.6 Universal Asynchronous Receiver Transmitter (UART) User Interface

Table 36-3. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Control Register	UART_CR	Write-only	–
0x0004	Mode Register	UART_MR	Read-write	0x0
0x0008	Interrupt Enable Register	UART_IER	Write-only	–
0x000C	Interrupt Disable Register	UART_IDR	Write-only	–
0x0010	Interrupt Mask Register	UART_IMR	Read-only	0x0
0x0014	Status Register	UART_SR	Read-only	–
0x0018	Receive Holding Register	UART_RHR	Read-only	0x0
0x001C	Transmit Holding Register	UART_THR	Write-only	–
0x0020	Baud Rate Generator Register	UART_BRGR	Read-write	0x0
0x0024 - 0x003C	Reserved	–	–	–
0x004C - 0x00FC	Reserved	–	–	–
0x0100 - 0x0124	Reserved for PDC registers	–	–	–

### 36.6.1 UART Control Register

**Name:** UART\_CR

**Address:** 0x400E0600 (0), 0x40060600 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	—	—

- **RSTRX: Reset Receiver**

0 = No effect.

1 = The receiver logic is reset and disabled. If a character is being received, the reception is aborted.

- **RSTTX: Reset Transmitter**

0 = No effect.

1 = The transmitter logic is reset and disabled. If a character is being transmitted, the transmission is aborted.

- **RXEN: Receiver Enable**

0 = No effect.

1 = The receiver is enabled if RXDIS is 0.

- **RXDIS: Receiver Disable**

0 = No effect.

1 = The receiver is disabled. If a character is being processed and RSTRX is not set, the character is completed before the receiver is stopped.

- **TXEN: Transmitter Enable**

0 = No effect.

1 = The transmitter is enabled if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0 = No effect.

1 = The transmitter is disabled. If a character is being processed and a character has been written in the UART\_THR and RSTTX is not set, both characters are completed before the transmitter is stopped.

- **RSTSTA: Reset Status Bits**

0 = No effect.

1 = Resets the status bits PARE, FRAME and OVRE in the UART\_SR.

### 36.6.2 UART Mode Register

**Name:** UART\_MR

**Address:** 0x400E0604 (0), 0x40060604 (1)

**Access:** Read-write

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

- **PAR: Parity Type**

Value	Name	Description
0	EVEN	Even Parity
1	ODD	Odd Parity
2	SPACE	Space: parity forced to 0
3	MARK	Mark: parity forced to 1
4	NO	No Parity

- **CHMODE: Channel Mode**

Value	Name	Description
0	NORMAL	Normal Mode
1	AUTOMATIC	Automatic Echo
2	LOCAL_LOOPBACK	Local Loopback
3	REMOTE_LOOPBACK	Remote Loopback

### 36.6.3 UART Interrupt Enable Register

**Name:** UART\_IER

**Address:** 0x400E0608 (0), 0x40060608 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	RXBUFF	TXBUFE	—	TXEMPTY	—
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	—	TXRDY	RXRDY

- **RXRDY:** Enable RXRDY Interrupt
- **TXRDY:** Enable TXRDY Interrupt
- **ENDRX:** Enable End of Receive Transfer Interrupt
- **ENDTX:** Enable End of Transmit Interrupt
- **OVRE:** Enable Overrun Error Interrupt
- **FRAME:** Enable Framing Error Interrupt
- **PARE:** Enable Parity Error Interrupt
- **TXEMPTY:** Enable TXEMPTY Interrupt
- **TXBUFE:** Enable Buffer Empty Interrupt
- **RXBUFF:** Enable Buffer Full Interrupt

0 = No effect.

1 = Enables the corresponding interrupt.

#### 36.6.4 UART Interrupt Disable Register

**Name:** UART\_IDR

**Address:** 0x400E060C (0), 0x4006060C (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	RXBUFF	TXBUFE	—	TXEMPTY	—
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	—	TXRDY	RXRDY

- **RXRDY:** Disable RXRDY Interrupt
- **TXRDY:** Disable TXRDY Interrupt
- **ENDRX:** Disable End of Receive Transfer Interrupt
- **ENDTX:** Disable End of Transmit Interrupt
- **OVRE:** Disable Overrun Error Interrupt
- **FRAME:** Disable Framing Error Interrupt
- **PARE:** Disable Parity Error Interrupt
- **TXEMPTY:** Disable TXEMPTY Interrupt
- **TXBUFE:** Disable Buffer Empty Interrupt
- **RXBUFF:** Disable Buffer Full Interrupt

0 = No effect.

1 = Disables the corresponding interrupt.

### 36.6.5 UART Interrupt Mask Register

**Name:** UART\_IMR

**Address:** 0x400E0610 (0), 0x40060610 (1)

**Access:** Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	RXBUFF	TXBUFE	—	TXEMPTY	—
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	—	TXRDY	RXRDY

- **RXRDY:** Mask RXRDY Interrupt
- **TXRDY:** Disable TXRDY Interrupt
- **ENDRX:** Mask End of Receive Transfer Interrupt
- **ENDTX:** Mask End of Transmit Interrupt
- **OVRE:** Mask Overrun Error Interrupt
- **FRAME:** Mask Framing Error Interrupt
- **PARE:** Mask Parity Error Interrupt
- **TXEMPTY:** Mask TXEMPTY Interrupt
- **TXBUFE:** Mask TXBUFE Interrupt
- **RXBUFF:** Mask RXBUFF Interrupt

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

### 36.6.6 UART Status Register

**Name:** UART\_SR

**Address:** 0x400E0614 (0), 0x40060614 (1)

**Access:** Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
—	—	—	RXBUFF	TXBUFE	—	TXEMPTY	—
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	—	TXRDY	RXRDY

- **RXRDY: Receiver Ready**

0 = No character has been received since the last read of the UART\_RHR or the receiver is disabled.

1 = At least one complete character has been received, transferred to UART\_RHR and not yet read.

- **TXRDY: Transmitter Ready**

0 = A character has been written to UART\_THR and not yet transferred to the Shift Register, or the transmitter is disabled.

1 = There is no character written to UART\_THR not yet transferred to the Shift Register.

- **ENDRX: End of Receiver Transfer**

0 = The End of Transfer signal from the receiver Peripheral Data Controller channel is inactive.

1 = The End of Transfer signal from the receiver Peripheral Data Controller channel is active.

- **ENDTX: End of Transmitter Transfer**

0 = The End of Transfer signal from the transmitter Peripheral Data Controller channel is inactive.

1 = The End of Transfer signal from the transmitter Peripheral Data Controller channel is active.

- **OVRE: Overrun Error**

0 = No overrun error has occurred since the last RSTSTA.

1 = At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0 = No framing error has occurred since the last RSTSTA.

1 = At least one framing error has occurred since the last RSTSTA.

- **PARE: Parity Error**

0 = No parity error has occurred since the last RSTSTA.

1 = At least one parity error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty**

0 = There are characters in UART\_THR, or characters being processed by the transmitter, or the transmitter is disabled.

1 = There are no characters in UART\_THR and there are no characters being processed by the transmitter.

- **TXBUFE: Transmission Buffer Empty**

0 = The buffer empty signal from the transmitter PDC channel is inactive.

1 = The buffer empty signal from the transmitter PDC channel is active.

- **RXBUFF: Receive Buffer Full**

0 = The buffer full signal from the receiver PDC channel is inactive.

1 = The buffer full signal from the receiver PDC channel is active.

### 36.6.7 UART Receiver Holding Register

**Name:** UART\_RHR

**Address:** 0x400E0618 (0), 0x40060618 (1)

**Access:** Read-only

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
RXCHR							

- **RXCHR: Received Character**

Last received character if RXRDY is set.

### 36.6.8 UART Transmit Holding Register

**Name:** UART\_THR

**Address:** 0x400E061C (0), 0x4006061C (1)

**Access:** Write-only

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
TXCHR							

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

### 36.6.9 UART Baud Rate Generator Register

**Name:** UART\_BRGR

**Address:** 0x400E0620 (0), 0x40060620 (1)

**Access:** Read-write

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

- **CD: Clock Divisor**

0 = Baud Rate Clock is disabled

1 to 65,535 = MCK / (CD x 16)

# 37. Universal Synchronous Asynchronous Receiver Transmitter (USART)

## 37.1 Description

The Universal Synchronous Asynchronous Receiver Transceiver (USART) provides one full duplex universal synchronous asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver time-out enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission.

The USART features three test modes: remote loopback, local loopback and automatic echo.

The USART supports specific operating modes providing interfaces on RS485 and SPI buses, with ISO7816 T = 0 or T = 1 smart card slots, infrared transceivers and connection to modem ports. The hardware handshaking feature enables an out-of-band flow control by automatic management of the pins RTS and CTS.

The USART supports the connection to the DMA Controller and the Peripheral DMA Controller, which enables data transfers to the transmitter and from the receiver. The PDC and DMAC provide chained buffer management without any intervention of the processor.

## 37.2 Embedded Characteristics

- Programmable Baud Rate Generator
- 5- to 9-bit Full-duplex Synchronous or Asynchronous Serial Communications
  - 1, 1.5 or 2 Stop Bits in Asynchronous Mode or 1 or 2 Stop Bits in Synchronous Mode
  - Parity Generation and Error Detection
  - Framing Error Detection, Overrun Error Detection
  - MSB- or LSB-first
  - Optional Break Generation and Detection
  - By 8 or by 16 Over-sampling Receiver Frequency
  - Optional Hardware Handshaking RTS-CTS
  - Optional Modem Signal Management DTR-DSR-DCD-RI
  - Receiver Time-out and Transmitter Timeguard
  - Optional Multidrop Mode with Address Generation and Detection
- RS485 with Driver Control Signal
- ISO7816, T = 0 or T = 1 Protocols for Interfacing with Smart Cards
  - NACK Handling, Error Counter with Repetition and Iteration Limit
- IrDA Modulation and Demodulation
  - Communication at up to 115.2 Kbps
- SPI Mode
  - Master or Slave
  - Serial Clock Programmable Phase and Polarity
  - SPI Serial Clock (SCK) Frequency up to Internal Clock Frequency MCK/6
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of:
  - Two DMA Controller Channels (DMAC) and Two Peripheral DMA Controller Channels (PDC)
- Offers Buffer Transfer without Processor Intervention

### 37.3 Block Diagram

Figure 37-1. USART Block Diagram

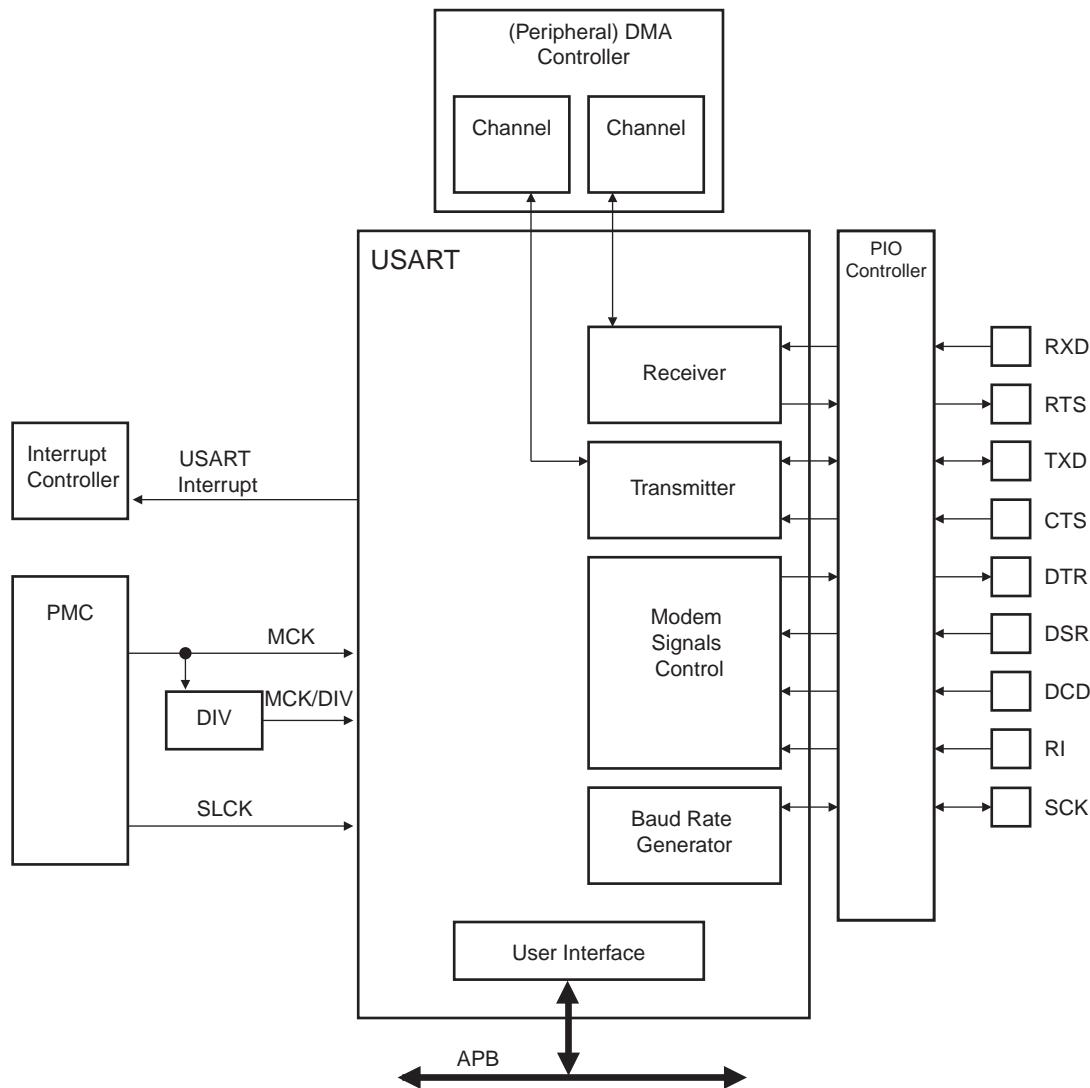
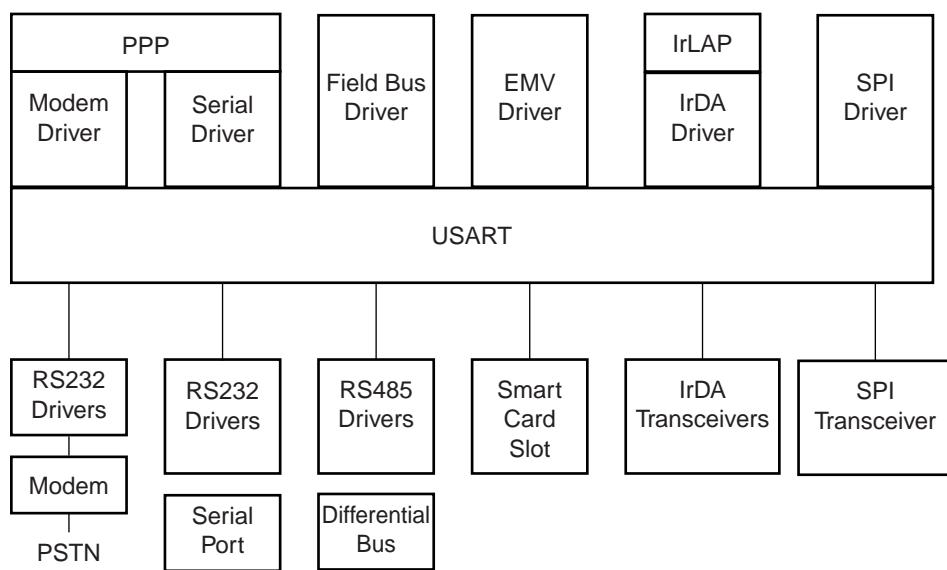


Table 37-1. SPI Operating Mode

PIN	USART	SPI Slave	SPI Master
RXD	RXD	MOSI	MISO
TXD	TXD	MISO	MOSI
RTS	RTS	–	CS
CTS	CTS	CS	–

## 37.4 Application Block Diagram

Figure 37-2. Application Block Diagram



## 37.5 I/O Lines Description

Table 37-2. I/O Line Description

Name	Description	Type	Active Level
SCK	Serial Clock	I/O	
TXD	Transmit Serial Data or Master Out Slave In (MOSI) in SPI Master Mode or Master In Slave Out (MISO) in SPI Slave Mode	I/O	
RXD	Receive Serial Data or Master In Slave Out (MISO) in SPI Master Mode or Master Out Slave In (MOSI) in SPI Slave Mode	Input	
RI	Ring Indicator	Input	Low
DSR	Data Set Ready	Input	Low
DCD	Data Carrier Detect	Input	Low
DTR	Data Terminal Ready	Output	Low
CTS	Clear to Send or Slave Select (NSS) in SPI Slave Mode	Input	Low
RTS	Request to Send or Slave Select (NSS) in SPI Master Mode	Output	Low

## 37.6 Product Dependencies

### 37.6.1 I/O Lines

The pins used for interfacing the USART may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired USART pins to their peripheral function. If I/O lines of the USART are not used by the application, they can be used for other purposes by the PIO Controller.

To prevent the TXD line from falling when the USART is disabled, the use of an internal pull up is mandatory. If the hardware handshaking feature or Modem mode is used, the internal pull up on TXD must also be enabled.

All the pins of the modems may or may not be implemented on the USART. Only USART is fully equipped with all the modem signals. On USARTs not equipped with the corresponding pin, the associated control bits and statuses have no effect on the behavior of the USART.

**Table 37-3. I/O Lines**

Instance	Signal	I/O Line	Peripheral
USART0	CTS0	PB2	C
USART0	RTS0	PB3	C
USART0	RXD0	PB0	C
USART0	SCK0	PB13	C
USART0	TXD0	PB1	C
USART1	CTS1	PA25	A
USART1	DCD1	PA26	A
USART1	DSR1	PA28	A
USART1	DTR1	PA27	A
USART1	RI1	PA29	A
USART1	RTS1	PA24	A
USART1	RXD1	PA21	A
USART1	SCK1	PA23	A
USART1	TXD1	PA22	A

### 37.6.2 Power Management

The USART is not continuously clocked. The programmer must first enable the USART Clock in the Power Management Controller (PMC) before using the USART. However, if the application does not require USART operations, the USART clock can be stopped when not needed and be restarted later. In this case, the USART will resume its operations where it left off.

Configuring the USART does not require the USART clock to be enabled.

### 37.6.3 Interrupt

The USART interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the USART

**Table 37-4. Peripheral IDs**

Instance	ID
USART0	14
USART1	15

interrupt requires the Interrupt Controller to be programmed first. Note that it is not recommended to use the USART interrupt line in edge sensitive mode.

## 37.7 Functional Description

The USART is capable of managing several types of serial synchronous or asynchronous communications.

It supports the following communication modes:

- 5- to 9-bit full-duplex asynchronous serial communication
  - MSB- or LSB-first
  - 1, 1.5 or 2 stop bits
  - Parity even, odd, marked, space or none
  - By 8 or by 16 over-sampling receiver frequency
  - Optional hardware handshaking
  - Optional modem signals management
  - Optional break management
  - Optional multidrop serial communication
- High-speed 5- to 9-bit full-duplex synchronous serial communication
  - MSB- or LSB-first
  - 1 or 2 stop bits
  - Parity even, odd, marked, space or none
  - By 8 or by 16 over-sampling frequency
  - Optional hardware handshaking
  - Optional modem signals management
  - Optional break management
  - Optional multidrop serial communication
- RS485 with driver control signal
- ISO7816, T0 or T1 protocols for interfacing with smart cards
  - NACK handling, error counter with repetition and iteration limit, inverted data.
- InfraRed IrDA Modulation and Demodulation
- SPI Mode
  - Master or Slave
  - Serial Clock Programmable Phase and Polarity
  - SPI Serial Clock (SCK) Frequency up to Internal Clock Frequency MCK/6
- Test modes
  - Remote loopback, local loopback, automatic echo

### 37.7.1 Baud Rate Generator

The Baud Rate Generator provides the bit period clock named the Baud Rate Clock to both the receiver and the transmitter.

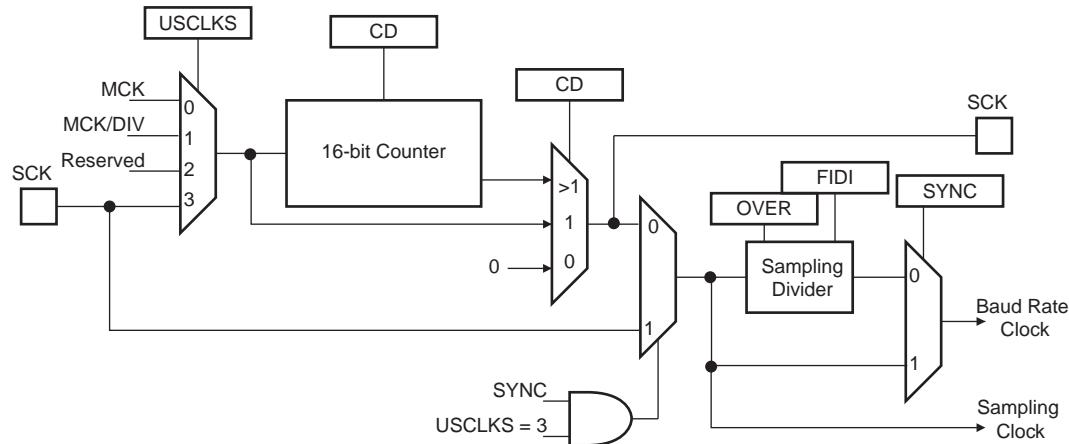
The Baud Rate Generator clock source can be selected by setting the USCLKS field in the Mode Register (US\_MR) between:

- The Master Clock MCK
- A division of the Master Clock, the divider being product dependent, but generally set to 8
- The external clock, available on the SCK pin

The Baud Rate Generator is based upon a 16-bit divider, which is programmed with the CD field of the Baud Rate Generator Register (US\_BRGR). If CD is programmed to 0, the Baud Rate Generator does not generate any clock. If CD is programmed to 1, the divider is bypassed and becomes inactive.

If the external SCK clock is selected, the duration of the low and high levels of the signal provided on the SCK pin must be longer than a Master Clock (MCK) period. The frequency of the signal provided on SCK must be at least 3 times lower than MCK in USART mode, or 6 times lower in SPI mode.

**Figure 37-3. Baud Rate Generator**



#### 37.7.1.1 Baud Rate in Asynchronous Mode

If the USART is programmed to operate in asynchronous mode, the selected clock is first divided by CD, which is field programmed in the Baud Rate Generator Register (US\_BRGR). The resulting clock is provided to the receiver as a sampling clock and then divided by 16 or 8, depending on the programming of the OVER bit in US\_MR.

If OVER is set to 1, the receiver sampling is 8 times higher than the baud rate clock. If OVER is cleared, the sampling is performed at 16 times the baud rate clock.

The following formula performs the calculation of the Baud Rate.

$$\text{Baudrate} = \frac{\text{SelectedClock}}{(8(2 - \text{Over})\text{CD})}$$

This gives a maximum baud rate of MCK divided by 8, assuming that MCK is the highest possible clock and that OVER is programmed to 1.

*Baud Rate Calculation Example*

**Table 37-5** shows calculations of CD to obtain a baud rate at 38400 bauds for different source clock frequencies. This table also shows the actual resulting baud rate and the error.

**Table 37-5. Baud Rate Example (OVER = 0)**

Source Clock	Expected Baud Rate	Calculation Result	CD	Actual Baud Rate	Error
MHz	Bit/s			Bit/s	
3 686 400	38 400	6.00	6	38 400.00	0.00%
4 915 200	38 400	8.00	8	38 400.00	0.00%
5 000 000	38 400	8.14	8	39 062.50	1.70%
7 372 800	38 400	12.00	12	38 400.00	0.00%
8 000 000	38 400	13.02	13	38 461.54	0.16%
12 000 000	38 400	19.53	20	37 500.00	2.40%
12 288 000	38 400	20.00	20	38 400.00	0.00%
14 318 180	38 400	23.30	23	38 908.10	1.31%
14 745 600	38 400	24.00	24	38 400.00	0.00%
18 432 000	38 400	30.00	30	38 400.00	0.00%
24 000 000	38 400	39.06	39	38 461.54	0.16%
24 576 000	38 400	40.00	40	38 400.00	0.00%
25 000 000	38 400	40.69	40	38 109.76	0.76%
32 000 000	38 400	52.08	52	38 461.54	0.16%
32 768 000	38 400	53.33	53	38 641.51	0.63%
33 000 000	38 400	53.71	54	38 194.44	0.54%
40 000 000	38 400	65.10	65	38 461.54	0.16%
50 000 000	38 400	81.38	81	38 580.25	0.47%

The baud rate is calculated with the following formula:

$$BaudRate = MCK/CD \times 16$$

The baud rate error is calculated with the following formula. It is not recommended to work with an error higher than 5%.

$$Error = 1 - \left( \frac{ExpectedBaudRate}{ActualBaudRate} \right)$$

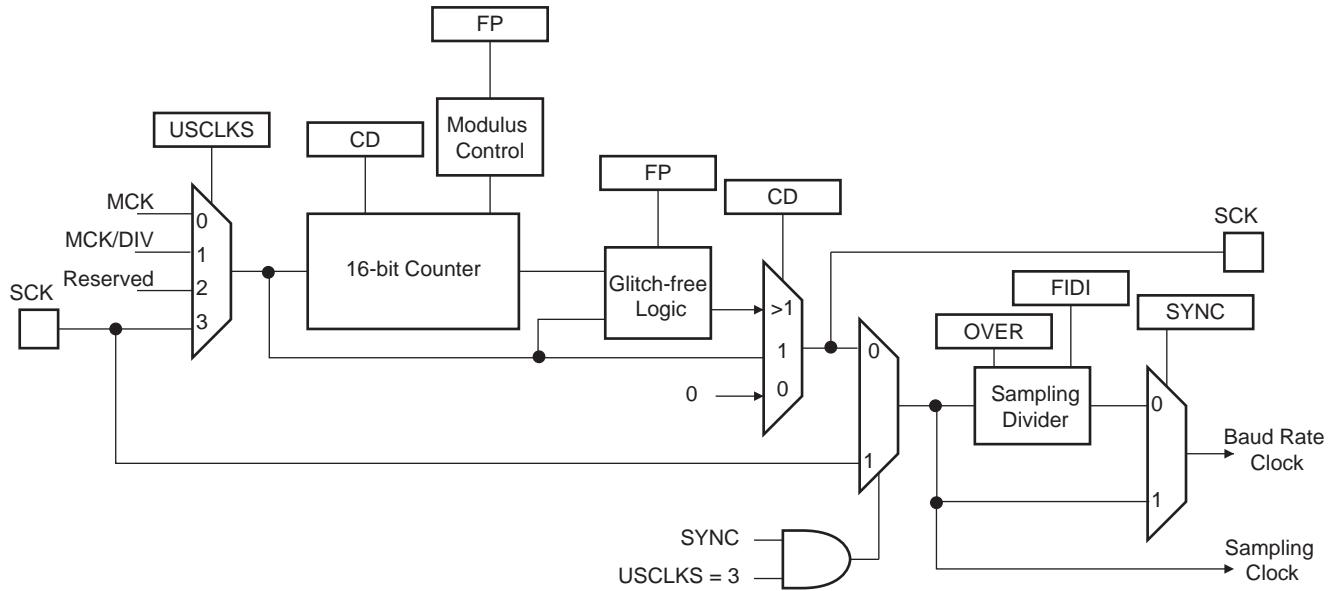
### 37.7.1.2 Fractional Baud Rate in Asynchronous Mode

The Baud Rate generator previously defined is subject to the following limitation: the output frequency changes by only integer multiples of the reference frequency. An approach to this problem is to integrate a fractional N clock generator that has a high resolution. The generator architecture is modified to obtain Baud Rate changes by a fraction of the reference source clock. This fractional part is programmed with the FP field in the Baud Rate Generator Register (US\_BRGR). If FP is not 0, the fractional part is activated. The resolution is one eighth of the clock divider. This feature is only available when using USART normal mode. The fractional Baud Rate is calculated using the following formula:

$$Baudrate = \frac{SelectedClock}{\left( 8(2 - Over) \left( CD + \frac{FP}{8} \right) \right)}$$

The modified architecture is presented below:

**Figure 37-4. Fractional Baud Rate Generator**



### 37.7.1.3 Baud Rate in Synchronous Mode or SPI Mode

If the USART is programmed to operate in synchronous mode, the selected clock is simply divided by the field CD in US\_BRGR.

$$\text{BaudRate} = \frac{\text{SelectedClock}}{\text{CD}}$$

In synchronous mode, if the external clock is selected (USCLKS = 3), the clock is provided directly by the signal on the USART SCK pin. No division is active. The value written in US\_BRGR has no effect. The external clock frequency must be at least 3 times lower than the system clock. In synchronous mode master (USCLKS = 0 or 1, CLK0 set to 1), the receive part limits the SCK maximum frequency to MCK/3 in USART mode, or MCK/6 in SPI mode.

When either the external clock SCK or the internal clock divided (MCK/DIV) is selected, the value programmed in CD must be even if the user has to ensure a 50:50 mark/space ratio on the SCK pin. If the internal clock MCK is selected, the Baud Rate Generator ensures a 50:50 duty cycle on the SCK pin, even if the value programmed in CD is odd.

### 37.7.1.4 Baud Rate in ISO 7816 Mode

The ISO7816 specification defines the bit rate with the following formula:

$$B = \frac{D_i}{F_i} \times f$$

where:

- B is the bit rate
- Di is the bit-rate adjustment factor
- Fi is the clock frequency division factor
- f is the ISO7816 clock frequency (Hz)

Di is a binary value encoded on a 4-bit field, named DI, as represented in Table 37-6.

**Table 37-6. Binary and Decimal Values for Di**

DI field	0001	0010	0011	0100	0101	0110	1000	1001
Di (decimal)	1	2	4	8	16	32	12	20

$F_i$  is a binary value encoded on a 4-bit field, named  $F_i$ , as represented in Table 37-7.

**Table 37-7. Binary and Decimal Values for  $F_i$**

$F_i$ field	0000	0001	0010	0011	0100	0101	0110	1001	1010	1011	1100	1101
$F_i$ (decimal)	372	372	558	744	1116	1488	1860	512	768	1024	1536	2048

Table 37-8 shows the resulting  $F_i/D_i$  Ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

**Table 37-8. Possible Values for the  $F_i/D_i$  Ratio**

$F_i/D_i$	372	558	774	1116	1488	1806	512	768	1024	1536	2048
1	372	558	744	1116	1488	1860	512	768	1024	1536	2048
2	186	279	372	558	744	930	256	384	512	768	1024
4	93	139.5	186	279	372	465	128	192	256	384	512
8	46.5	69.75	93	139.5	186	232.5	64	96	128	192	256
16	23.25	34.87	46.5	69.75	93	116.2	32	48	64	96	128
32	11.62	17.43	23.25	34.87	46.5	58.13	16	24	32	48	64
12	31	46.5	62	93	124	155	42.66	64	85.33	128	170.6
20	18.6	27.9	37.2	55.8	74.4	93	25.6	38.4	51.2	76.8	102.4

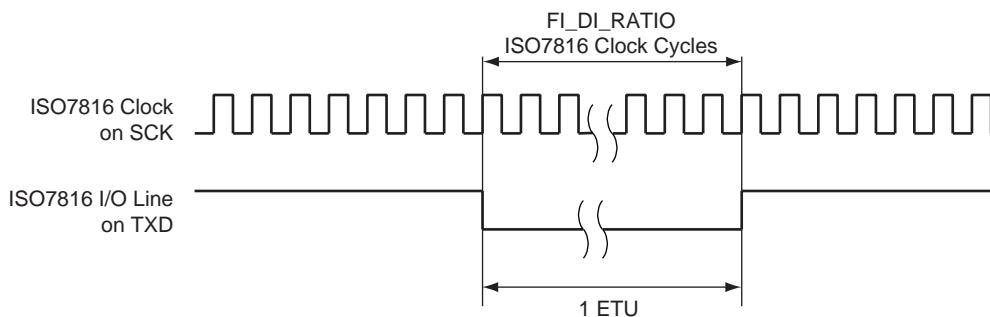
If the USART is configured in ISO7816 Mode, the clock selected by the USCLKS field in the Mode Register (US\_MR) is first divided by the value programmed in the field CD in the Baud Rate Generator Register (US\_BRGR). The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that the CLKO bit can be set in US\_MR.

This clock is then divided by the value programmed in the  $F_i/D_i$  Ratio field in the  $F_i/D_i$  Ratio register (US\_FIDI). This is performed by the Sampling Divider, which performs a division by up to  $F_i/D_i$  in ISO7816 Mode. The non-integer values of the  $F_i/D_i$  Ratio are not supported and the user must program the  $F_i/D_i$  Ratio field to a value as close as possible to the expected value.

The  $F_i/D_i$  Ratio field resets to the value 0x174 (372 in decimal) and is the most common divider between the ISO7816 clock and the bit rate ( $F_i = 372$ ,  $D_i = 1$ ).

Figure 37-5 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

**Figure 37-5. Elementary Time Unit (ETU)**



### 37.7.2 Receiver and Transmitter Control

After reset, the receiver is disabled. The user must enable the receiver by setting the RXEN bit in the Control Register (US\_CR). However, the receiver registers can be programmed before the receiver clock is enabled.

After reset, the transmitter is disabled. The user must enable it by setting the TXEN bit in the Control Register (US\_CR). However, the transmitter registers can be programmed before being enabled.

The Receiver and the Transmitter can be enabled together or independently.

At any time, the software can perform a reset on the receiver or the transmitter of the USART by setting the corresponding bit, RSTRX and RSTTX respectively, in the Control Register (US\_CR). The software resets clear the status flag and reset internal state machines but the user interface configuration registers hold the value configured prior to software reset. Regardless of what the receiver or the transmitter is performing, the communication is immediately stopped.

The user can also independently disable the receiver or the transmitter by setting RXDIS and TXDIS respectively in US\_CR. If the receiver is disabled during a character reception, the USART waits until the end of reception of the current character, then the reception is stopped. If the transmitter is disabled while it is operating, the USART waits the end of transmission of both the current character and character being stored in the Transmit Holding Register (US\_THR). If a timeguard is programmed, it is handled normally.

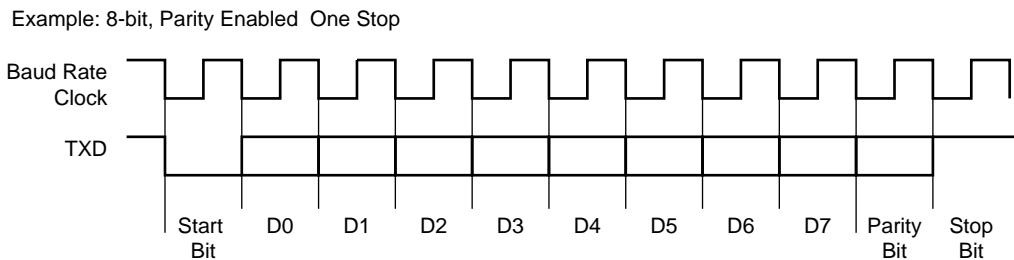
### 37.7.3 Synchronous and Asynchronous Modes

#### 37.7.3.1 Transmitter Operations

The transmitter performs the same in both synchronous and asynchronous operating modes (SYNC = 0 or SYNC = 1). One start bit, up to 9 data bits, one optional parity bit and up to two stop bits are successively shifted out on the TXD pin at each falling edge of the programmed serial clock.

The number of data bits is selected by the CHRL field and the MODE 9 bit in the Mode Register (US\_MR). Nine bits are selected by setting the MODE 9 bit regardless of the CHRL field. The parity bit is set according to the PAR field in US\_MR. The even, odd, space, marked or none parity bit can be configured. The MSBF field in US\_MR configures which data bit is sent first. If written to 1, the most significant bit is sent first. If written to 0, the less significant bit is sent first. The number of stop bits is selected by the NBSTOP field in US\_MR. The 1.5 stop bit is supported in asynchronous mode only.

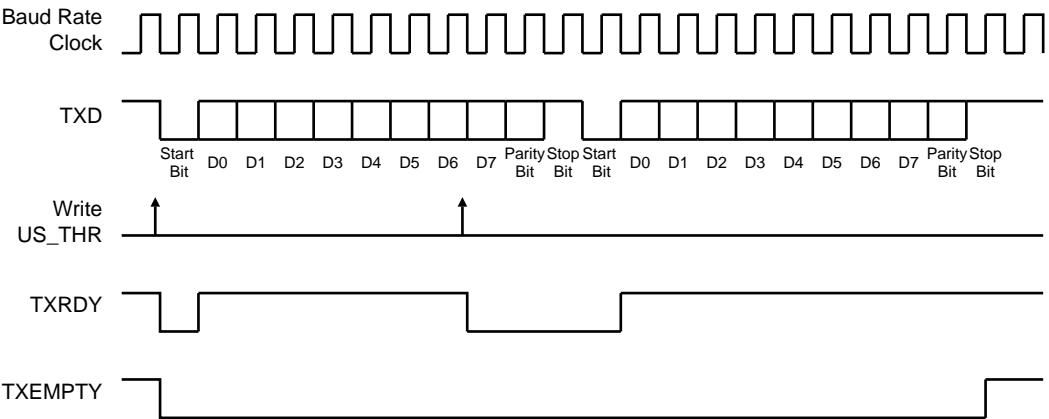
**Figure 37-6. Character Transmit**



The characters are sent by writing in the Transmit Holding Register (US\_THR). The transmitter reports two status bits in the Channel Status Register (US\_CSR): TXRDY (Transmitter Ready), which indicates that US\_THR is empty and TXEMPTY, which indicates that all the characters written in US\_THR have been processed. When the current character processing is completed, the last character written in US\_THR is transferred into the Shift Register of the transmitter and US\_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in US\_THR while TXRDY is low has no effect and the written character is lost.

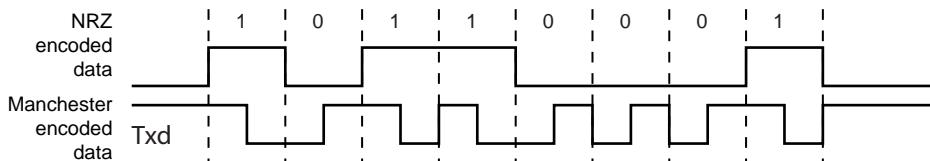
**Figure 37-7. Transmitter Status**



### 37.7.3.2 Manchester Encoder

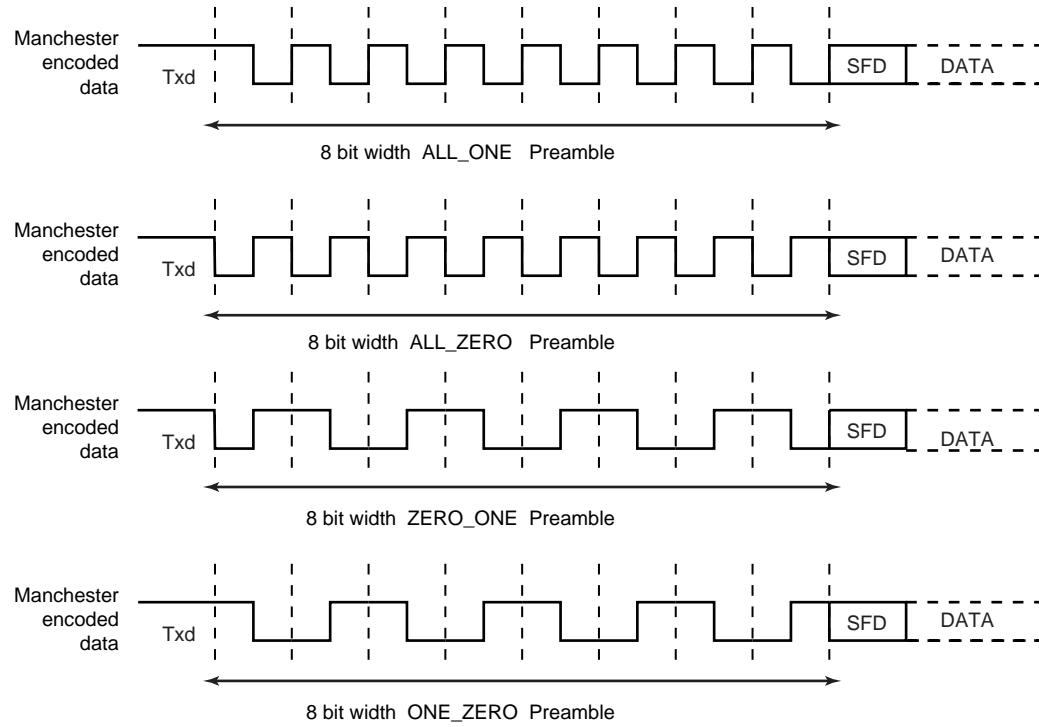
When the Manchester encoder is in use, characters transmitted through the USART are encoded based on biphase Manchester II format. To enable this mode, set the MAN field in the US\_MR register to 1. Depending on polarity configuration, a logic level (zero or one), is transmitted as a coded signal one-to-zero or zero-to-one. Thus, a transition always occurs at the midpoint of each bit time. It consumes more bandwidth than the original NRZ signal (2x) but the receiver has more error control since the expected input must show a change at the center of a bit cell. An example of Manchester encoded sequence is: the byte 0xB1 or 10110001 encodes to 10 01 10 10 01 01 01 10, assuming the default polarity of the encoder. [Figure 37-8](#) illustrates this coding scheme.

**Figure 37-8. NRZ to Manchester Encoding**



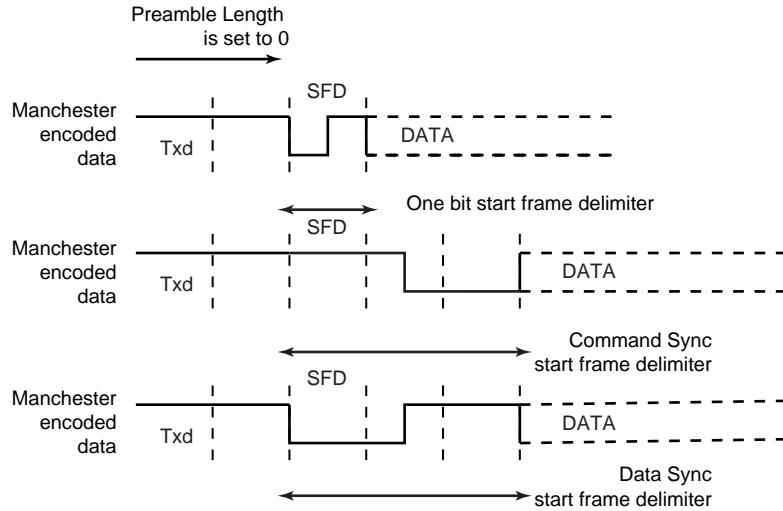
The Manchester encoded character can also be encapsulated by adding both a configurable preamble and a start frame delimiter pattern. Depending on the configuration, the preamble is a training sequence, composed of a pre-defined pattern with a programmable length from 1 to 15 bit times. If the preamble length is set to 0, the preamble waveform is not generated prior to any character. The preamble pattern is chosen among the following sequences: ALL\_ONE, ALL\_ZERO, ONE\_ZERO or ZERO\_ONE, writing the field TX\_PP in the US\_MAN register, the field TX\_PL is used to configure the preamble length. [Figure 37-9](#) illustrates and defines the valid patterns. To improve flexibility, the encoding scheme can be configured using the TX\_MPOL field in the US\_MAN register. If the TX\_MPOL field is set to zero (default), a logic zero is encoded with a zero-to-one transition and a logic one is encoded with a one-to-zero transition. If the TX\_MPOL field is set to one, a logic one is encoded with a one-to-zero transition and a logic zero is encoded with a zero-to-one transition.

**Figure 37-9. Preamble Patterns, Default Polarity Assumed**



A start frame delimiter is to be configured using the ONEBIT field in the US\_MR register. It consists of a user-defined pattern that indicates the beginning of a valid data. [Figure 37-10](#) illustrates these patterns. If the start frame delimiter, also known as the start bit, is one bit, (ONEBIT to 1), a logic zero is Manchester encoded and indicates that a new character is being sent serially on the line. If the start frame delimiter is a synchronization pattern also referred to as sync (ONEBIT to 0), a sequence of 3 bit times is sent serially on the line to indicate the start of a new character. The sync waveform is in itself an invalid Manchester waveform as the transition occurs at the middle of the second bit time. Two distinct sync patterns are used: the command sync and the data sync. The command sync has a logic one level for one and a half bit times, then a transition to logic zero for the second one and a half bit times. If the MODSYNC field in the US\_MR register is set to 1, the next character is a command. If it is set to 0, the next character is a data. When direct memory access is used, the MODSYNC field can be immediately updated with a modified character located in memory. To enable this mode, VAR\_SYNC field in US\_MR register must be set to 1. In this case, the MODSYNC field in US\_MR is bypassed and the sync configuration is held in the TXSYNH in the US\_THR register. The USART character format is modified and includes sync information.

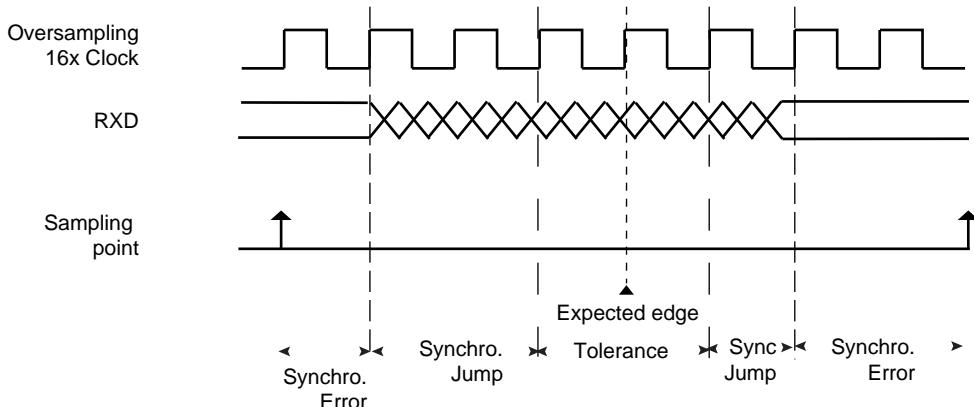
**Figure 37-10. Start Frame Delimiter**



#### Drift Compensation

Drift compensation is available only in 16X oversampling mode. An hardware recovery system allows a larger clock drift. To enable the hardware system, the bit in the USART\_MAN register must be set. If the RXD edge is one 16X clock cycle from the expected edge, this is considered as normal jitter and no corrective actions are taken. If the RXD event is between 4 and 2 clock cycles before the expected edge, then the current period is shortened by one clock cycle. If the RXD event is between 2 and 3 clock cycles after the expected edge, then the current period is lengthened by one clock cycle. These intervals are considered to be drift and so corrective actions are automatically taken.

**Figure 37-11. Bit Resynchronization**



#### 37.7.3.3 Asynchronous Receiver

If the USART is programmed in asynchronous operating mode (SYNC = 0), the receiver oversamples the RXD input line. The oversampling is either 16 or 8 times the Baud Rate clock, depending on the OVER bit in the Mode Register (US\_MR).

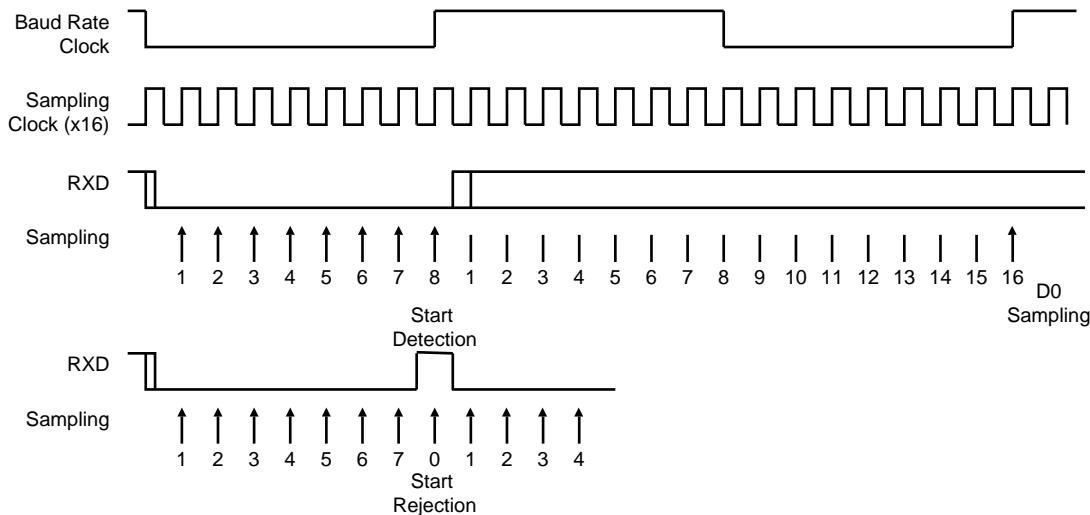
The receiver samples the RXD line. If the line is sampled during one half of a bit time to 0, a start bit is detected and data, parity and stop bits are successively sampled on the bit rate clock.

If the oversampling is 16, (OVER to 0), a start is detected at the eighth sample to 0. Then, data bits, parity bit and stop bit are sampled on each 16 sampling clock cycle. If the oversampling is 8 (OVER to 1), a start bit is detected at the fourth sample to 0. Then, data bits, parity bit and stop bit are sampled on each 8 sampling clock cycle.

The number of data bits, first bit sent and parity mode are selected by the same fields and bits as the transmitter, i.e. respectively CHRL, MODE9, MSBF and PAR. For the synchronization mechanism **only**, the number of stop bits has no effect on the receiver as it considers only one stop bit, regardless of the field NBSTOP, so that resynchronization between the receiver and the transmitter can occur. Moreover, as soon as the stop bit is sampled, the receiver starts looking for a new start bit so that resynchronization can also be accomplished when the transmitter is operating with one stop bit.

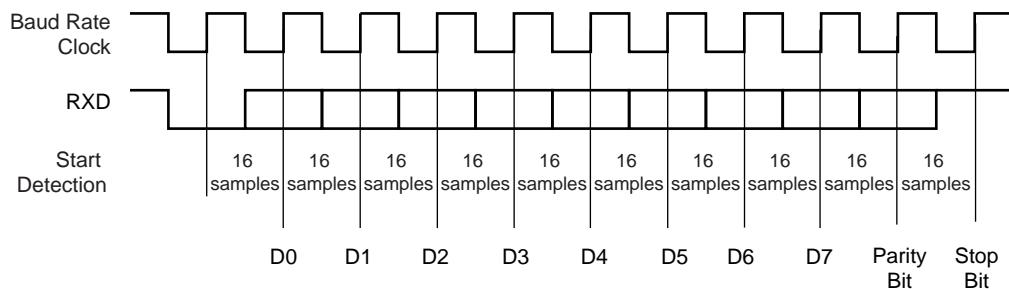
[Figure 37-12](#) and [Figure 37-13](#) illustrate start detection and character reception when USART operates in asynchronous mode.

**Figure 37-12. Asynchronous Start Detection**



**Figure 37-13. Asynchronous Character Reception**

Example: 8-bit, Parity Enabled



#### 37.7.3.4 Manchester Decoder

When the MAN field in US\_MR register is set to 1, the Manchester decoder is enabled. The decoder performs both preamble and start frame delimiter detection. One input line is dedicated to Manchester encoded input data.

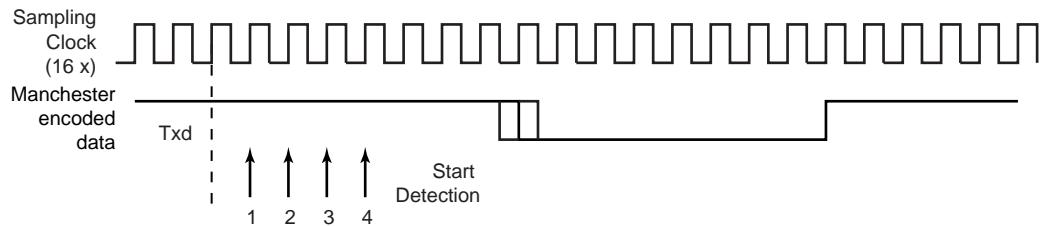
An optional preamble sequence can be defined, its length is user-defined and totally independent of the emitter side. Use RX\_PL in US\_MAN register to configure the length of the preamble sequence. If the length is set to 0, no preamble is detected and the function is disabled. In addition, the polarity of the input stream is programmable with RX\_MPOL field in US\_MAN register. Depending on the desired application the preamble pattern matching is to be defined via the RX\_PP field in US\_MAN. See [Figure 37-9](#) for available preamble patterns.

Unlike preamble, the start frame delimiter is shared between Manchester Encoder and Decoder. So, if ONEBIT field is set to 1, only a zero encoded Manchester can be detected as a valid start frame delimiter. If ONEBIT is set to 0, only a sync pattern is detected as a valid start frame delimiter. Decoder operates by detecting transition on incoming stream. If

RXD is sampled during one quarter of a bit time to zero, a start bit is detected. See [Figure 37-14](#). The sample pulse rejection mechanism applies.

In order to increase the compatibility the RXIDLV bit in the US\_MAN register allows to inform the USART block of the Rx line idle state value (Rx line undriven), it can be either level one (pull-up) or level zero (pull-down). By default this bit is set to one (Rx line is at level 1 if undriven).

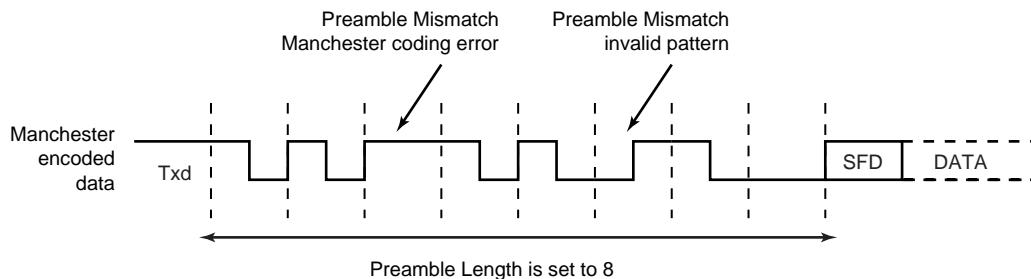
**Figure 37-14. Asynchronous Start Bit Detection**



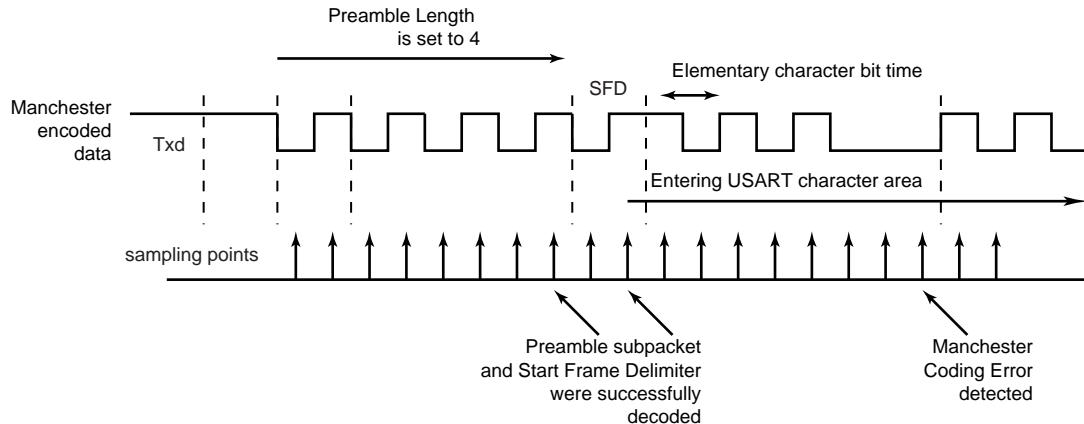
The receiver is activated and starts Preamble and Frame Delimiter detection, sampling the data at one quarter and then three quarters. If a valid preamble pattern or start frame delimiter is detected, the receiver continues decoding with the same synchronization. If the stream does not match a valid pattern or a valid start frame delimiter, the receiver resynchronizes on the next valid edge. The minimum time threshold to estimate the bit value is three quarters of a bit time.

If a valid preamble (if used) followed with a valid start frame delimiter is detected, the incoming stream is decoded into NRZ data and passed to USART for processing. [Figure 37-15](#) illustrates Manchester pattern mismatch. When incoming data stream is passed to the USART, the receiver is also able to detect Manchester code violation. A code violation is a lack of transition in the middle of a bit cell. In this case, MANE flag in US\_CSR register is raised. It is cleared by writing the Control Register (US\_CR) with the RSTSTA bit to 1. See [Figure 37-16](#) for an example of Manchester error detection during data phase.

**Figure 37-15. Preamble Pattern Mismatch**



**Figure 37-16. Manchester Error Flag**



When the start frame delimiter is a sync pattern (ONEBIT field to 0), both command and data delimiter are supported. If a valid sync is detected, the received character is written as RXCHR field in the US\_RHR register and the RXSYNH is updated. RXCHR is set to 1 when the received character is a command, and it is set to 0 if the received character is a data. This mechanism alleviates and simplifies the direct memory access as the character contains its own sync field in the same register.

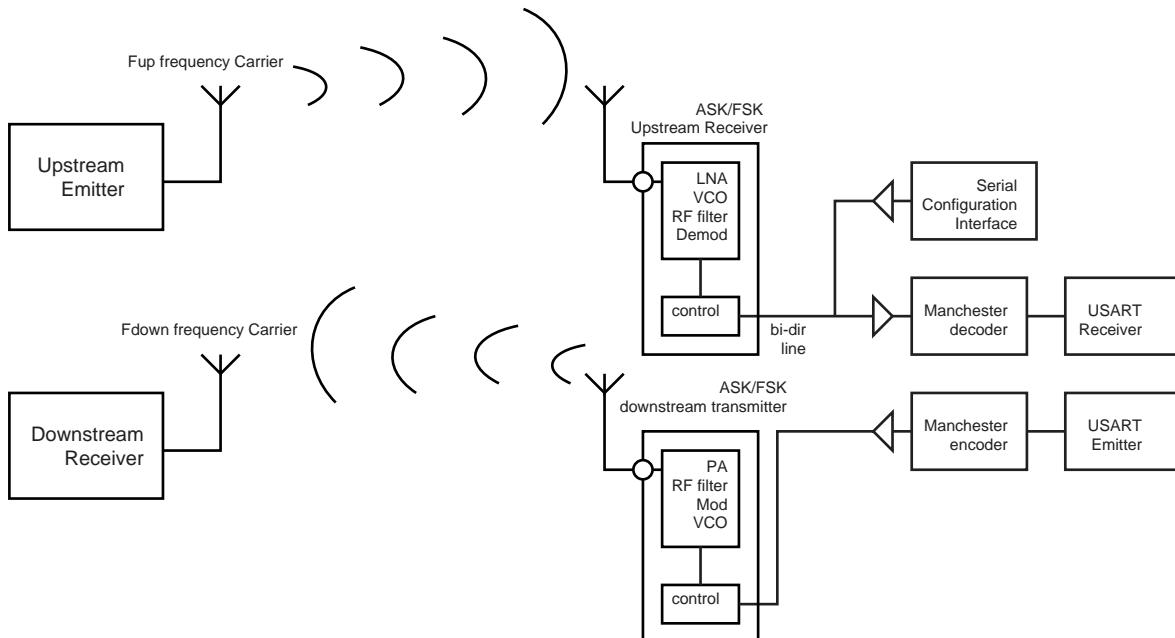
As the decoder is setup to be used in unipolar mode, the first bit of the frame has to be a zero-to-one transition.

#### 37.7.3.5 Radio Interface: Manchester Encoded USART Application

This section describes low data rate RF transmission systems and their integration with a Manchester encoded USART. These systems are based on transmitter and receiver ICs that support ASK and FSK modulation schemes.

The goal is to perform full duplex radio transmission of characters using two different frequency carriers. See the configuration in [Figure 37-17](#).

**Figure 37-17. Manchester Encoded Characters RF Transmission**

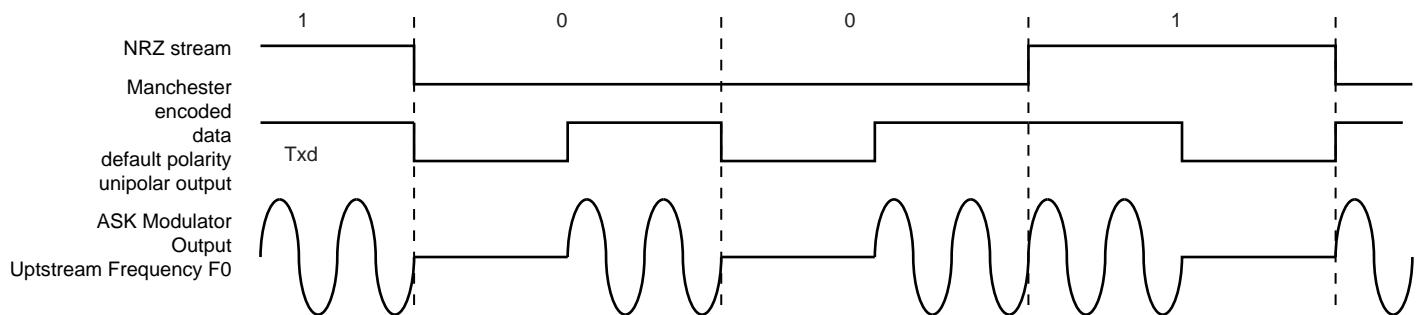


The USART module is configured as a Manchester encoder/decoder. Looking at the downstream communication channel, Manchester encoded characters are serially sent to the RF emitter. This may also include a user defined

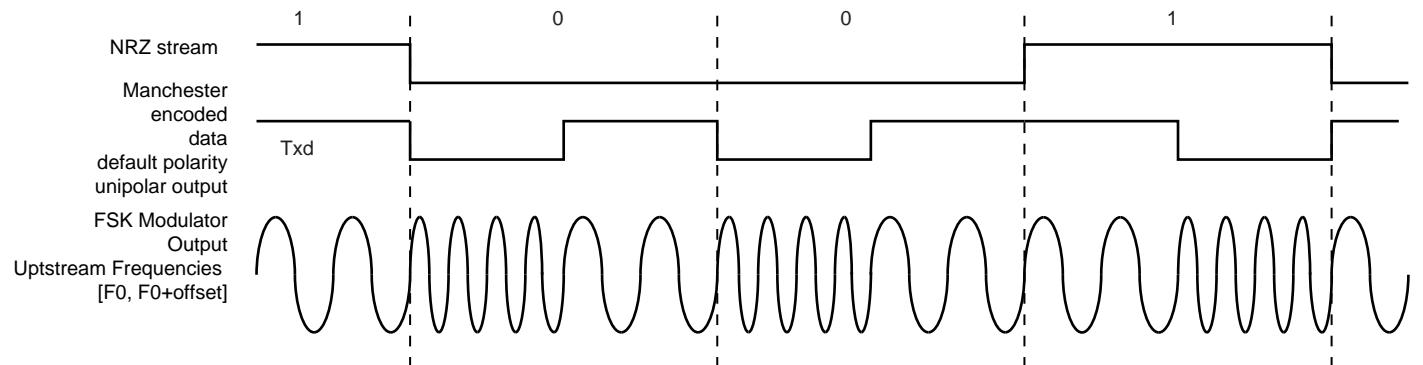
preamble and a start frame delimiter. Mostly, preamble is used in the RF receiver to distinguish between a valid data from a transmitter and signals due to noise. The Manchester stream is then modulated. See [Figure 37-18](#) for an example of ASK modulation scheme. When a logic one is sent to the ASK modulator, the power amplifier, referred to as PA, is enabled and transmits an RF signal at downstream frequency. When a logic zero is transmitted, the RF signal is turned off. If the FSK modulator is activated, two different frequencies are used to transmit data. When a logic 1 is sent, the modulator outputs an RF signal at frequency  $F_0$  and switches to  $F_1$  if the data sent is a 0. See [Figure 37-19](#).

From the receiver side, another carrier frequency is used. The RF receiver performs a bit check operation examining demodulated data stream. If a valid pattern is detected, the receiver switches to receiving mode. The demodulated stream is sent to the Manchester decoder. Because of bit checking inside RF IC, the data transferred to the microcontroller is reduced by a user-defined number of bits. The Manchester preamble length is to be defined in accordance with the RF IC configuration.

**Figure 37-18. ASK Modulator Output**



**Figure 37-19. FSK Modulator Output**



### 37.7.3.6 Synchronous Receiver

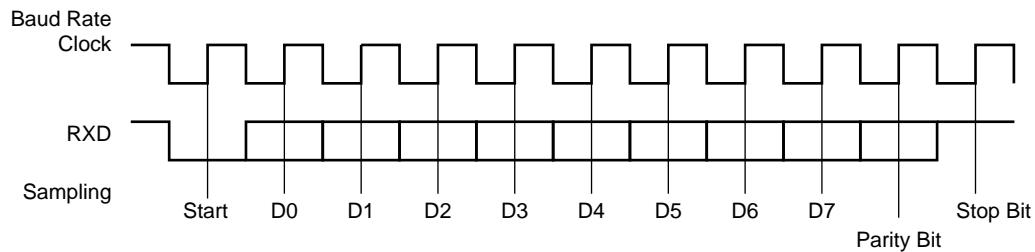
In synchronous mode ( $\text{SYNC} = 1$ ), the receiver samples the RXD signal on each rising edge of the Baud Rate Clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations provide a high speed transfer capability.

Configuration fields and bits are the same as in asynchronous mode.

[Figure 37-20](#) illustrates a character reception in synchronous mode.

**Figure 37-20. Synchronous Mode Character Reception**

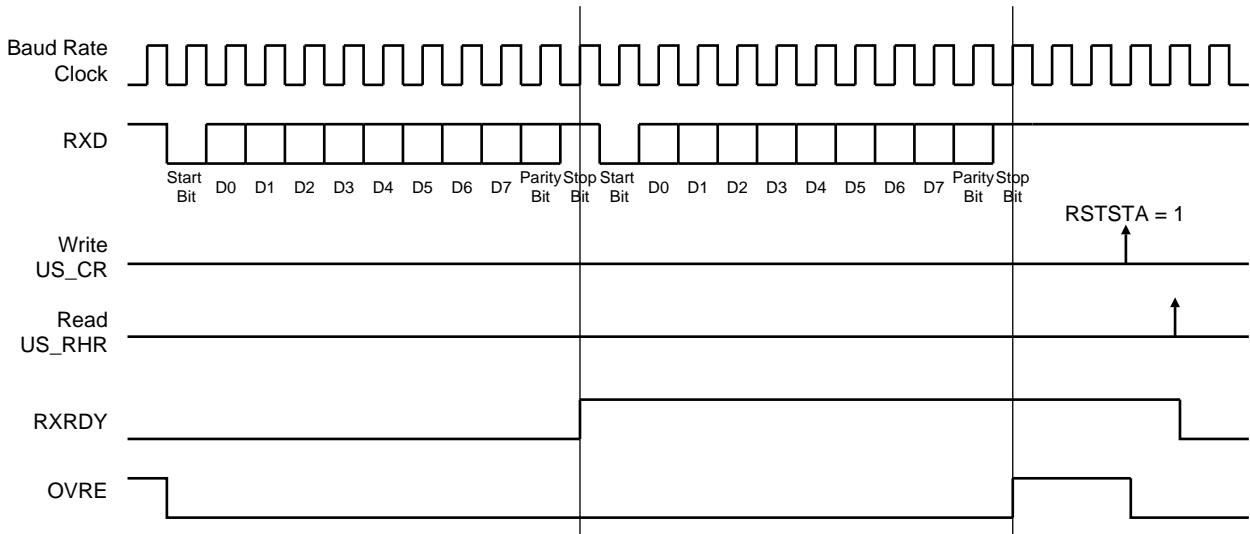
Example: 8-bit, Parity Enabled 1 Stop



### 37.7.3.7 Receiver Operations

When a character reception is completed, it is transferred to the Receive Holding Register (US\_RHR) and the RXRDY bit in the Status Register (US\_CSR) rises. If a character is completed while the RXRDY is set, the OVRE (Overrun Error) bit is set. The last character is transferred into US\_RHR and overwrites the previous one. The OVRE bit is cleared by writing the Control Register (US\_CR) with the RSTSTA (Reset Status) bit to 1.

**Figure 37-21. Receiver Status**



### 37.7.3.8 Parity

The USART supports five parity modes selected by programming the PAR field in the Mode Register (US\_MR). The PAR field also enables the Multidrop mode, see “[Multidrop Mode](#)” on page 841. Even and odd parity bit generation and error detection are supported.

If even parity is selected, the parity generator of the transmitter drives the parity bit to 0 if a number of 1s in the character data bit is even, and to 1 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If odd parity is selected, the parity generator of the transmitter drives the parity bit to 1 if a number of 1s in the character data bit is even, and to 0 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If the mark parity is used, the parity generator of the transmitter drives the parity bit to 1 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 0. If the space parity is used, the parity generator of the transmitter drives the parity bit to 0 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 1. If parity is disabled, the transmitter does not generate any parity bit and the receiver does not report any parity error.

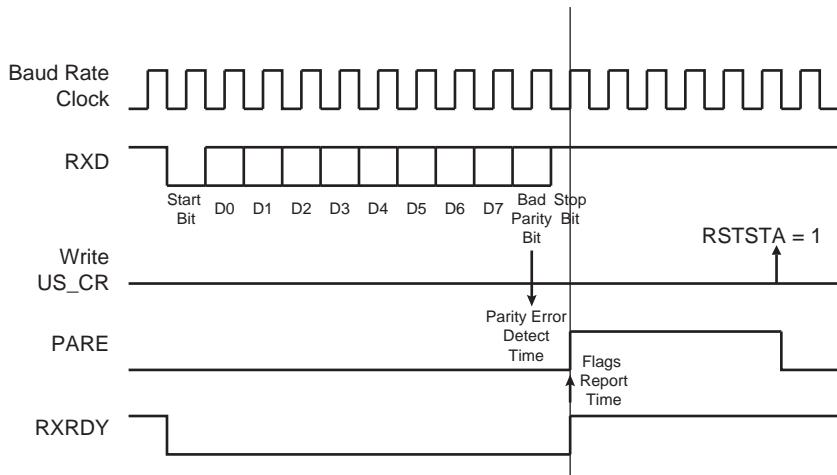
**Table 37-9** shows an example of the parity bit for the character 0x41 (character ASCII "A") depending on the configuration of the USART. Because there are two bits to 1, 1 bit is added when a parity is odd, or 0 is added when a parity is even.

**Table 37-9. Parity Bit Examples**

Character	Hexa	Binary	Parity Bit	Parity Mode
A	0x41	0100 0001	1	Odd
A	0x41	0100 0001	0	Even
A	0x41	0100 0001	1	Mark
A	0x41	0100 0001	0	Space
A	0x41	0100 0001	None	None

When the receiver detects a parity error, it sets the PARE (Parity Error) bit in the Channel Status Register (US\_CSR). The PARE bit can be cleared by writing the Control Register (US\_CR) with the RSTSTA bit to 1. [Figure 37-22](#) illustrates the parity bit status setting and clearing.

**Figure 37-22. Parity Error**



### 37.7.3.9 Multidrop Mode

If the PAR field in the Mode Register (US\_MR) is programmed to the value 0x6 or 0x07, the USART runs in Multidrop Mode. This mode differentiates the data characters and the address characters. Data is transmitted with the parity bit to 0 and addresses are transmitted with the parity bit to 1.

If the USART is configured in multidrop mode, the receiver sets the PARE parity error bit when the parity bit is high and the transmitter is able to send a character with the parity bit high when the Control Register is written with the SENDA bit to 1.

To handle parity error, the PARE bit is cleared when the Control Register is written with the bit RSTSTA to 1.

The transmitter sends an address byte (parity bit set) when SENDA is written to US\_CR. In this case, the next byte written to US\_THR is transmitted as an address. Any character written in US\_THR without having written the command SENDA is transmitted normally with the parity to 0.

### 37.7.3.10 Transmitter Timeguard

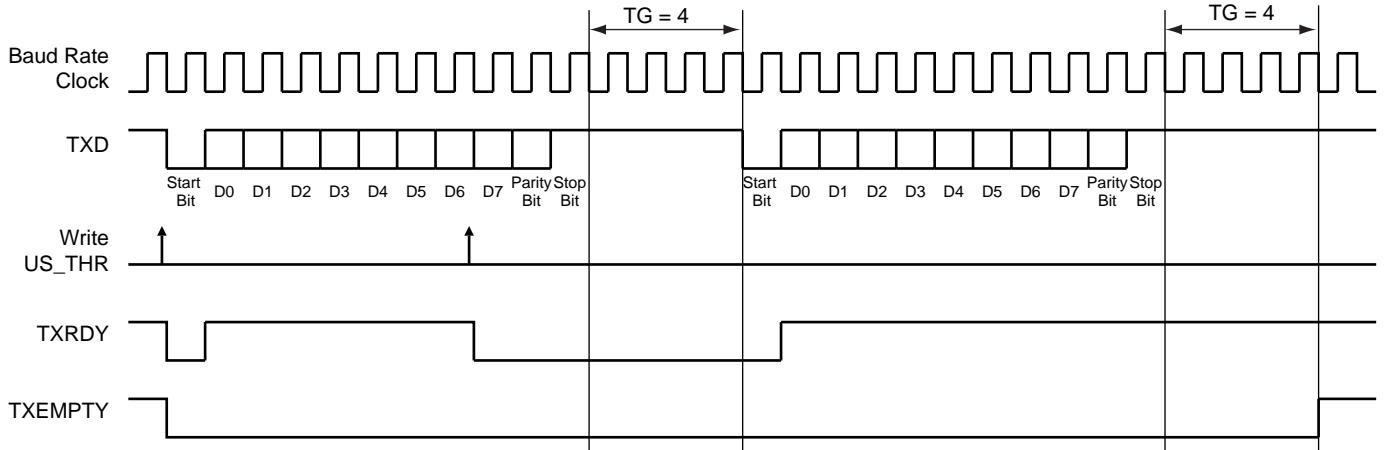
The timeguard feature enables the USART interface with slow remote devices.

The timeguard function enables the transmitter to insert an idle state on the TXD line between two characters. This idle state actually acts as a long stop bit.

The duration of the idle state is programmed in the TG field of the Transmitter Timeguard Register (US\_TTGR). When this field is programmed to zero no timeguard is generated. Otherwise, the transmitter holds a high level on TXD after each transmitted byte during the number of bit periods programmed in TG in addition to the number of stop bits.

As illustrated in [Figure 37-23](#), the behavior of TXRDY and TXEMPTY status bits is modified by the programming of a timeguard. TXRDY rises only when the start bit of the next character is sent, and thus remains to 0 during the timeguard transmission if a character has been written in US\_THR. TXEMPTY remains low until the timeguard transmission is completed as the timeguard is part of the current character being transmitted.

**Figure 37-23. Timeguard Operations**



[Table 37-10](#) indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the Baud Rate.

**Table 37-10. Maximum Timeguard Length Depending on Baud Rate**

Baud Rate	Bit time	Timeguard
Bit/sec	μs	ms
1 200	833	212.50
9 600	104	26.56
14 400	69.4	17.71
19 200	52.1	13.28
28 800	34.7	8.85
38 400	26	6.63
56 000	17.9	4.55
57 600	17.4	4.43
115 200	8.7	2.21

### 37.7.3.11 Receiver Time-out

The Receiver Time-out provides support in handling variable-length frames. This feature detects an idle condition on the RXD line. When a time-out is detected, the bit TIMEOUT in the Channel Status Register (US\_CSR) rises and can generate an interrupt, thus indicating to the driver an end of frame.

The time-out delay period (during which the receiver waits for a new character) is programmed in the TO field of the Receiver Time-out Register (US\_RTOR). If the TO field is programmed to 0, the Receiver Time-out is disabled and no time-out is detected. The TIMEOUT bit in US\_CSR remains to 0. Otherwise, the receiver loads a 16-bit counter with the

value programmed in TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, the TIMEOUT bit in the Status Register rises. Then, the user can either:

- Stop the counter clock until a new character is received. This is performed by writing the Control Register (US\_CR) with the STTTO (Start Time-out) bit to 1. In this case, the idle state on RXD before a new character is received will not provide a time-out. This prevents having to handle an interrupt before a character is received and allows waiting for the next idle state on RXD after a frame is received.
- Obtain an interrupt while no character is received. This is performed by writing US\_CR with the RETTO (Reload and Start Time-out) bit to 1. If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

If STTTO is performed, the counter clock is stopped until a first character is received. The idle state on RXD before the start of the frame does not provide a time-out. This prevents having to obtain a periodic interrupt and enables a wait of the end of frame when the idle state on RXD is detected.

If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

Figure 37-24 shows the block diagram of the Receiver Time-out feature.

**Figure 37-24. Receiver Time-out Block Diagram**

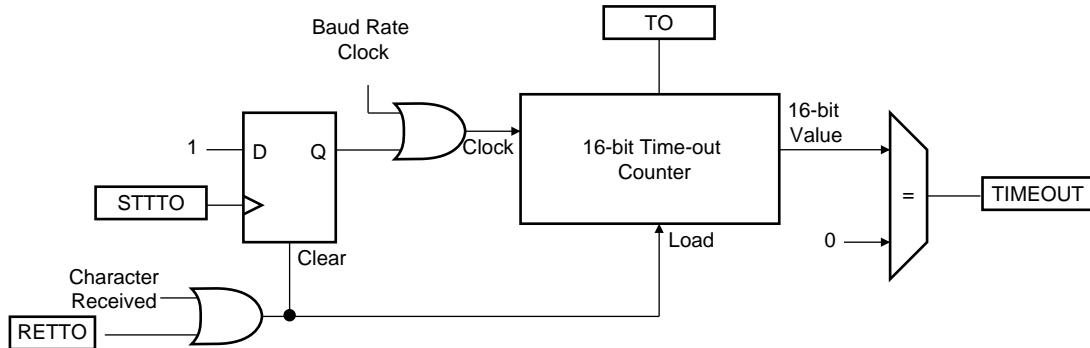


Table 37-11 gives the maximum time-out period for some standard baud rates.

**Table 37-11. Maximum Time-out Period**

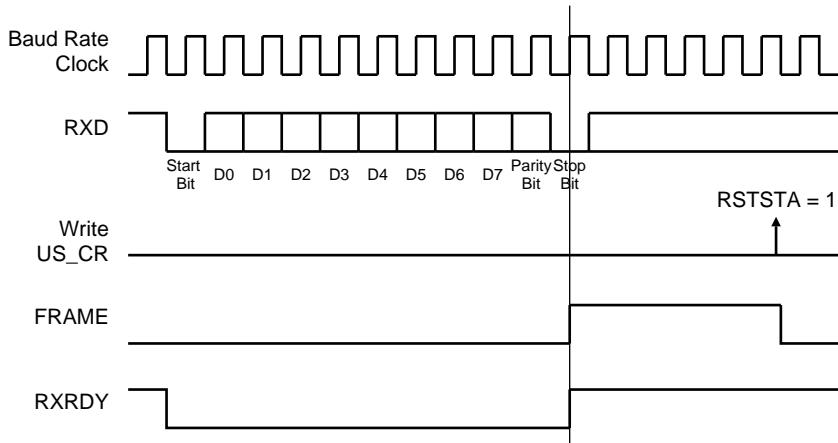
Baud Rate	Bit Time	Time-out
bit/sec	μs	ms
600	1 667	109 225
1 200	833	54 613
2 400	417	27 306
4 800	208	13 653
9 600	104	6 827
14400	69	4 551
19200	52	3 413
28800	35	2 276
38400	26	1 704
56000	18	1 170
57600	17	1 138
200000	5	328

### 37.7.3.12 Framing Error

The receiver is capable of detecting framing errors. A framing error happens when the stop bit of a received character is detected at level 0. This can occur if the receiver and the transmitter are fully desynchronized.

A framing error is reported on the FRAME bit of the Channel Status Register (US\_CSR). The FRAME bit is asserted in the middle of the stop bit as soon as the framing error is detected. It is cleared by writing the Control Register (US\_CR) with the RSTSTA bit to 1.

**Figure 37-25. Framing Error Status**



### 37.7.3.13 Transmit Break

The user can request the transmitter to generate a break condition on the TXD line. A break condition drives the TXD line low during at least one complete character. It appears the same as a 0x00 character sent with the parity and the stop bits to 0. However, the transmitter holds the TXD line at least during one character until the user requests the break condition to be removed.

A break is transmitted by writing the Control Register (US\_CR) with the STTBRK bit to 1. This can be performed at any time, either while the transmitter is empty (no character in either the Shift Register or in US\_THR) or when a character is being transmitted. If a break is requested while a character is being shifted out, the character is first completed before the TXD line is held low.

Once STTBRK command is requested further STTBRK commands are ignored until the end of the break is completed.

The break condition is removed by writing US\_CR with the STPBRK bit to 1. If the STPBRK is requested before the end of the minimum break duration (one character, including start, data, parity and stop bits), the transmitter ensures that the break condition completes.

The transmitter considers the break as though it is a character, i.e. the STTBRK and STPBRK commands are taken into account only if the TXRDY bit in US\_CSR is to 1 and the start of the break condition clears the TXRDY and TXEMPTY bits as if a character is processed.

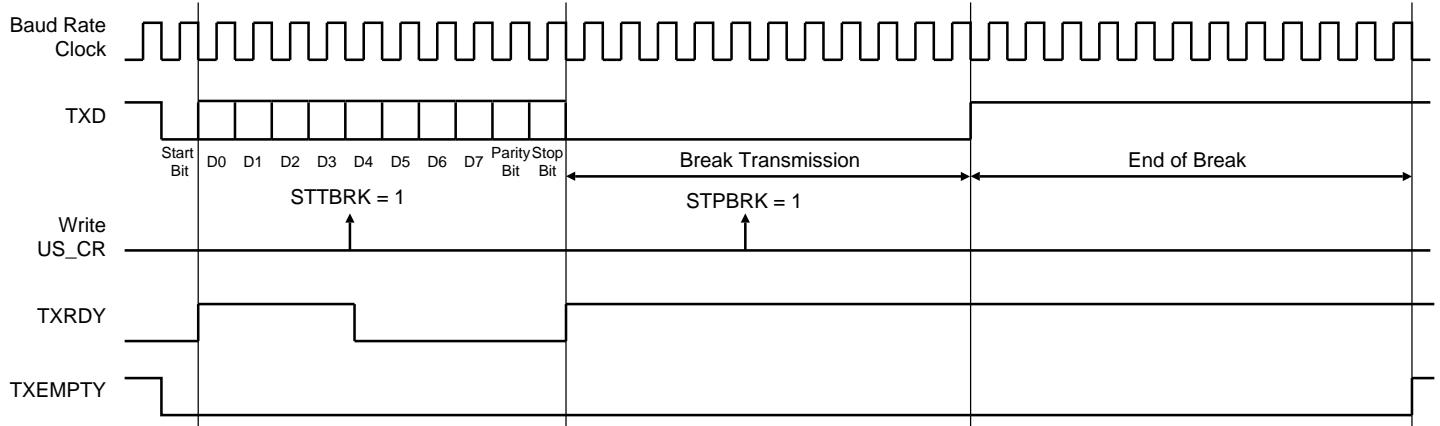
Writing US\_CR with both STTBRK and STPBRK bits to 1 can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into the Transmit Holding Register while a break is pending, but not started, is ignored.

After the break condition, the transmitter returns the TXD line to 1 for a minimum of 12 bit times. Thus, the transmitter ensures that the remote receiver detects correctly the end of break and the start of the next character. If the timeguard is programmed with a value higher than 12, the TXD line is held high for the timeguard period.

After holding the TXD line for this period, the transmitter resumes normal operations.

[Figure 37-26](#) illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

**Figure 37-26. Break Transmission**



### 37.7.3.14 Receive Break

The receiver detects a break condition when all data, parity and stop bits are low. This corresponds to detecting a framing error with data to 0x00, but FRAME remains low.

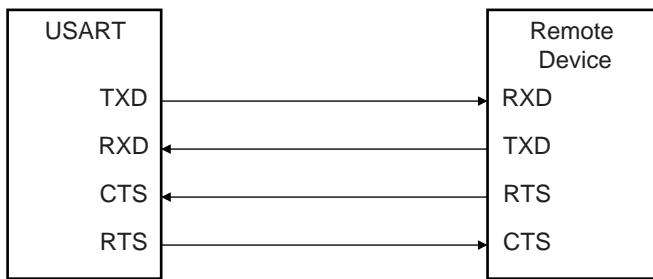
When the low stop bit is detected, the receiver asserts the RXBRK bit in US\_CSR. This bit may be cleared by writing the Control Register (US\_CR) with the bit RSTSTA to 1.

An end of receive break is detected by a high level for at least 2/16 of a bit period in asynchronous operating mode or one sample at high level in synchronous operating mode. The end of break detection also asserts the RXBRK bit.

### 37.7.3.15 Hardware Handshaking

The USART features a hardware handshaking out-of-band flow control. The RTS and CTS pins are used to connect with the remote device, as shown in [Figure 37-27](#).

**Figure 37-27. Connection with a Remote Device for Hardware Handshaking**



Setting the USART to operate with hardware handshaking is performed by writing the USART\_MODE field in the Mode Register (US\_MR) to the value 0x2.

The USART behavior when hardware handshaking is enabled is the same as the behavior in standard synchronous or asynchronous mode, except that the receiver drives the RTS pin as described below and the level on the CTS pin modifies the behavior of the transmitter as described below. Using this mode requires using the PDC or DMAC channel for reception. The transmitter can handle hardware handshaking in any case.

[Figure 37-28](#) shows how the receiver operates if hardware handshaking is enabled. The RTS pin is driven high if the receiver is disabled and if the status RXBUFF (Receive Buffer Full) coming from the PDC channel is high. Normally, the remote device does not start transmitting while its CTS pin (driven by RTS) is high. As soon as the Receiver is enabled, the RTS falls, indicating to the remote device that it can start transmitting. Defining a new buffer to the PDC clears the status bit RXBUFF and, as a result, asserts the pin RTS low.

**Figure 37-28. Receiver Behavior when Operating with Hardware Handshaking**

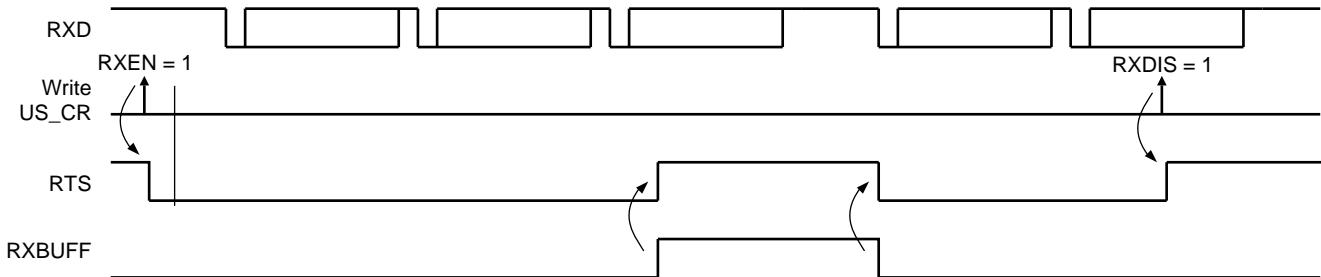
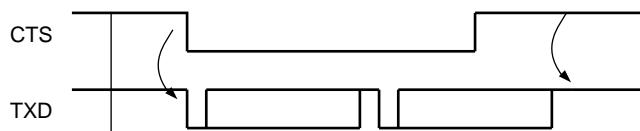


Figure 37-29 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processed, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

**Figure 37-29. Transmitter Behavior when Operating with Hardware Handshaking**



### 37.7.4 ISO7816 Mode

The USART features an ISO7816-compatible operating mode. This mode permits interfacing with smart cards and Security Access Modules (SAM) communicating through an ISO7816 link. Both T = 0 and T = 1 protocols defined by the ISO7816 specification are supported.

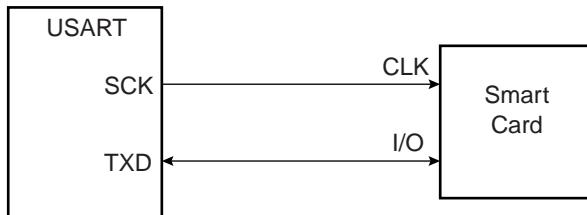
Setting the USART in ISO7816 mode is performed by writing the USART\_MODE field in the Mode Register (US\_MR) to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

#### 37.7.4.1 ISO7816 Mode Overview

The ISO7816 is a half duplex communication on only one bidirectional line. The baud rate is determined by a division of the clock provided to the remote device (see “[Baud Rate Generator](#)” on page 828).

The USART connects to a smart card as shown in Figure 37-30. The TXD line becomes bidirectional and the Baud Rate Generator feeds the ISO7816 clock on the SCK pin. As the TXD pin becomes bidirectional, its output remains driven by the output of the transmitter but only when the transmitter is active while its input is directed to the input of the receiver. The USART is considered as the master of the communication as it generates the clock.

**Figure 37-30. Connection of a Smart Card to the USART**



When operating in ISO7816, either in T = 0 or T = 1 modes, the character format is fixed. The configuration is 8 data bits, even parity and 1 or 2 stop bits, regardless of the values programmed in the CHRL, MODE9, PAR and CHMODE fields. MSBF can be used to transmit LSB or MSB first. Parity Bit (PAR) can be used to transmit in normal or inverse mode. Refer to “[USART Mode Register](#)” on page 865 and “[PAR: Parity Type](#)” on page 866.

The USART cannot operate concurrently in both receiver and transmitter modes as the communication is unidirectional at a time. It has to be configured according to the required mode by enabling or disabling either the receiver or the

transmitter as desired. Enabling both the receiver and the transmitter at the same time in ISO7816 mode may lead to unpredictable results.

The ISO7816 specification defines an inverse transmission format. Data bits of the character must be transmitted on the I/O line at their negative value.

#### 37.7.4.2 Protocol T = 0

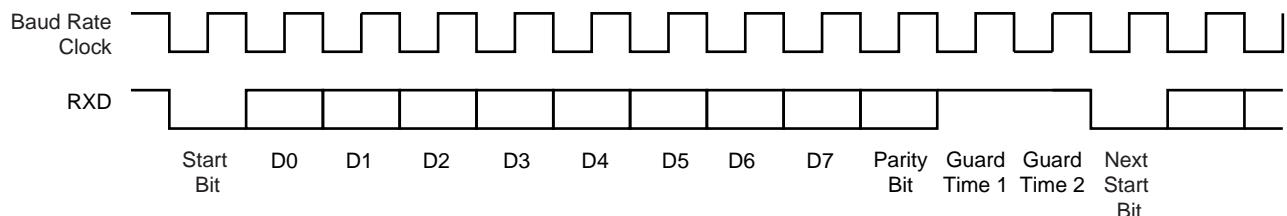
In T = 0 protocol, a character is made up of one start bit, eight data bits, one parity bit and one guard time, which lasts two bit times. The transmitter shifts out the bits and does not drive the I/O line during the guard time.

If no parity error is detected, the I/O line remains to 1 during the guard time and the transmitter can continue with the transmission of the next character, as shown in [Figure 37-31](#).

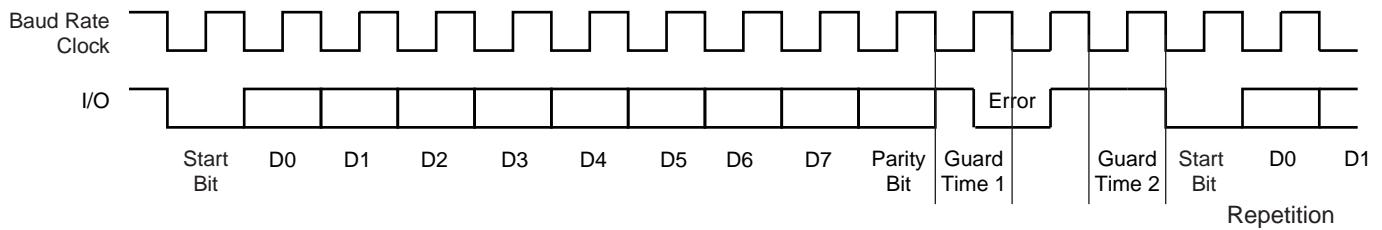
If a parity error is detected by the receiver, it drives the I/O line to 0 during the guard time, as shown in [Figure 37-32](#). This error bit is also named NACK, for Non Acknowledge. In this case, the character lasts 1 bit time more, as the guard time length is the same and is added to the error bit time which lasts 1 bit time.

When the USART is the receiver and it detects an error, it does not load the erroneous character in the Receive Holding Register (US\_RHR). It appropriately sets the PARE bit in the Status Register (US\_SR) so that the software can handle the error.

**Figure 37-31. T = 0 Protocol without Parity Error**



**Figure 37-32. T = 0 Protocol with Parity Error**



#### Receive Error Counter

The USART receiver also records the total number of errors. This can be read in the Number of Error (US\_NER) register.

The NB\_ERRORS field can record up to 255 errors. Reading US\_NER automatically clears the NB\_ERRORS field.

#### Receive NACK Inhibit

The USART can also be configured to inhibit an error. This can be achieved by setting the INACK bit in the Mode Register (US\_MR). If INACK is to 1, no error signal is driven on the I/O line even if a parity bit is detected.

Moreover, if INACK is set, the erroneous received character is stored in the Receive Holding Register, as if no error occurred and the RXRDY bit does rise.

#### Transmit Character Repetition

When the USART is transmitting a character and gets a NACK, it can automatically repeat the character before moving on to the next one. Repetition is enabled by writing the MAX\_ITERATION field in the Mode Register (US\_MR) at a value higher than 0. Each character can be transmitted up to eight times; the first transmission plus seven repetitions.

If MAX\_ITERATION does not equal zero, the USART repeats the character as many times as the value loaded in MAX\_ITERATION.

When the USART repetition number reaches MAX\_ITERATION, the ITERATION bit is set in the Channel Status Register (US\_CSR). If the repetition of the character is acknowledged by the receiver, the repetitions are stopped and the iteration counter is cleared.

The ITERATION bit in US\_CSR can be cleared by writing the Control Register with the RSIT bit to 1.

#### Disable Successive Receive NACK

The receiver can limit the number of successive NACKs sent back to the remote transmitter. This is programmed by setting the bit DSNACK in the Mode Register (US\_MR). The maximum number of NACK transmitted is programmed in the MAX\_ITERATION field. As soon as MAX\_ITERATION is reached, the character is considered as correct, an acknowledge is sent on the line and the ITERATION bit in the Channel Status Register is set.

#### 37.7.4.3 Protocol T = 1

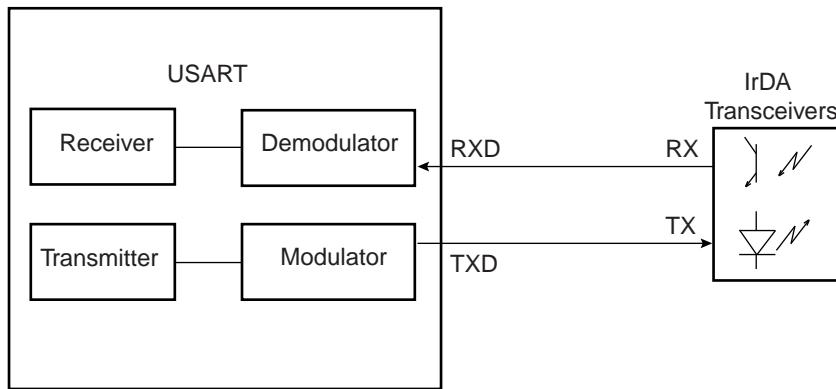
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the Channel Status Register (US\_CSR).

#### 37.7.5 IrDA Mode

The USART features an IrDA mode supplying half-duplex point-to-point wireless communication. It embeds the modulator and demodulator which allows a glueless connection to the infrared transceivers, as shown in [Figure 37-33](#). The modulator and demodulator are compliant with the IrDA specification version 1.1 and support data transfer speeds ranging from 2.4 Kb/s to 115.2 Kb/s.

The USART IrDA mode is enabled by setting the USART\_MODE field in the Mode Register (US\_MR) to the value 0x8. The IrDA Filter Register (US\_IF) allows configuring the demodulator filter. The USART transmitter and receiver operate in a normal asynchronous mode and all parameters are accessible. Note that the modulator and the demodulator are activated.

**Figure 37-33. Connection to IrDA Transceivers**



The receiver and the transmitter must be enabled or disabled according to the direction of the transmission to be managed.

To receive IrDA signals, the following needs to be done:

- Disable TX and Enable RX
- Configure the TXD pin as PIO and set it as an output to 0 (to avoid LED emission). Disable the internal pull-up (better for power consumption).
- Receive data

### 37.7.5.1 IrDA Modulation

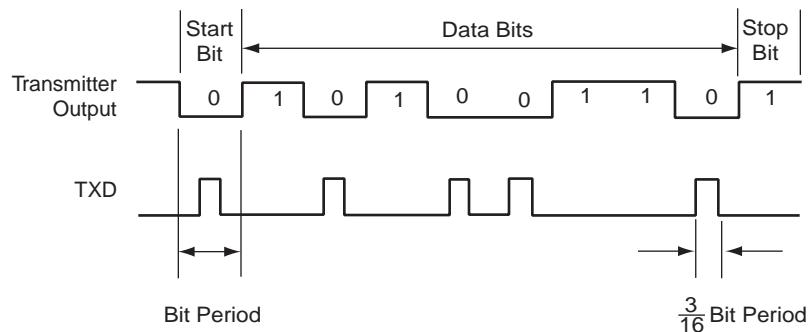
For baud rates up to and including 115.2 Kbits/sec, the RZI modulation scheme is used. "0" is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in [Table 37-12](#).

**Table 37-12. IrDA Pulse Duration**

Baud Rate	Pulse Duration (3/16)
2.4 Kb/s	78.13 µs
9.6 Kb/s	19.53 µs
19.2 Kb/s	9.77 µs
38.4 Kb/s	4.88 µs
57.6 Kb/s	3.26 µs
115.2 Kb/s	1.63 µs

[Figure 37-34](#) shows an example of character transmission.

**Figure 37-34. IrDA Modulation**



### 37.7.5.2 IrDA Baud Rate

[Table 37-13](#) gives some examples of CD values, baud rate error and pulse duration. Note that the requirement on the maximum acceptable error of 1.87% must be met.

**Table 37-13. IrDA Baud Rate Error**

Peripheral Clock	Baud Rate	CD	Baud Rate Error	Pulse Time
3 686 400	115 200	2	0.00%	1.63
20 000 000	115 200	11	1.38%	1.63
32 768 000	115 200	18	1.25%	1.63
40 000 000	115 200	22	1.38%	1.63
3 686 400	57 600	4	0.00%	3.26
20 000 000	57 600	22	1.38%	3.26
32 768 000	57 600	36	1.25%	3.26
40 000 000	57 600	43	0.93%	3.26
3 686 400	38 400	6	0.00%	4.88
20 000 000	38 400	33	1.38%	4.88
32 768 000	38 400	53	0.63%	4.88
40 000 000	38 400	65	0.16%	4.88
3 686 400	19 200	12	0.00%	9.77
20 000 000	19 200	65	0.16%	9.77
32 768 000	19 200	107	0.31%	9.77

**Table 37-13. IrDA Baud Rate Error**

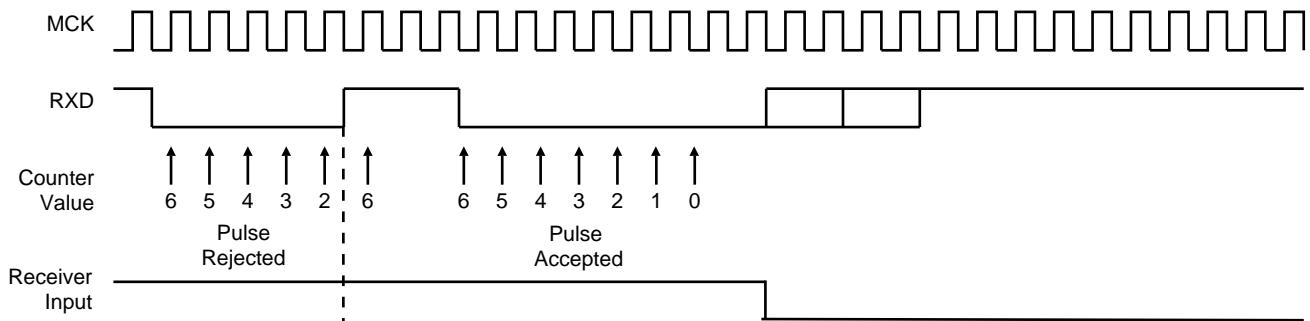
Peripheral Clock	Baud Rate	CD	Baud Rate Error	Pulse Time
40 000 000	19 200	130	0.16%	9.77
3 686 400	9 600	24	0.00%	19.53
20 000 000	9 600	130	0.16%	19.53
32 768 000	9 600	213	0.16%	19.53
40 000 000	9 600	260	0.16%	19.53
3 686 400	2 400	96	0.00%	78.13
20 000 000	2 400	521	0.03%	78.13
32 768 000	2 400	853	0.04%	78.13

### 37.7.5.3 IrDA Demodulator

The demodulator is based on the IrDA Receive filter comprised of an 8-bit down counter which is loaded with the value programmed in US\_IF. When a falling edge is detected on the RXD pin, the Filter Counter starts counting down at the Master Clock (MCK) speed. If a rising edge is detected on the RXD pin, the counter stops and is reloaded with US\_IF. If no rising edge is detected when the counter reaches 0, the input of the receiver is driven low during one bit time.

Figure 37-35 illustrates the operations of the IrDA demodulator.

**Figure 37-35. IrDA Demodulator Operations**



The programmed value in the US\_IF register must always meet the following criteria:

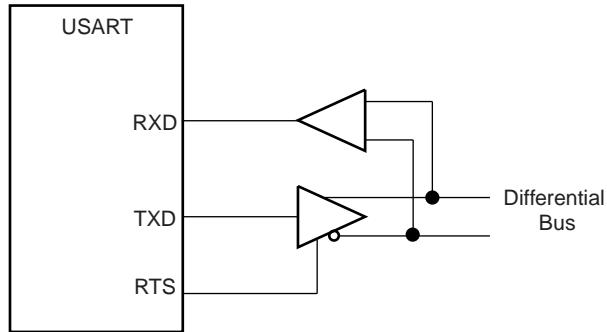
$$T_{MCK} * (IRDA\_FILTER + 3) < 1.41 \text{ us}$$

As the IrDA mode uses the same logic as the ISO7816, note that the FI\_DI\_RATIO field in US\_FIDI must be set to a value higher than 0 in order to assure IrDA communications operate correctly.

### 37.7.6 RS485 Mode

The USART features the RS485 mode to enable line driver control. While operating in RS485 mode, the USART behaves as though in asynchronous or synchronous mode and configuration of all the parameters is possible. The difference is that the RTS pin is driven high when the transmitter is operating. The behavior of the RTS pin is controlled by the TXEMPTY bit. A typical connection of the USART to a RS485 bus is shown in Figure 37-36.

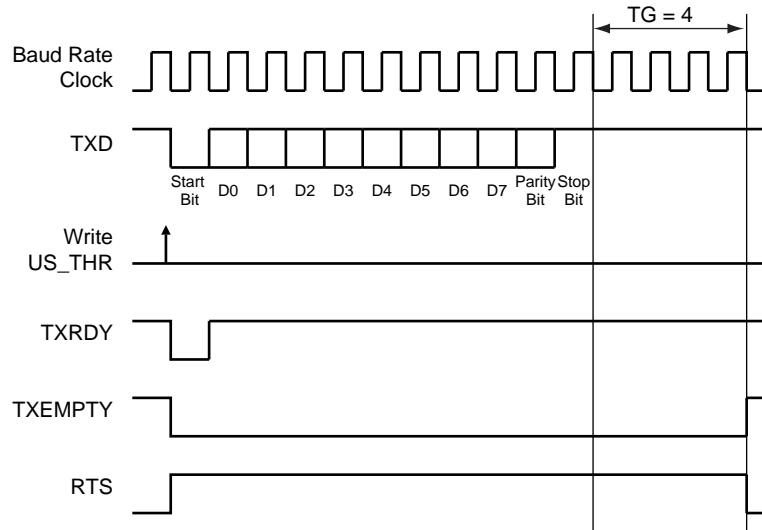
**Figure 37-36. Typical Connection to a RS485 Bus**



The USART is set in RS485 mode by programming the USART\_MODE field in the Mode Register (US\_MR) to the value 0x1.

The RTS pin is at a level inverse to the TXEMPTY bit. Significantly, the RTS pin remains high when a timeguard is programmed so that the line can remain driven after the last character completion. [Figure 37-37](#) gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

**Figure 37-37. Example of RTS Drive with Timeguard**



### 37.7.7 Modem Mode

The USART features modem mode, which enables control of the signals: DTR (Data Terminal Ready), DSR (Data Set Ready), RTS (Request to Send), CTS (Clear to Send), DCD (Data Carrier Detect) and RI (Ring Indicator). While operating in modem mode, the USART behaves as a DTE (Data Terminal Equipment) as it drives DTR and RTS and can detect level change on DSR, DCD, CTS and RI.

Setting the USART in modem mode is performed by writing the USART\_MODE field in the Mode Register (US\_MR) to the value 0x3. While operating in modem mode the USART behaves as though in asynchronous mode and all the parameter configurations are available.

[Table 37-14](#) gives the correspondence of the USART signals with modem connection standards.

**Table 37-14. Circuit References**

USART Pin	V24	CCITT	Direction
TXD	2	103	From terminal to modem
RTS	4	105	From terminal to modem
DTR	20	108.2	From terminal to modem
RXD	3	104	From modem to terminal
CTS	5	106	From terminal to modem
DSR	6	107	From terminal to modem
DCD	8	109	From terminal to modem
RI	22	125	From terminal to modem

The control of the DTR output pin is performed by writing the Control Register (US\_CR) with the DTRDIS and DTREN bits respectively to 1. The disable command forces the corresponding pin to its inactive level, i.e. high. The enable command forces the corresponding pin to its active level, i.e. low. RTS output pin is automatically controlled in this mode.

The level changes are detected on the RI, DSR, DCD and CTS pins. If an input change is detected, the RIIC, DSRIC, DCDIC and CTSIC bits in the Channel Status Register (US\_CSR) are set respectively and can trigger an interrupt. The status is automatically cleared when US\_CSR is read. Furthermore, the CTS automatically disables the transmitter when it is detected at its inactive state. If a character is being transmitted when the CTS rises, the character transmission is completed before the transmitter is actually disabled.

### 37.7.8 SPI Mode

The Serial Peripheral Interface (SPI) Mode is a synchronous serial data link that provides communication with external devices in Master or Slave Mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master” which controls the data flow, while the other devices act as “slaves” which have data shifted into and out by the master. Different CPUs can take turns being masters and one master may simultaneously shift data into multiple slaves. (Multiple Master Protocol is the opposite of Single Master Protocol, where one CPU is always the master while all of the others are always slaves.) However, only one slave may drive its output to write data back to the master at any given time.

A slave device is selected when its NSS signal is asserted by the master. The USART in SPI Master mode can address only one SPI Slave because it can generate only one NSS signal.

The SPI system consists of two data lines and two control lines:

- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input of the slave.
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master.
- Serial Clock (SCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates. The SCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows the master to select or deselect the slave.

#### 37.7.8.1 Modes of Operation

The USART can operate in SPI Master Mode or in SPI Slave Mode.

Operation in SPI Master Mode is programmed by writing to 0xE the USART\_MODE field in the Mode Register. In this case the SPI lines must be connected as described below:

- The MOSI line is driven by the output pin TXD
- The MISO line drives the input pin RXD
- The SCK line is driven by the output pin SCK
- The NSS line is driven by the output pin RTS

Operation in SPI Slave Mode is programmed by writing to 0xF the USART\_MODE field in the Mode Register. In this case the SPI lines must be connected as described below:

- The MOSI line drives the input pin RXD
- The MISO line is driven by the output pin TXD
- The SCK line drives the input pin SCK
- The NSS line drives the input pin CTS

In order to avoid unpredicted behavior, any change of the SPI Mode must be followed by a software reset of the transmitter and of the receiver (except the initial configuration after a hardware reset). (See [Section 37.7.8.4 "Receiver and Transmitter Control"](#)).

#### 37.7.8.2 Baud Rate

In SPI Mode, the baudrate generator operates in the same way as in USART synchronous mode: See [“Baud Rate in Synchronous Mode or SPI Mode” on page 830](#). However, there are some restrictions:

In SPI Master Mode:

- The external clock SCK must not be selected (USCLKS = 0x3), and the bit CLKO must be set to “1” in the Mode Register (US\_MR), in order to generate correctly the serial clock on the SCK pin.
- To obtain correct behavior of the receiver and the transmitter, the value programmed in CD must be superior or equal to 6.
- If the internal clock divided (MCK/DIV) is selected, the value programmed in CD must be even to ensure a 50:50 mark/space ratio on the SCK pin, this value can be odd if the internal clock is selected (MCK).

### In SPI Slave Mode:

- The external clock (SCK) selection is forced regardless of the value of the USCLKS field in the Mode Register (US\_MR). Likewise, the value written in US\_BRGR has no effect, because the clock is provided directly by the signal on the USART SCK pin.
- To obtain correct behavior of the receiver and the transmitter, the external clock (SCK) frequency must be at least 6 times lower than the system clock.

### 37.7.8.3 Data Transfer

Up to 9 data bits are successively shifted out on the TXD pin at each rising or falling edge (depending of CPOL and CPHA) of the programmed serial clock. There is no Start bit, no Parity bit and no Stop bit.

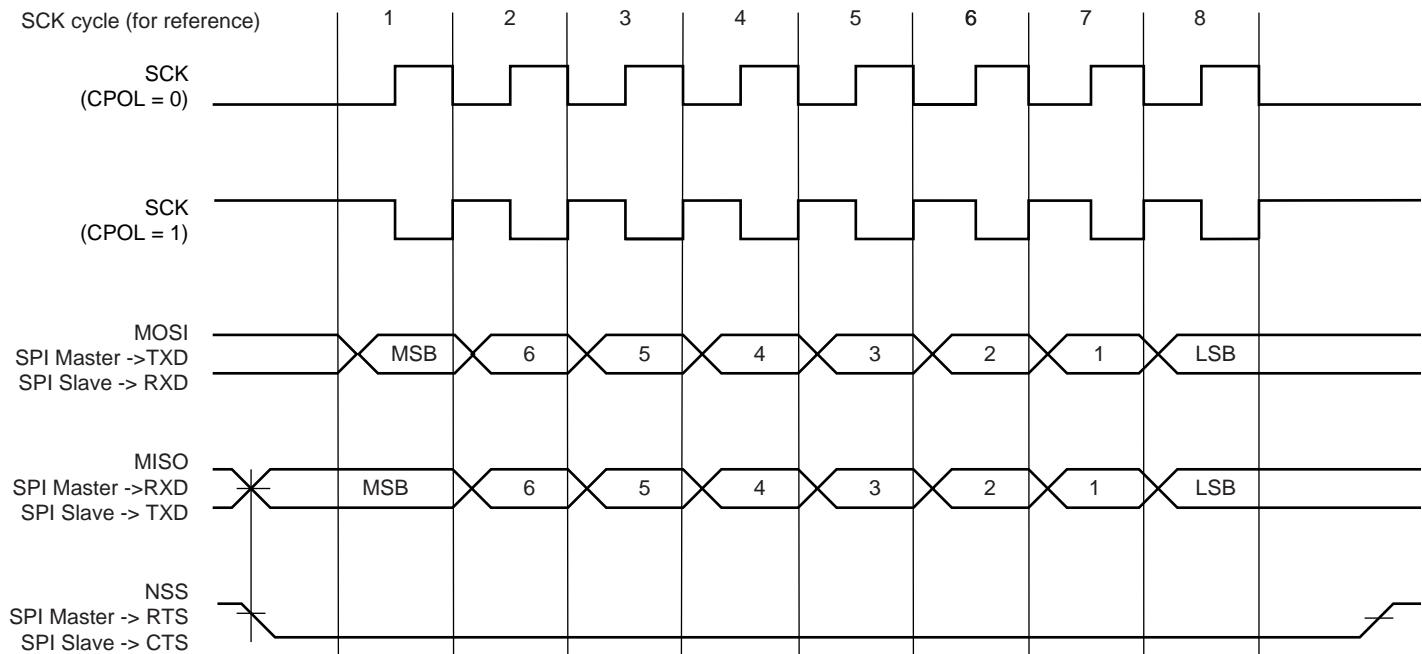
The number of data bits is selected by the CHRL field and the MODE 9 bit in the Mode Register (US\_MR). The 9 bits are selected by setting the MODE 9 bit regardless of the CHRL field. The MSB data bit is always sent first in SPI Mode (Master or Slave).

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the Mode Register. The clock phase is programmed with the CPHA bit. These two parameters determine the edges of the clock signal upon which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

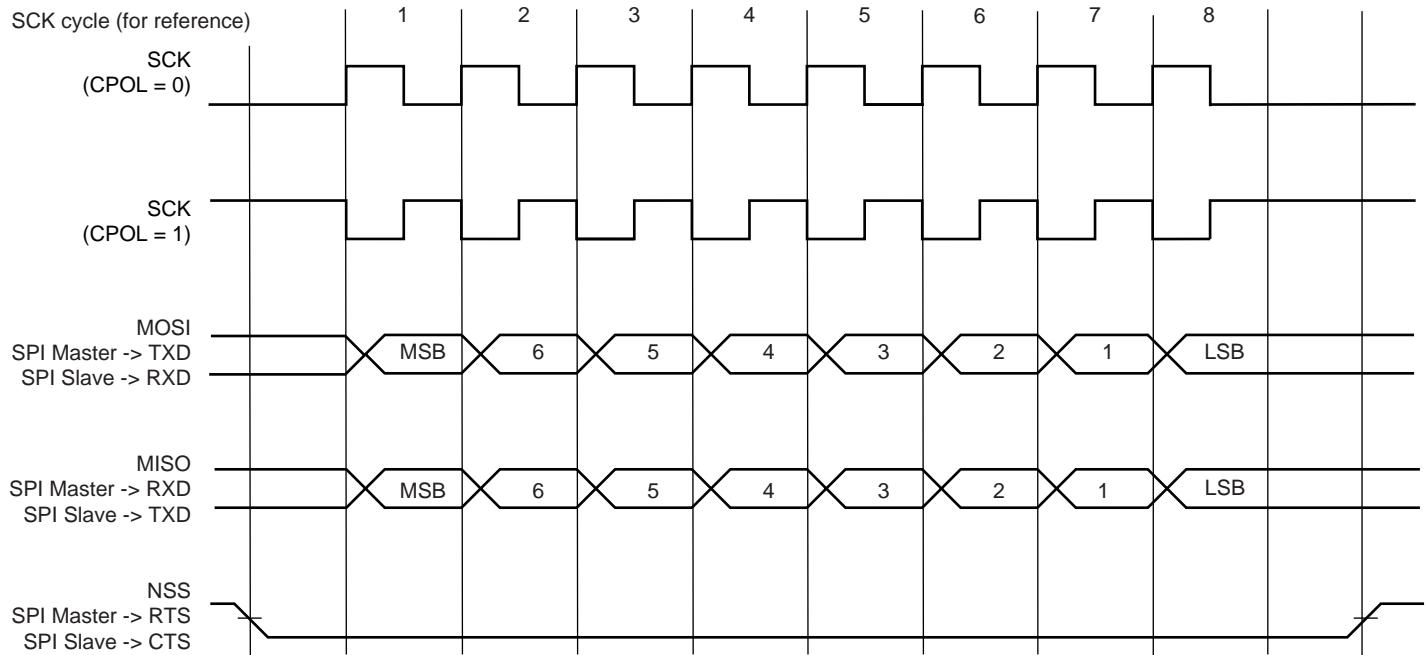
**Table 37-15. SPI Bus Protocol Mode**

SPI Bus Protocol Mode	CPOL	CPHA
0	0	1
1	0	0
2	1	1
3	1	0

**Figure 37-38. SPI Transfer Format (CPHA=1, 8 bits per transfer)**



**Figure 37-39. SPI Transfer Format (CPHA=0, 8 bits per transfer)**



#### 37.7.8.4 Receiver and Transmitter Control

See “[Receiver and Transmitter Control](#)” on page 831.

#### 37.7.8.5 Character Transmission

The characters are sent by writing in the Transmit Holding Register (US\_THR). An additional condition for transmitting a character can be added when the USART is configured in SPI master mode. In the USART\_MR register, the value configured on INACK field can prevent any character transmission (even if US\_THR has been written) while the receiver side is not ready (character not read). When WRDBT equals 0, the character is transmitted whatever the receiver status. If WRDBT is set to 1, the transmitter waits for the receiver holding register to be read before transmitting the character (RXRDY flag cleared), thus preventing any overflow (character loss) on the receiver side.

The transmitter reports two status bits in the Channel Status Register (US\_CSR): TXRDY (Transmitter Ready), which indicates that US\_THR is empty and TXEMPTY, which indicates that all the characters written in US\_THR have been processed. When the current character processing is completed, the last character written in US\_THR is transferred into the Shift Register of the transmitter and US\_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in US\_THR while TXRDY is low has no effect and the written character is lost.

If the USART is in SPI Slave Mode and if a character must be sent while the Transmit Holding Register (US\_THR) is empty, the UNRE (Underrun Error) bit is set. The TXD transmission line stays at high level during all this time. The UNRE bit is cleared by writing the Control Register (US\_CR) with the RSTSTA (Reset Status) bit to 1.

In SPI Master Mode, the slave select line (NSS) is asserted at low level 1 Tbit (Time bit) before the transmission of the MSB bit and released at high level 1 Tbit after the transmission of the LSB bit. So, the slave select line (NSS) is always released between each character transmission and a minimum delay of 3 Tbits always inserted. However, in order to address slave devices supporting the CSAAT mode (Chip Select Active After Transfer), the slave select line (NSS) can be forced at low level by writing the Control Register (US\_CR) with the RTSEN bit to 1. The slave select line (NSS) can be released at high level only by writing the Control Register (US\_CR) with the RTSDIS bit to 1 (for example, when all data have been transferred to the slave device).

In SPI Slave Mode, the transmitter does not require a falling edge of the slave select line (NSS) to initiate a character transmission but only a low level. However, this low level must be present on the slave select line (NSS) at least 1 Tbit before the first serial clock cycle corresponding to the MSB bit.

#### 37.7.8.6 Character Reception

When a character reception is completed, it is transferred to the Receive Holding Register (US\_RHR) and the RXRDY bit in the Status Register (US\_CSR) rises. If a character is completed while RXRDY is set, the OVRE (Overrun Error) bit is set. The last character is transferred into US\_RHR and overwrites the previous one. The OVRE bit is cleared by writing the Control Register (US\_CR) with the RSTSTA (Reset Status) bit to 1.

To ensure correct behavior of the receiver in SPI Slave Mode, the master device sending the frame must ensure a minimum delay of 1 Tbit between each character transmission. The receiver does not require a falling edge of the slave select line (NSS) to initiate a character reception but only a low level. However, this low level must be present on the slave select line (NSS) at least 1 Tbit before the first serial clock cycle corresponding to the MSB bit.

#### 37.7.8.7 Receiver Timeout

Because the receiver baudrate clock is active only during data transfers in SPI Mode, a receiver timeout is impossible in this mode, whatever the Time-out value is (field TO) in the Time-out Register (US\_RTOR).

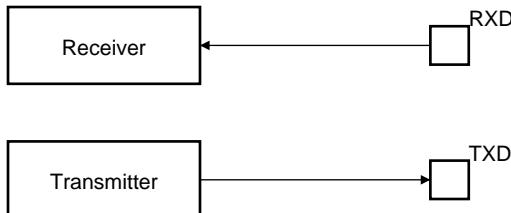
### 37.7.9 Test Modes

The USART can be programmed to operate in three different test modes. The internal loopback capability allows on-board diagnostics. In the loopback mode the USART interface pins are disconnected or not and reconfigured for loopback internally or externally.

#### 37.7.9.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

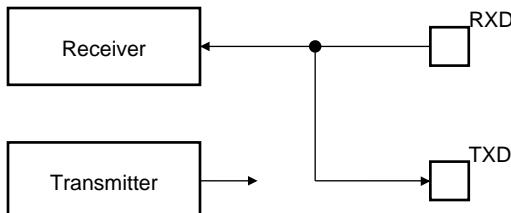
**Figure 37-40. Normal Mode Configuration**



#### 37.7.9.2 Automatic Echo Mode

Automatic echo mode allows bit-by-bit retransmission. When a bit is received on the RXD pin, it is sent to the TXD pin, as shown in [Figure 37-41](#). Programming the transmitter has no effect on the TXD pin. The RXD pin is still connected to the receiver input, thus the receiver remains active.

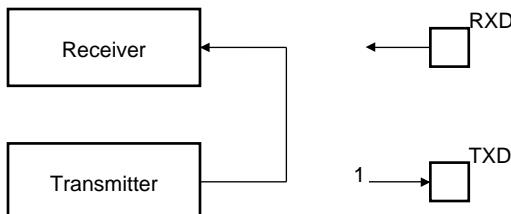
**Figure 37-41. Automatic Echo Mode Configuration**



#### 37.7.9.3 Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in [Figure 37-42](#). The TXD and RXD pins are not used. The RXD pin has no effect on the receiver and the TXD pin is continuously driven high, as in idle state.

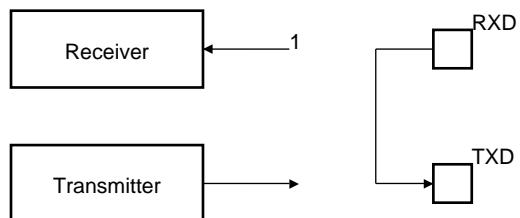
**Figure 37-42. Local Loopback Mode Configuration**



#### 37.7.9.4 Remote Loopback Mode

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in [Figure 37-43](#). The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

**Figure 37-43. Remote Loopback Mode Configuration**



### 37.7.10 Write Protection Registers

To prevent any single software error that may corrupt USART behavior, certain address spaces can be write-protected by setting the WPEN bit in the USART Write Protect Mode Register (US\_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the USART Write Protect Status Register (US\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is automatically reset by reading the USART Write Protect Mode Register (US\_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- “USART Mode Register”
- “USART Baud Rate Generator Register”
- “USART Receiver Time-out Register”
- “USART Transmitter Timeguard Register”
- “USART FI DI RATIO Register”
- “USART IrDA FILTER Register”
- “USART Manchester Configuration Register”

## 37.8 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface

Table 37-16. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Control Register	US_CR	Write-only	
0x0004	Mode Register	US_MR	Read-write	–
0x0008	Interrupt Enable Register	US_IER	Write-only	–
0x000C	Interrupt Disable Register	US_IDR	Write-only	–
0x0010	Interrupt Mask Register	US_IMR	Read-only	0x0
0x0014	Channel Status Register	US_CSR	Read-only	–
0x0018	Receiver Holding Register	US_RHR	Read-only	0x0
0x001C	Transmitter Holding Register	US_THR	Write-only	–
0x0020	Baud Rate Generator Register	US_BRGR	Read-write	0x0
0x0024	Receiver Time-out Register	US_RTOR	Read-write	0x0
0x0028	Transmitter Timeguard Register	US_TTGR	Read-write	0x0
0x2C - 0x3C	Reserved	–	–	–
0x0040	FI DI Ratio Register	US_FIDI	Read-write	0x174
0x0044	Number of Errors Register	US_NER	Read-only	–
0x0048	Reserved	–	–	–
0x004C	IrDA Filter Register	US_IF	Read-write	0x0
0x0050	Manchester Encoder Decoder Register	US_MAN	Read-write	0xB0011004
0x0054 - 0x005C	Reserved	–	–	–
0x0060-0x00E0	Reserved	–	–	–
0x00E4	Write Protect Mode Register	US_WPMR	Read-write	0x0
0x00E8	Write Protect Status Register	US_WPSR	Read-only	0x0
0x00EC - 0x00FC	Reserved	–	–	–
0x100 - 0x128	Reserved for PDC Registers	–	–	–

### 37.8.1 USART Control Register

**Name:** US\_CR

**Address:** 0x400A0000 (0), 0x400A4000 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	RTSDIS	RTSEN	DTRDIS	DTREN
15	14	13	12	11	10	9	8
RETTTO	RSTNACK	RSTIT	SENDA	STTTO	STPBRK	STTBRK	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	—	—

For SPI control, see “[USART Control Register \(SPI\\_MODE\)](#)” on page 863.

- **RSTRX: Reset Receiver**

0: No effect.

1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.

1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.

1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.

1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits PARE, FRAME, OVRE, MANERR and RXBRK in US\_CSR.

- **STTBRK: Start Break**

0: No effect.

1: Starts transmission of a break after the characters present in US\_THR and the Transmit Shift Register have been transmitted.  
No effect if a break is already being transmitted.

- **STPBRK: Stop Break**

0: No effect.

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods. No effect if no break is being transmitted.

- **STTTO: Start Time-out**

0: No effect.

1: Starts waiting for a character before clocking the time-out counter. Resets the status bit TIMEOUT in US\_CSR.

- **SENDA: Send Address**

0: No effect.

1: In Multidrop Mode only, the next character written to the US\_THR is sent with the address bit set.

- **RSTIT: Reset Iterations**

0: No effect.

1: Resets ITERATION in US\_CSR. No effect if the ISO7816 is not enabled.

- **RSTNACK: Reset Non Acknowledge**

0: No effect

1: Resets NACK in US\_CSR.

- **RETTTO: Rearm Time-out**

0: No effect

1: Restart Time-out

- **DTREN: Data Terminal Ready Enable**

0: No effect.

1: Drives the pin DTR to 0.

- **DTRDIS: Data Terminal Ready Disable**

0: No effect.

1: Drives the pin DTR to 1.

- **RTSEN: Request to Send Enable**

0: No effect.

1: Drives the pin RTS to 0.

- **RTSDIS: Request to Send Disable**

0: No effect.

1: Drives the pin RTS to 1.

### 37.8.2 USART Control Register (SPI\_MODE)

**Name:** US\_CR (SPI\_MODE)  
**Address:** 0x400A0000 (0), 0x400A4000 (1)  
**Access:** Write-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	RCS	FCS	—	—
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	—	—

This configuration is relevant only if USART\_MODE=0xE or 0xF in “USART Mode Register” on page 865.

- **RSTRX: Reset Receiver**

0: No effect.  
 1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.  
 1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.  
 1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.  
 1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.  
 1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.  
 1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits OVRE, UNRE in US\_CSR.

- **FCS: Force SPI Chip Select**

– Applicable if USART operates in SPI Master Mode (USART\_MODE = 0xE):

FCS = 0: No effect.

FCS = 1: Forces the Slave Select Line NSS (RTS pin) to 0, even if USART is no transmitting, in order to address SPI slave devices supporting the CSAAT Mode (Chip Select Active After Transfer).

- **RCS: Release SPI Chip Select**

– Applicable if USART operates in SPI Master Mode (USART\_MODE = 0xE):

RCS = 0: No effect.

RCS = 1: Releases the Slave Select Line NSS (RTS pin).

### 37.8.3 USART Mode Register

**Name:** US\_MR

**Address:** 0x400A0004 (0), 0x400A4004 (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
ONEBIT	MODSYNC	MAN	FILTER	-		MAX_ITERATION	
23	22	21	20	19	18	17	16
INVDATA	VAR_SYNC	DSNACK	INACK	OVER	CLKO	MODE9	MSBF
15	14	13	12	11	10	9	8
CHMODE		NBSTOP			PAR		SYNC
7	6	5	4	3	2	1	0
CHRL		USCLKS			USART_MODE		

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 893.

For SPI configuration, see “USART Mode Register (SPI\_MODE)” on page 868.

- **USART\_MODE: USART Mode of Operation**

Value	Name	Description
0x0	NORMAL	Normal mode
0x1	RS485	RS485
0x2	HW_HANDSHAKING	Hardware Handshaking
0x3	MODEM	Modem
0x4	IS07816_T_0	IS07816 Protocol: T = 0
0x6	IS07816_T_1	IS07816 Protocol: T = 1
0x8	IRDA	IrDA
0xE	SPI_MASTER	SPI Master
0xF	SPI_SLAVE	SPI Slave

The PDC transfers are supported in all USART Mode of Operation.

- **USCLKS: Clock Selection**

Value	Name	Description
0	MCK	Master Clock MCK is selected
1	DIV	Internal Clock Divided MCK/DIV (DIV=) is selected
3	SCK	Serial Clock SLK is selected

- **CHRL: Character Length.**

Value	Name	Description
0	5_BIT	Character length is 5 bits
1	6_BIT	Character length is 6 bits
2	7_BIT	Character length is 7 bits
3	8_BIT	Character length is 8 bits

- **SYNC: Synchronous Mode Select**

0: USART operates in Asynchronous Mode.

1: USART operates in Synchronous Mode.

- **PAR: Parity Type**

Value	Name	Description
0	EVEN	Even parity
1	ODD	Odd parity
2	SPACE	Parity forced to 0 (Space)
3	MARK	Parity forced to 1 (Mark)
4	NO	No parity
6	MULTIDROP	Multidrop mode

- **NBSTOP: Number of Stop Bits**

Value	Name	Description
0	1_BIT	1 stop bit
1	1_5_BIT	1.5 stop bit (SYNC = 0) or reserved (SYNC = 1)
2	2_BIT	2 stop bits

- **CHMODE: Channel Mode**

Value	Name	Description
0	NORMAL	Normal Mode
1	AUTOMATIC	Automatic Echo. Receiver input is connected to the TXD pin.
2	LOCAL_LOOPBACK	Local Loopback. Transmitter output is connected to the Receiver Input.
3	REMOTE_LOOPBACK	Remote Loopback. RXD pin is internally connected to the TXD pin.

- **MSBF: Bit Order**

0: Least Significant Bit is sent/received first.

1: Most Significant Bit is sent/received first.

- **MODE9: 9-bit Character Length**

0: CHRL defines character length.

1: 9-bit character length.

- **CLKO: Clock Output Select**

0: The USART does not drive the SCK pin.

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK.

- **OVER: Oversampling Mode**

0: 16x Oversampling.

1: 8x Oversampling.

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX\_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITERATION is asserted.

- **INVDATA: Inverted Data**

0: The data field transmitted on TXD line is the same as the one written in US\_THR register or the content read in US\_RHR is the same as RXD line. Normal mode of operation.

1: The data field transmitted on TXD line is inverted (voltage polarity only) compared to the value written on US\_THR register or the content read in US\_RHR is inverted compared to what is received on RXD line (or ISO7816 IO line). Inverted Mode of operation, useful for contactless card application. To be used with configuration bit MSBF.

- **VAR\_SYNC: Variable Synchronization of Command/Data Sync Start Frame Delimiter**

0: User defined configuration of command or data sync field depending on MODSYNC value.

1: The sync field is updated when a character is written into US\_THR register.

- **MAX\_ITERATION: Maximum Number of Automatic Iteration**

0 - 7: Defines the maximum number of iterations in mode ISO7816, protocol T= 0.

- **FILTER: Infrared Receive Line Filter**

0: The USART does not filter the receive line.

1: The USART filters the receive line using a three-sample filter (1/16-bit clock) (2 over 3 majority).

- **MAN: Manchester Encoder/Decoder Enable**

0: Manchester Encoder/Decoder are disabled.

1: Manchester Encoder/Decoder are enabled.

- **MODSYNC: Manchester Synchronization Mode**

0: The Manchester Start bit is a 0 to 1 transition

1: The Manchester Start bit is a 1 to 0 transition.

- **ONEBIT: Start Frame Delimiter Selector**

0: Start Frame delimiter is COMMAND or DATA SYNC.

1: Start Frame delimiter is One Bit.

### 37.8.4 USART Mode Register (SPI\_MODE)

**Name:** US\_MR (SPI\_MODE)  
**Address:** 0x400A0004 (0), 0x400A4004 (1)  
**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	WRDBT	—	—	—	CPOL
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	CPHA
7	6	5	4	3	2	1	0
CHRL	USCLKS						USART_MODE

This configuration is relevant only if USART\_MODE=0xE or 0xF in [“USART Mode Register” on page 865](#).

This register can only be written if the WPEN bit is cleared in [“USART Write Protect Mode Register” on page 893](#).

- **USART\_MODE: USART Mode of Operation**

Value	Name	Description
0xE	SPI_MASTER	SPI Master
0xF	SPI_SLAVE	SPI Slave

- **USCLKS: Clock Selection**

Value	Name	Description
0	MCK	Master Clock MCK is selected
1	DIV	Internal Clock Divided MCK/DIV (DIV=) is selected
3	SCK	Serial Clock SLK is selected

- **CHRL: Character Length.**

Value	Name	Description
3	8_BIT	Character length is 8 bits

- **CPHA: SPI Clock Phase**

– Applicable if USART operates in SPI Mode (USART\_MODE = 0xE or 0xF):

CPHA = 0: Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

CPHA = 1: Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

CPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. CPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CHMODE: Channel Mode**

Value	Name	Description
0	NORMAL	Normal Mode
1	AUTOMATIC	Automatic Echo. Receiver input is connected to the TXD pin.
2	LOCAL_LOOPBACK	Local Loopback. Transmitter output is connected to the Receiver Input.
3	REMOTE_LOOPBACK	Remote Loopback. RXD pin is internally connected to the TXD pin.

- **CPOL: SPI Clock Polarity**

– Applicable if USART operates in SPI Mode (Slave or Master, USART\_MODE = 0xE or 0xF):

CPOL = 0: The inactive state value of SPCK is logic level zero.

CPOL = 1: The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with CPHA to produce the required clock/data relationship between master and slave devices.

- **WRDBT: Wait Read Data Before Transfer**

0: The character transmission starts as soon as a character is written into US\_THR register (assuming TXRDY was set).

1: The character transmission starts when a character is written and only if RXRDY flag is cleared (Receiver Holding Register has been read).

### 37.8.5 USART Interrupt Enable Register

**Name:** US\_IER

**Address:** 0x400A0008 (0), 0x400A4008 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24	
–	–	–	–	–	–	–	–	
23	22	21	20	19	18	17	16	
–	–	–		CTSIC	DCDIC	DSRIC	RIIC	
15	14	13	12	11	10	9	8	
–	–	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT	
7	6	5	4	3	2	1	0	
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY	

For SPI specific configuration, see “[USART Interrupt Enable Register \(SPI\\_MODE\)](#)” on page 872.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **RXBRK: Receiver Break Interrupt Enable**
- **ENDRX: End of Receive Transfer Interrupt Enable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Interrupt Enable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Time-out Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **ITER: Max number of Repetitions Reached Interrupt Enable**
- **TXBUFE: Buffer Empty Interrupt Enable (available in all USART modes of operation)**
- **RXBUFF: Buffer Full Interrupt Enable (available in all USART modes of operation)**
- **NACK: Non AcknowledgeInterrupt Enable**
- **RIIC: Ring Indicator Input Change Enable**
- **DSRIC: Data Set Ready Input Change Enable**
- **DCDIC: Data Carrier Detect Input Change Interrupt Enable**

- **CTSIC: Clear to Send Input Change Interrupt Enable**

- **MANE: Manchester Error Interrupt Enable**

0: No effect

1: Enables the corresponding interrupt.

### 37.8.6 USART Interrupt Enable Register (SPI\_MODE)

**Name:** US\_IER (SPI\_MODE)  
**Address:** 0x400A0008 (0), 0x400A4008 (1)  
**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART\_MODE=0xE or 0xF in “USART Mode Register” on page 865.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **UNRE: SPI Underrun Error Interrupt Enable**

0: No effect

1: Enables the corresponding interrupt.

### 37.8.7 USART Interrupt Disable Register

**Name:** US\_IDR

**Address:** 0x400A000C (0), 0x400A400C (1)

**Access:** Write-only

31	30	29	28	27	26	25	24	
—	—	—	—	—	—	—	—	
23	22	21	20	19	18	17	16	
—	—	—		CTSIC	DCDIC	DSRIC	RIIC	
15	14	13	12	11	10	9	8	
—	—	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT	
7	6	5	4	3	2	1	0	
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY	

For SPI specific configuration, see “[USART Interrupt Disable Register \(SPI\\_MODE\)](#)” on page 875.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **RXBRK: Receiver Break Interrupt Disable**
- **ENDRX: End of Receive Transfer Interrupt Disable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Interrupt Disable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Time-out Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **ITER: Max Number of Repetitions Reached Interrupt Disable**
- **TXBUFE: Buffer Empty Interrupt Disable (available in all USART modes of operation)**
- **RXBUFF: Buffer Full Interrupt Disable (available in all USART modes of operation)**
- **NACK: Non Acknowledge Interrupt Disable**
- **RIIC: Ring Indicator Input Change Disable**
- **DSRIC: Data Set Ready Input Change Disable**
- **DCDIC: Data Carrier Detect Input Change Interrupt Disable**

- **CTSIC: Clear to Send Input Change Interrupt Disable**

- **MANE: Manchester Error Interrupt Disable**

0: No effect

1: Disables the corresponding interrupt.

### 37.8.8 USART Interrupt Disable Register (SPI\_MODE)

**Name:** US\_IDR (SPI\_MODE)

**Address:** 0x400A000C (0), 0x400A400C (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART\_MODE=0xE or 0xF in “USART Mode Register” on page 865.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **UNRE: SPI Underrun Error Interrupt Disable**

0: No effect

1: Disables the corresponding interrupt.

### 37.8.9 USART Interrupt Mask Register

**Name:** US\_IMR

**Address:** 0x400A0010 (0), 0x400A4010 (1)

**Access:** Read-only

31	30	29	28	27	26	25	24	
–	–	–	–	–	–	–	–	
23	22	21	20	19	18	17	16	
–	–	–		CTSIC	DCDIC	DSRIC	RIIC	
15	14	13	12	11	10	9	8	
–	–	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT	
7	6	5	4	3	2	1	0	
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY	

For SPI specific configuration, see “[USART Interrupt Mask Register \(SPI\\_MODE\)](#)” on page 878.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **RXBRK: Receiver Break Interrupt Mask**
- **ENDRX: End of Receive Transfer Interrupt Mask (available in all USART modes of operation)**
- **ENDTX: End of Transmit Interrupt Mask (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Time-out Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **ITER: Max Number of Repetitions Reached Interrupt Mask**
- **TXBUFE: Buffer Empty Interrupt Mask (available in all USART modes of operation)**
- **RXBUFF: Buffer Full Interrupt Mask (available in all USART modes of operation)**
- **NACK: Non AcknowledgeInterrupt Mask**
- **RIIC: Ring Indicator Input Change Mask**
- **DSRIC: Data Set Ready Input Change Mask**
- **DCDIC: Data Carrier Detect Input Change Interrupt Mask**

- **CTSIC: Clear to Send Input Change Interrupt Mask**

- **MANE: Manchester Error Interrupt Mask**

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

### 37.8.10 USART Interrupt Mask Register (SPI\_MODE)

**Name:** US\_IMR (SPI\_MODE)  
**Address:** 0x400A0010 (0), 0x400A4010 (1)  
**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART\_MODE=0xE or 0xF in “USART Mode Register” on page 865.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **UNRE: SPI Underrun Error Interrupt Mask**

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

### 37.8.11 USART Channel Status Register

**Name:** US\_CSR

**Address:** 0x400A0014 (0), 0x400A4014 (1)

**Access:** Read-only

31	30	29	28	27	26	25	24	
—	—	—	—	—	—	—	—	RR
23	22	21	20	19	18	17	16	
CTS	DCD	DSR	RI	CTSIC	DCDIC	DSRIC	RIIC	
15	14	13	12	11	10	9	8	
—	—	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT	
7	6	5	4	3	2	1	0	
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY	

For SPI specific configuration, see “[USART Channel Status Register \(SPI\\_MODE\)](#)” on page 882.

- **RXRDY: Receiver Ready**

0: No complete character has been received since the last read of US\_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US\_RHR has not yet been read.

- **TXRDY: Transmitter Ready**

0: A character is in the US\_THR waiting to be transferred to the Transmit Shift Register, or an STTBRK command has been requested, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US\_THR.

- **RXBRK: Break Received/End of Break**

0: No Break received or End of Break detected since the last RSTSTA.

1: Break Received or End of Break detected since the last RSTSTA.

- **ENDRX: End of Receiver Transfer**

0: The End of Transfer signal from the Receive PDC channel is inactive.

1: The End of Transfer signal from the Receive PDC channel is active.

- **ENDTX: End of Transmitter Transfer**

0: The End of Transfer signal from the Transmit PDC channel is inactive.

1: The End of Transfer signal from the Transmit PDC channel is active.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0: No stop bit has been detected low since the last RSTSTA.

1: At least one stop bit has been detected low since the last RSTSTA.

- **PARE: Parity Error**

0: No parity error has been detected since the last RSTSTA.

1: At least one parity error has been detected since the last RSTSTA.

- **TIMEOUT: Receiver Time-out**

0: There has not been a time-out since the last Start Time-out command (STTTO in US\_CR) or the Time-out Register is 0.

1: There has been a time-out since the last Start Time-out command (STTTO in US\_CR).

- **TXEMPTY: Transmitter Empty**

0: There are characters in either US\_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US\_THR, nor in the Transmit Shift Register.

- **ITER: Max Number of Repetitions Reached**

0: Maximum number of repetitions has not been reached since the last RSTSTA.

1: Maximum number of repetitions has been reached since the last RSTSTA.

- **TXBUFE: Transmission Buffer Empty**

0: The signal Buffer Empty from the Transmit PDC channel is inactive.

1: The signal Buffer Empty from the Transmit PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The signal Buffer Full from the Receive PDC channel is inactive.

1: The signal Buffer Full from the Receive PDC channel is active.

- **NACK: Non AcknowledgeInterrupt**

0: Non Acknowledge has not been detected since the last RSTNACK.

1: At least one Non Acknowledge has been detected since the last RSTNACK.

- **RIIC: Ring Indicator Input Change Flag**

0: No input change has been detected on the RI pin since the last read of US\_CSR.

1: At least one input change has been detected on the RI pin since the last read of US\_CSR.

- **DSRIC: Data Set Ready Input Change Flag**

0: No input change has been detected on the DSR pin since the last read of US\_CSR.

1: At least one input change has been detected on the DSR pin since the last read of US\_CSR.

- **DCDIC: Data Carrier Detect Input Change Flag**

0: No input change has been detected on the DCD pin since the last read of US\_CSR.

1: At least one input change has been detected on the DCD pin since the last read of US\_CSR.

- **CTSIC: Clear to Send Input Change Flag**

0: No input change has been detected on the CTS pin since the last read of US\_CSR.

1: At least one input change has been detected on the CTS pin since the last read of US\_CSR.

- **RI: Image of RI Input**

0: RI is set to 0.

1: RI is set to 1.

- **DSR: Image of DSR Input**

0: DSR is set to 0

1: DSR is set to 1.

- **DCD: Image of DCD Input**

0: DCD is set to 0.

1: DCD is set to 1.

- **CTS: Image of CTS Input**

0: CTS is set to 0.

1: CTS is set to 1.

- **MANERR: Manchester Error**

0: No Manchester error has been detected since the last RSTSTA.

1: At least one Manchester error has been detected since the last RSTSTA.

### 37.8.12 USART Channel Status Register (SPI\_MODE)

**Name:** US\_CSR (SPI\_MODE)  
**Address:** 0x400A0014 (0), 0x400A4014 (1)  
**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART\_MODE=0xE or 0xF in “USART Mode Register” on page 865.

- **RXRDY: Receiver Ready**

0: No complete character has been received since the last read of US\_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US\_RHR has not yet been read.

- **TXRDY: Transmitter Ready**

0: A character is in the US\_THR waiting to be transferred to the Transmit Shift Register or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US\_THR.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty**

0: There are characters in either US\_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US\_THR, nor in the Transmit Shift Register.

- **UNRE: Underrun Error**

0: No SPI underrun error has occurred since the last RSTSTA.

1: At least one SPI underrun error has occurred since the last RSTSTA.

### 37.8.13 USART Receive Holding Register

**Name:** US\_RHR

**Address:** 0x400A0018 (0), 0x400A4018 (1)

**Access:** Read-only

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

- **RXCHR: Received Character**

Last character received if RXRDY is set.

- **RXSYNH: Received Sync**

0: Last Character received is a Data.

1: Last Character received is a Command.

### 37.8.14 USART Transmit Holding Register

**Name:** US\_THR

**Address:** 0x400A001C (0), 0x400A401C (1)

**Access:** Write-only

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

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

- **TXSYNH: Sync Field to be Transmitted**

0: The next character sent is encoded as a data. Start Frame Delimiter is DATA SYNC.

1: The next character sent is encoded as a command. Start Frame Delimiter is COMMAND SYNC.

### 37.8.15 USART Baud Rate Generator Register

**Name:** US\_BRGR

**Address:** 0x400A0020 (0), 0x400A4020 (1)

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 893.

- **CD: Clock Divider**

CD	USART_MODE ≠ ISO7816			USART_MODE = ISO7816
	SYNC = 0		SYNC = 1 or USART_MODE = SPI (Master or Slave)	
	OVER = 0	OVER = 1		
0	Baud Rate Clock Disabled			
1 to 65535	Baud Rate = Selected Clock/(16*CD)	Baud Rate = Selected Clock/(8*CD)	Baud Rate = Selected Clock /CD	Baud Rate = Selected Clock/(FI_DI_RATIO*CD)

- **FP: Fractional Part**

0: Fractional divider is disabled.

1 - 7: Baud rate resolution, defined by FP x 1/8.

### 37.8.16 USART Receiver Time-out Register

**Name:** US\_RTOR

**Address:** 0x400A0024 (0), 0x400A4024 (1)

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 893.

- **TO: Time-out Value**

0: The Receiver Time-out is disabled.

1 - 65535: The Receiver Time-out is enabled and the Time-out delay is TO x Bit Period.

### 37.8.17 USART Transmitter Timeguard Register

**Name:** US\_TTGR

**Address:** 0x400A0028 (0), 0x400A4028 (1)

**Access:** Read-write

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
TG							

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 893.

- **TG: Timeguard Value**

0: The Transmitter Timeguard is disabled.

1 - 255: The Transmitter timeguard is enabled and the timeguard delay is TG x Bit Period.

### 37.8.18 USART FI DI RATIO Register

**Name:** US\_FIDI

**Address:** 0x400A0040 (0), 0x400A4040 (1)

**Access:** Read-write

**Reset Value:** 0x174

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

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 893.

- **FI\_DI\_RATIO: FI Over DI Ratio Value**

0: If ISO7816 mode is selected, the Baud Rate Generator generates no signal.

1 - : If ISO7816 mode is selected, the Baud Rate is the clock provided on SCK divided by FI\_DI\_RATIO.

### 37.8.19 USART Number of Errors Register

**Name:** US\_NER

**Address:** 0x400A0044 (0), 0x400A4044 (1)

**Access:** Read-only

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
NB_ERRORS							

This register is relevant only if USART\_MODE=0x4 or 0x6 in “USART Mode Register” on page 865.

- **NB\_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

### 37.8.20 USART IrDA FILTER Register

**Name:** US\_IF

**Address:** 0x400A004C (0), 0x400A404C (1)

**Access:** Read-write

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
IRDA_FILTER							

This register is relevant only if USART\_MODE=0x8 in “USART Mode Register” on page 865.

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 893.

- **IRDA\_FILTER: IrDA Filter**

The IRDA\_FILTER value must be defined to meet the following criteria:

$$T_{MCK} * (IRDA\_FILTER + 3) < 1.41 \text{ us}$$

### 37.8.21 USART Manchester Configuration Register

**Name:** US\_MAN

**Address:** 0x400A0050 (0), 0x400A4050 (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
–	DRIFT	ONE	RX_MPOL	–	–	RX_PP	
23	22	21	20	19	18	17	16
–	–	–	–		RX_PL		
15	14	13	12	11	10	9	8
–	–	–	TX_MPOL	–	–	TX_PP	
7	6	5	4	3	2	1	0
–	–	–	–		TX_PL		

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 893.

- **TX\_PL: Transmitter Preamble Length**

0: The Transmitter Preamble pattern generation is disabled

1 - 15: The Preamble Length is TX\_PL x Bit Period

- **TX\_PP: Transmitter Preamble Pattern**

The following values assume that TX\_MPOL field is not set:

Value	Name	Description
00	ALL_ONE	The preamble is composed of '1's
01	ALL_ZERO	The preamble is composed of '0's
10	ZERO_ONE	The preamble is composed of '01's
11	ONE_ZERO	The preamble is composed of '10's

- **TX\_MPOL: Transmitter Manchester Polarity**

0: Logic Zero is coded as a zero-to-one transition, Logic One is coded as a one-to-zero transition.

1: Logic Zero is coded as a one-to-zero transition, Logic One is coded as a zero-to-one transition.

- **RX\_PL: Receiver Preamble Length**

0: The receiver preamble pattern detection is disabled

1 - 15: The detected preamble length is RX\_PL x Bit Period

- **RX\_PP: Receiver Preamble Pattern detected**

The following values assume that RX\_MPOL field is not set:

Value	Name	Description
00	ALL_ONE	The preamble is composed of '1's

01	ALL_ZERO	The preamble is composed of '0's
10	ZERO_ONE	The preamble is composed of '01's
11	ONE_ZERO	The preamble is composed of '10's

- **RX\_MPOL: Receiver Manchester Polarity**

0: Logic Zero is coded as a zero-to-one transition, Logic One is coded as a one-to-zero transition.

1: Logic Zero is coded as a one-to-zero transition, Logic One is coded as a zero-to-one transition.

- **ONE: Must Be Set to 1**

Bit 29 must always be set to 1 when programming the US\_MAN register.

- **DRIFT: Drift Compensation**

0: The USART can not recover from an important clock drift

1: The USART can recover from clock drift. The 16X clock mode must be enabled.

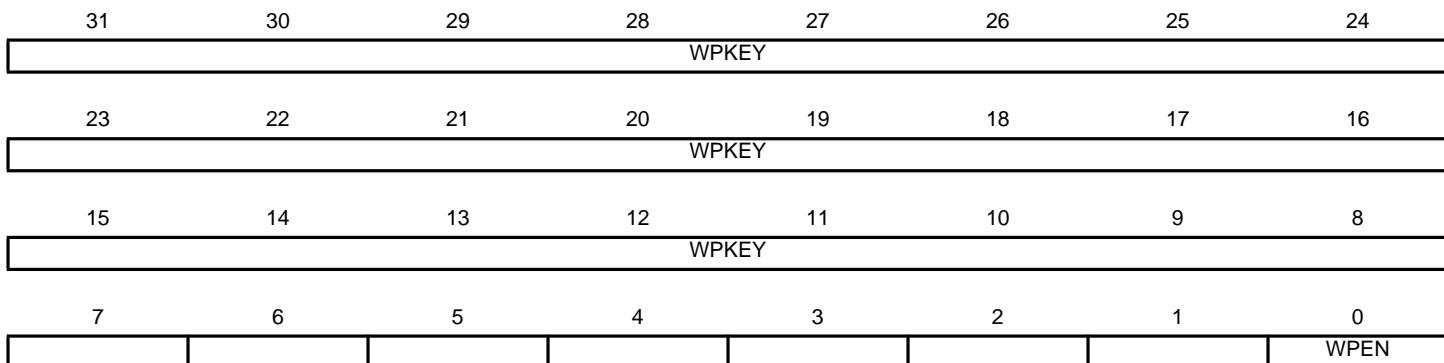
### 37.8.22 USART Write Protect Mode Register

**Name:** US\_WPMR

**Address:** 0x400A00E4 (0), 0x400A40E4 (1)

**Access:** Read-write

**Reset:** See [Table 37-16](#)



- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x555341 ("USA" in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x555341 ("USA" in ASCII).

Protects the registers:

- ["USART Mode Register" on page 865](#)
- ["USART Baud Rate Generator Register" on page 885](#)
- ["USART Receiver Time-out Register" on page 886](#)
- ["USART Transmitter Timeguard Register" on page 887](#)
- ["USART F1 DI RATIO Register" on page 888](#)
- ["USART IrDA FILTER Register" on page 890](#)
- ["USART Manchester Configuration Register" on page 891](#)

- **WPKEY: Write Protect KEY.**

Value	Name	Description
0x555341	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

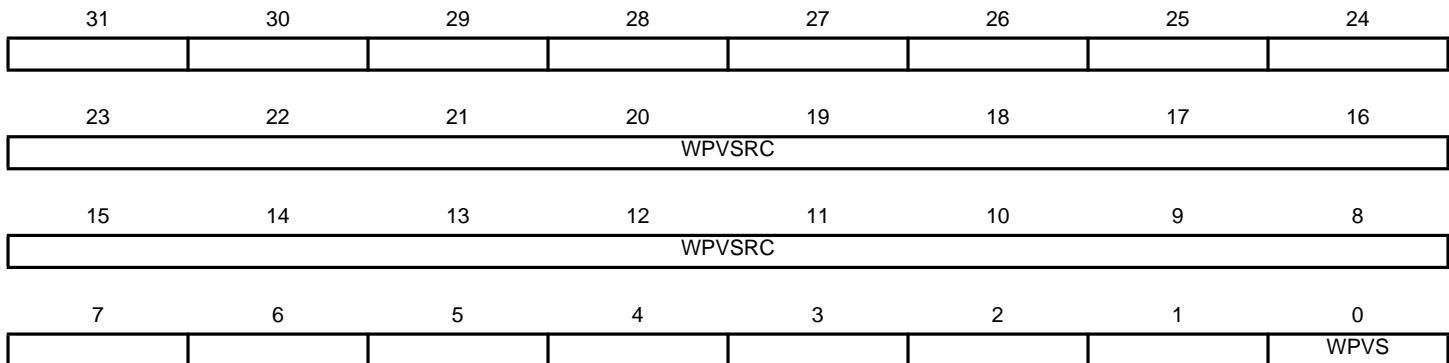
### 37.8.23 USART Write Protect Status Register

**Name:** US\_WPSR

**Address:** 0x400A00E8 (0), 0x400A40E8 (1)

**Access:** Read-only

**Reset:** See [Table 37-16](#)



- **WPVS: Write Protect Violation Status**

0 = No Write Protect Violation has occurred since the last read of the US\_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the US\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Note: Reading US\_WPSR automatically clears all fields.

## 38. Timer Counter (TC)

### 38.1 Description

The Timer Counter (TC) includes nine identical 32-bit Timer Counter channels.

Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

The Timer Counter (TC) embeds a quadrature decoder logic connected in front of the timers and driven by TIOA0, TIOB0 and TIOB1 inputs. When enabled, the quadrature decoder performs the input lines filtering, decoding of quadrature signals and connects to the timers/counters in order to read the position and speed of the motor through the user interface.

The Timer Counter block has two global registers which act upon all TC channels.

The Block Control Register allows the channels to be started simultaneously with the same instruction.

The Block Mode Register defines the external clock inputs for each channel, allowing them to be chained.

[Table 38-1](#) gives the assignment of the device Timer Counter clock inputs common to Timer Counter 0 to 2.

**Table 38-1. Timer Counter Clock Assignment**

Name	Definition
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5 <sup>(1)</sup>	SLCK

Note: 1. When Slow Clock is selected for Master Clock (CSS = 0 in PMC Master Clock Register), TIMER\_CLOCK5 input is equivalent to Master Clock.

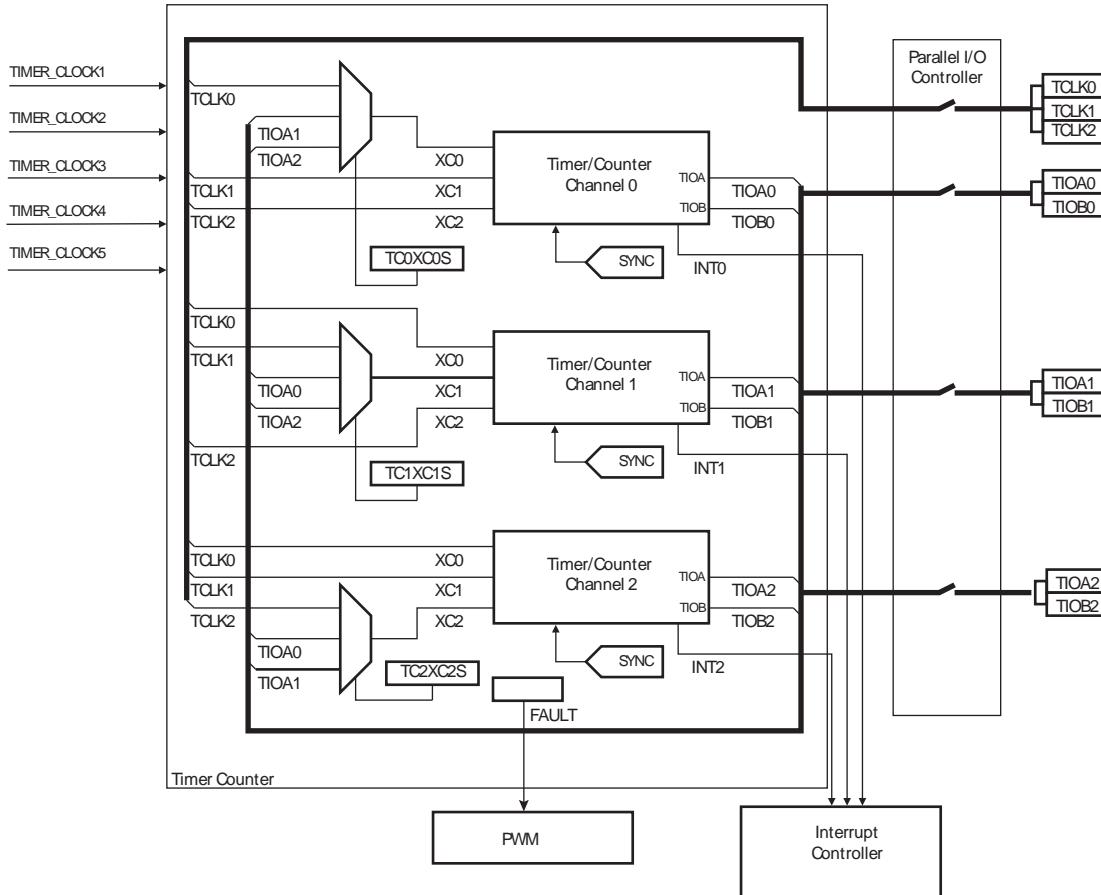
### 38.2 Embedded Characteristics

- Provides nine 32-bit Timer Counter channels
- Wide range of functions including:
  - Frequency measurement
  - Event counting
  - Interval measurement
  - Pulse generation
  - Delay timing
  - Pulse Width Modulation
  - Up/down capabilities
  - Quadrature decoder logic
  - 2-bit gray up/down count for stepper motor
- Each channel is user-configurable and contains:
  - Three external clock inputs
  - Five Internal clock inputs
  - Two multi-purpose input/output signals acting as trigger event
  - Trigger/capture events can be directly synchronized by PWM signals

- Internal interrupt signal
- Two global registers that act on all TC channels
- Read of the Capture registers by the PDC
- Compare event fault generation for PWM
- Configuration registers can be write protected

### 38.3 Block Diagram

Figure 38-1. Timer Counter Block Diagram



Note: The quadrature decoder logic connections are detailed in [Figure 38-17 "Predefined Connection of the Quadrature Decoder with Timer Counters"](#)

Table 38-2. Signal Name Description

Block/Channel	Signal Name	Description
Channel Signal	XC0, XC1, XC2	External Clock Inputs
	TIOA	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Output
	TIOB	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Input/Output
	INT	Interrupt Signal Output (internal signal)
	SYNC	Synchronization Input Signal (from configuration register)

## 38.4 Pin Name List

**Table 38-3. TC pin list**

Pin Name	Description	Type
TCLK0-TCLK2	External Clock Input	Input
TIOA0-TIOA2	I/O Line A	I/O
TIOB0-TIOB2	I/O Line B	I/O

## 38.5 Product Dependencies

### 38.5.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the TC pins to their peripheral functions.

**Table 38-4. I/O Lines**

Instance	Signal	I/O Line	Peripheral
TC0	TCLK0	PA4	B
TC0	TCLK1	PA28	B
TC0	TCLK2	PA29	B
TC0	TIOA0	PA0	B
TC0	TIOA1	PA15	B
TC0	TIOA2	PA26	B
TC0	TIOB0	PA1	B
TC0	TIOB1	PA16	B
TC0	TIOB2	PA27	B
TC1	TCLK3	PC25	B
TC1	TCLK4	PC28	B
TC1	TCLK5	PC31	B
TC1	TIOA3	PC23	B
TC1	TIOA4	PC26	B
TC1	TIOA5	PC29	B
TC1	TIOB3	PC24	B
TC1	TIOB4	PC27	B
TC1	TIOB5	PC30	B
TC2	TCLK6	PC7	B
TC2	TCLK7	PC10	B
TC2	TCLK8	PC14	B
TC2	TIOA6	PC5	B
TC2	TIOA7	PC8	B
TC2	TIOA8	PC11	B

**Table 38-4. I/O Lines**

TC2	TIOB6	PC6	B
TC2	TIOB7	PC9	B
TC2	TIOB8	PC12	B

### 38.5.2 Power Management

The TC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the Timer Counter clock.

### 38.5.3 Interrupt

The TC has an interrupt line connected to the Interrupt Controller (IC). Handling the TC interrupt requires programming the IC before configuring the TC.

### 38.5.4 Synchronization inputs from PWM

The TC has trigger/capture inputs internally connected to the PWM. Refer to [Section 38.6.14 "Synchronization with PWM"](#), and to the product Pulse Width Modulation (PWM) implementation.

### 38.5.5 Fault Output

The TC has the FAULT output internally connected to the fault input of PWM. Refer to [Section 38.6.19 "Fault Mode"](#), and to the product Pulse Width Modulation (PWM) implementation.

## 38.6 Functional Description

### 38.6.1 TC Description

The nine channels of the Timer Counter are independent and identical in operation except when quadrature decoder is enabled. The registers for channel programming are listed in [Table 38-5 on page 921](#).

### 38.6.2 32-bit Counter

Each channel is organized around a 32-bit counter. The value of the counter is incremented at each positive edge of the selected clock. When the counter has reached the value 0xFFFF and passes to 0x0000, an overflow occurs and the COVFS bit in TC\_SR (Status Register) is set.

The current value of the counter is accessible in real time by reading the Counter Value Register, TC\_CV. The counter can be reset by a trigger. In this case, the counter value passes to 0x0000 on the next valid edge of the selected clock.

### 38.6.3 Clock Selection

At block level, input clock signals of each channel can either be connected to the external inputs TCLK0, TCLK1 or TCLK2, or be connected to the internal I/O signals TIOA0, TIOA1 or TIOA2 for chaining by programming the TC\_BMR (Block Mode). See [Figure 38-2 "Clock Chaining Selection"](#).

Each channel can independently select an internal or external clock source for its counter:

- Internal clock signals: TIMER\_CLOCK1, TIMER\_CLOCK2, TIMER\_CLOCK3, TIMER\_CLOCK4, TIMER\_CLOCK5
- External clock signals: XC0, XC1 or XC2

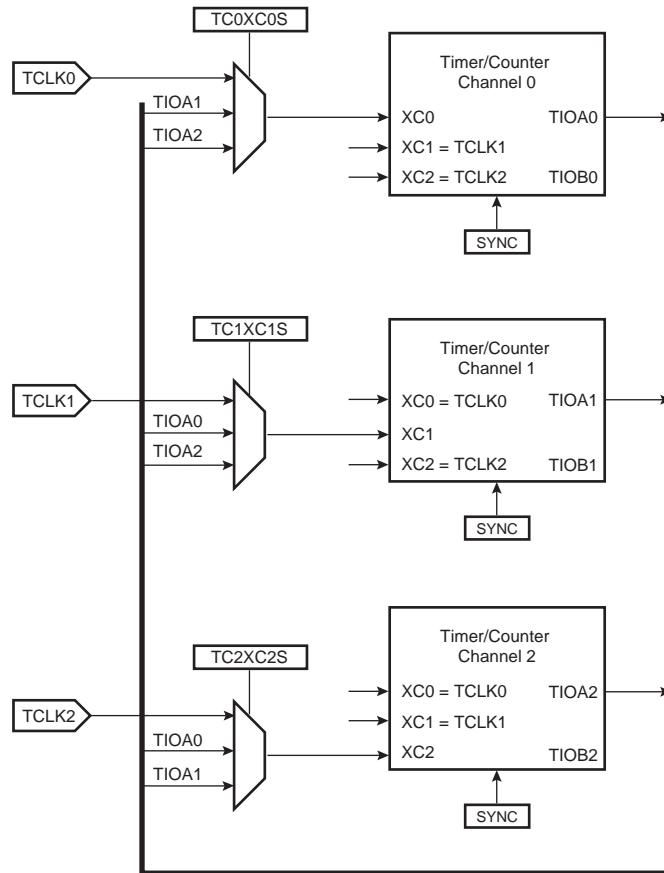
This selection is made by the TCCLKS bits in the TC Channel Mode Register.

The selected clock can be inverted with the CLKI bit in TC\_CM.R. This allows counting on the opposite edges of the clock.

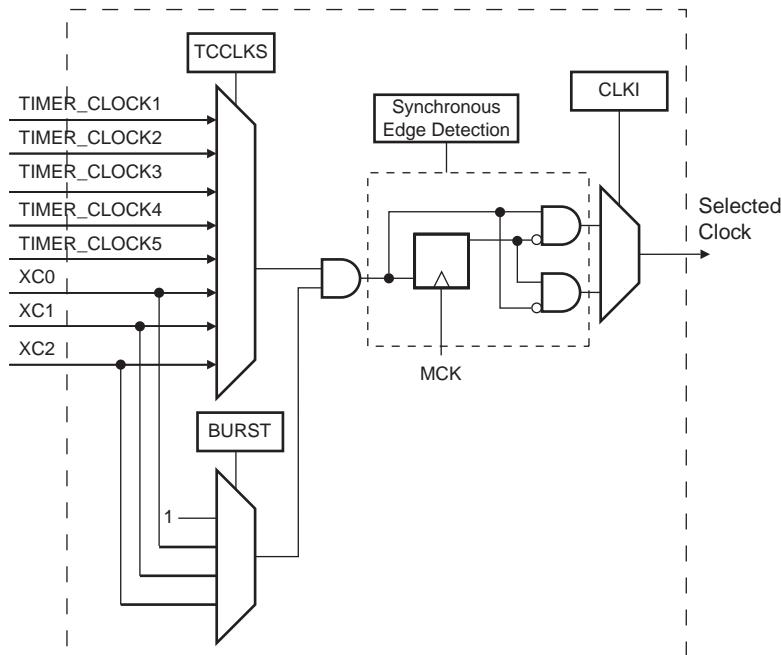
The burst function allows the clock to be validated when an external signal is high. The BURST parameter in the Mode Register defines this signal (none, XC0, XC1, XC2). See [Figure 38-3 "Clock Selection"](#)

Note: In all cases, if an external clock is used, the duration of each of its levels must be longer than the master clock period. The external clock frequency must be at least 2.5 times lower than the master clock

**Figure 38-2. Clock Chaining Selection**



**Figure 38-3. Clock Selection**

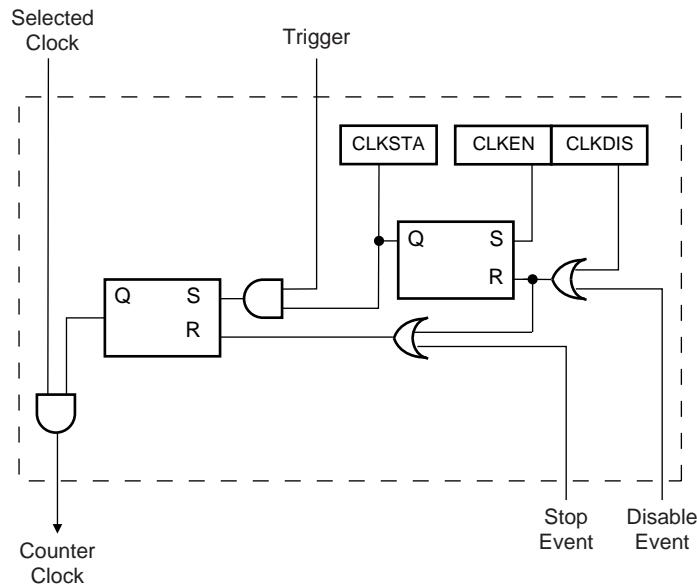


### 38.6.4 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 38-4.

- The clock can be enabled or disabled by the user with the CLKEN and the CLKDIS commands in the Control Register. In Capture Mode it can be disabled by an RB load event if LBDIS is set to 1 in TC\_CM.R. In Waveform Mode, it can be disabled by an RC Compare event if CPCDIS is set to 1 in TC\_CM.R. When disabled, the start or the stop actions have no effect: only a CLKEN command in the Control Register can re-enable the clock. When the clock is enabled, the CLKSTA bit is set in the Status Register.
- The clock can also be started or stopped: a trigger (software, synchro, external or compare) always starts the clock. The clock can be stopped by an RB load event in Capture Mode (LDBSTOP = 1 in TC\_CM.R) or a RC compare event in Waveform Mode (CPCSTOP = 1 in TC\_CM.R). The start and the stop commands have effect only if the clock is enabled.

Figure 38-4. Clock Control



### 38.6.5 TC Operating Modes

Each channel can independently operate in two different modes:

- Capture Mode provides measurement on signals.
- Waveform Mode provides wave generation.

The TC Operating Mode is programmed with the WAVE bit in the TC Channel Mode Register.

In Capture Mode, TIOA and TIOB are configured as inputs.

In Waveform Mode, TIOA is always configured to be an output and TIOB is an output if it is not selected to be the external trigger.

### 38.6.6 Trigger

A trigger resets the counter and starts the counter clock. Three types of triggers are common to both modes, and a fourth external trigger is available to each mode.

Regardless of the trigger used, it will be taken into account at the following active edge of the selected clock. This means that the counter value can be read differently from zero just after a trigger, especially when a low frequency signal is selected as the clock.

The following triggers are common to both modes:

- Software Trigger: Each channel has a software trigger, available by setting SWTRG in TC\_CCR.
- SYNC: Each channel has a synchronization signal SYNC. When asserted, this signal has the same effect as a software trigger. The SYNC signals of all channels are asserted simultaneously by writing TC\_BCR (Block Control) with SYNC set.
- Compare RC Trigger: RC is implemented in each channel and can provide a trigger when the counter value matches the RC value if CPCTRG is set in TC\_CMR.

The channel can also be configured to have an external trigger. In Capture Mode, the external trigger signal can be selected between TIOA and TIOB. In Waveform Mode, an external event can be programmed on one of the following signals: TIOB, XC0, XC1 or XC2. This external event can then be programmed to perform a trigger by setting ENETRG in TC\_CMR.

If an external trigger is used, the duration of the pulses must be longer than the master clock period in order to be detected.

### 38.6.7 Capture Operating Mode

This mode is entered by clearing the WAVE parameter in TC\_CMR (Channel Mode Register).

Capture Mode allows the TC channel to perform measurements such as pulse timing, frequency, period, duty cycle and phase on TIOA and TIOB signals which are considered as inputs.

[Figure 38-6](#) shows the configuration of the TC channel when programmed in Capture Mode.

### 38.6.8 Capture Registers A and B

Registers A and B (RA and RB) are used as capture registers. This means that they can be loaded with the counter value when a programmable event occurs on the signal TIOA.

The LDRA parameter in TC\_CMR defines the TIOA selected edge for the loading of register A, and the LDRB parameter defines the TIOA selected edge for the loading of Register B.

The subsampling ratio defined by the SBSMPLR field in TC\_CMR is applied to these selected edges, so that the loading of Register A and Register B occurs once every 1, 2, 4, 8 or 16 selected edges.

RA is loaded only if it has not been loaded since the last trigger or if RB has been loaded since the last loading of RA.

RB is loaded only if RA has been loaded since the last trigger or the last loading of RB.

Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS) in TC\_SR (Status Register). In this case, the old value is overwritten.

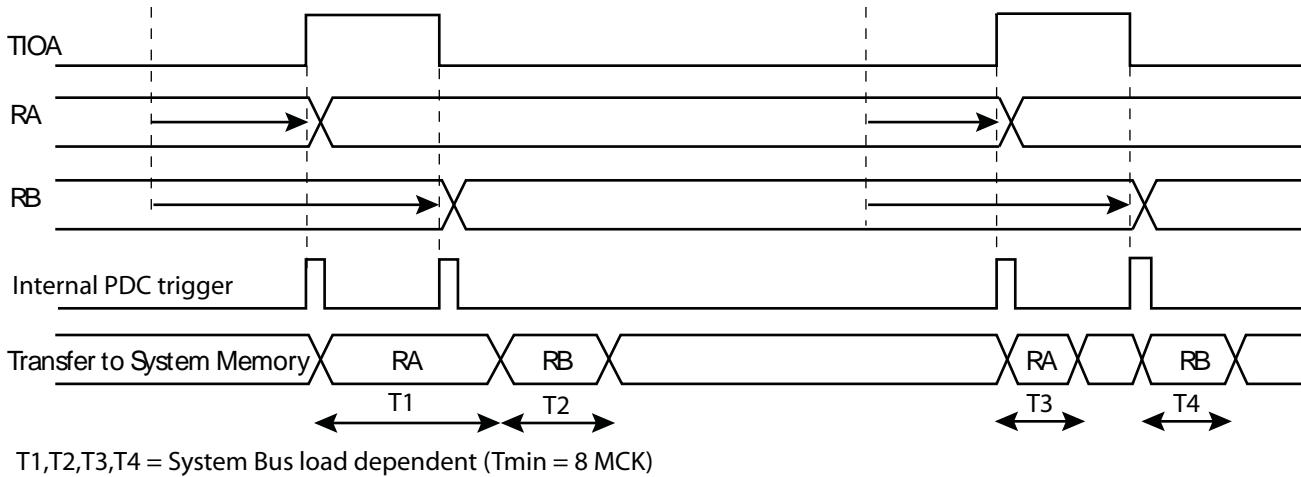
When DMA is used, the RAB register address must be configured as source address of the transfer. The RAB register provides the next unread value from Register A and Register B. It may be read by the **DMA** after a request has been triggered upon loading Register A or Register B.

### 38.6.9 Transfer with PDC

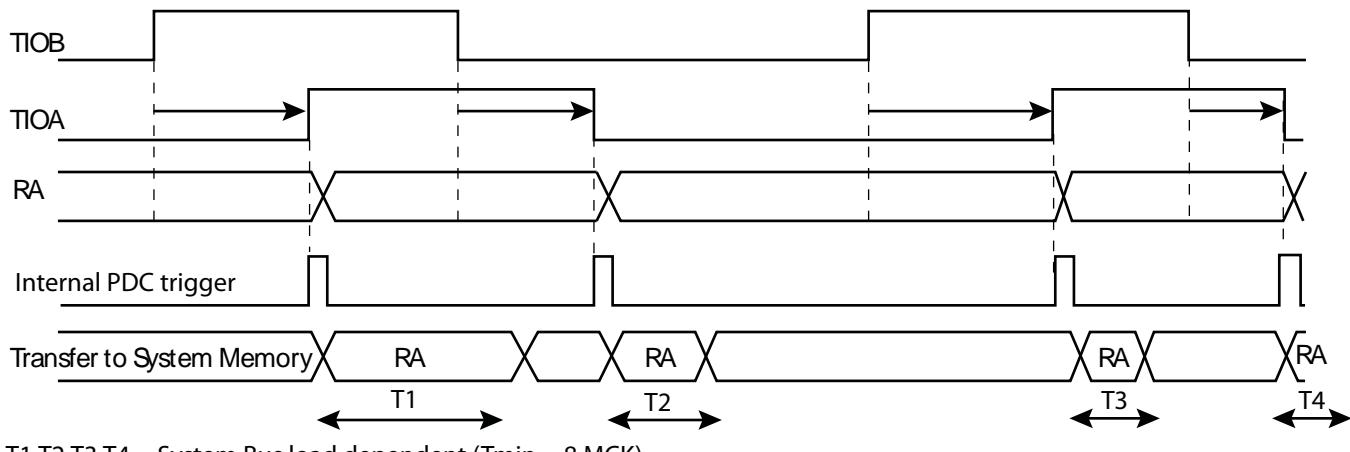
The PDC can only perform access from timer to system memory.

The figure "[Example of transfer with PDC](#)" on page 903 illustrates how the RA and RB registers can be loaded in the system memory without CPU intervention.

**Figure 38-5. Example of transfer with PDC**



$\text{ETRGEDG}=3, \text{LDRA}=3, \text{LDRB}=0, \text{ABETRG}=0$

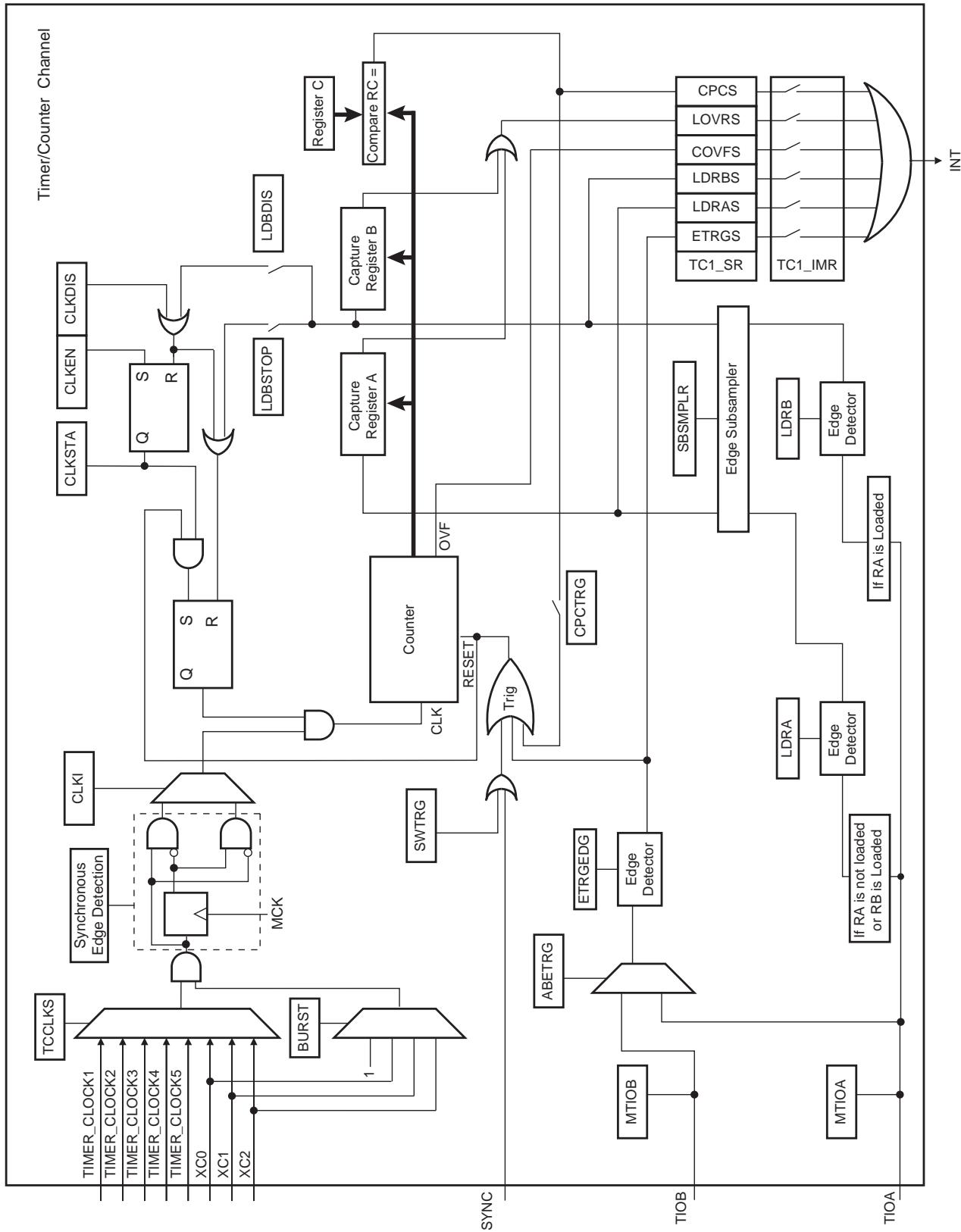


### 38.6.10 Trigger Conditions

In addition to the SYNC signal, the software trigger and the RC compare trigger, an external trigger can be defined.

The ABETRG bit in the TC\_CMR register selects TIOA or TIOB input signal as an external trigger or the trigger signal from the output comparator of the PWM module. The ETRGEDG parameter defines the edge (rising, falling or both) detected to generate an external trigger. If ETRGEDG = 0 (none), the external trigger is disabled.

**Figure 38-6. Capture Mode**



### 38.6.11 Waveform Operating Mode

Waveform operating mode is entered by setting the WAVE parameter in TC\_CMR (Channel Mode Register).

In Waveform Operating Mode the TC channel generates 1 or 2 PWM signals with the same frequency and independently programmable duty cycles, or generates different types of one-shot or repetitive pulses.

In this mode, TIOA is configured as an output and TIOB is defined as an output if it is not used as an external event (EEVT parameter in TC\_CMR).

[Figure 38-7](#) shows the configuration of the TC channel when programmed in Waveform Operating Mode.

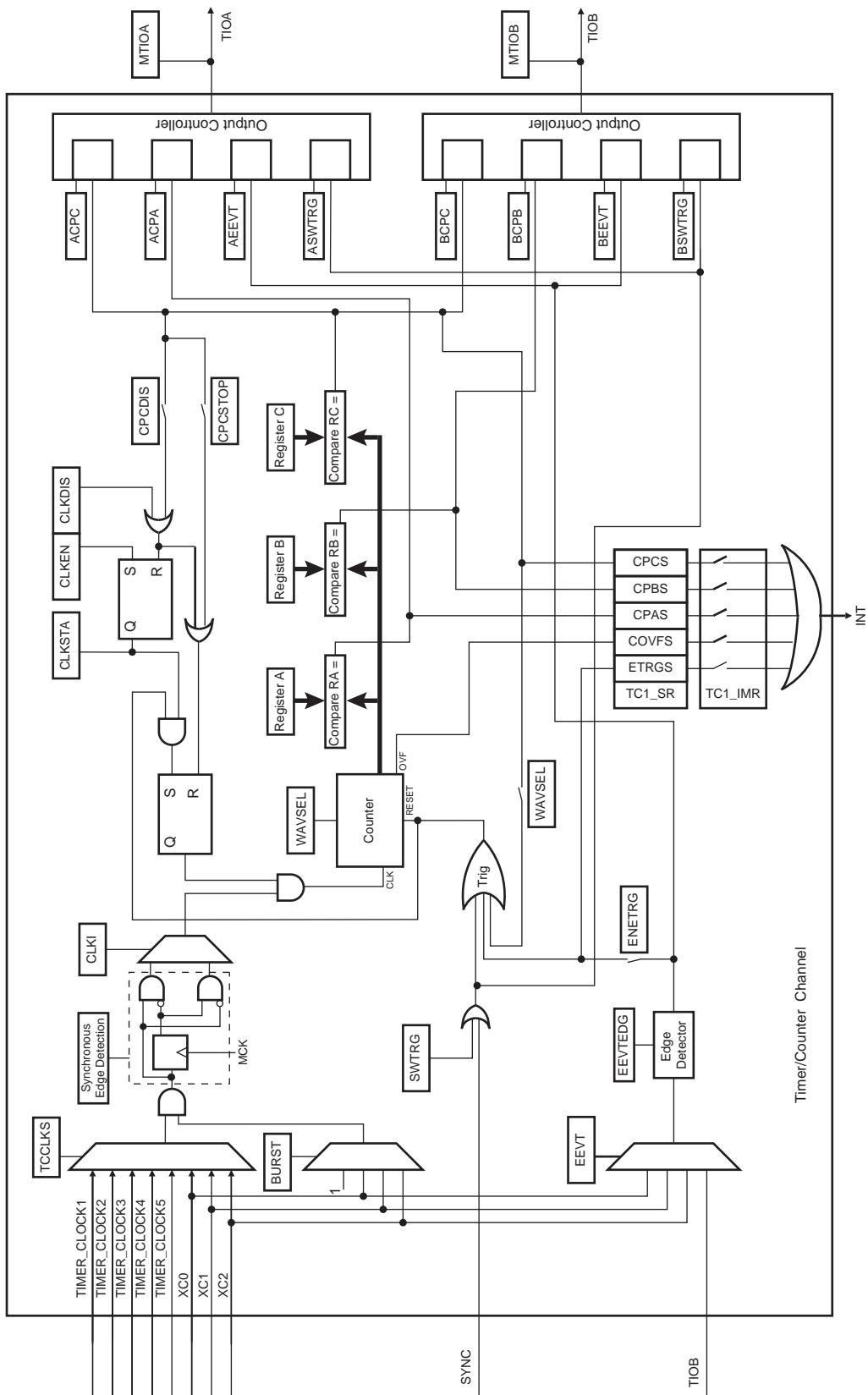
### 38.6.12 Waveform Selection

Depending on the WAVSEL parameter in TC\_CMR (Channel Mode Register), the behavior of TC\_CV varies.

With any selection, RA, RB and RC can all be used as compare registers.

RA Compare is used to control the TIOA output, RB Compare is used to control the TIOB output (if correctly configured) and RC Compare is used to control TIOA and/or TIOB outputs.

**Figure 38-7. Waveform Mode**



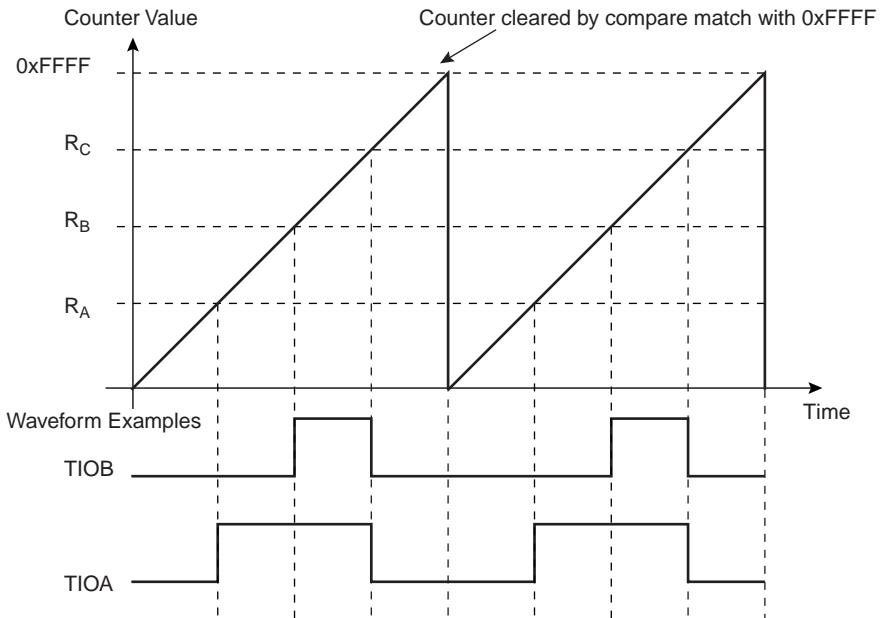
### 38.6.12.1 WAVSEL = 00

When WAVSEL = 00, the value of TC\_CV is incremented from 0 to 0xFFFF. Once 0xFFFF has been reached, the value of TC\_CV is reset. Incrementation of TC\_CV starts again and the cycle continues. See [Figure 38-8](#).

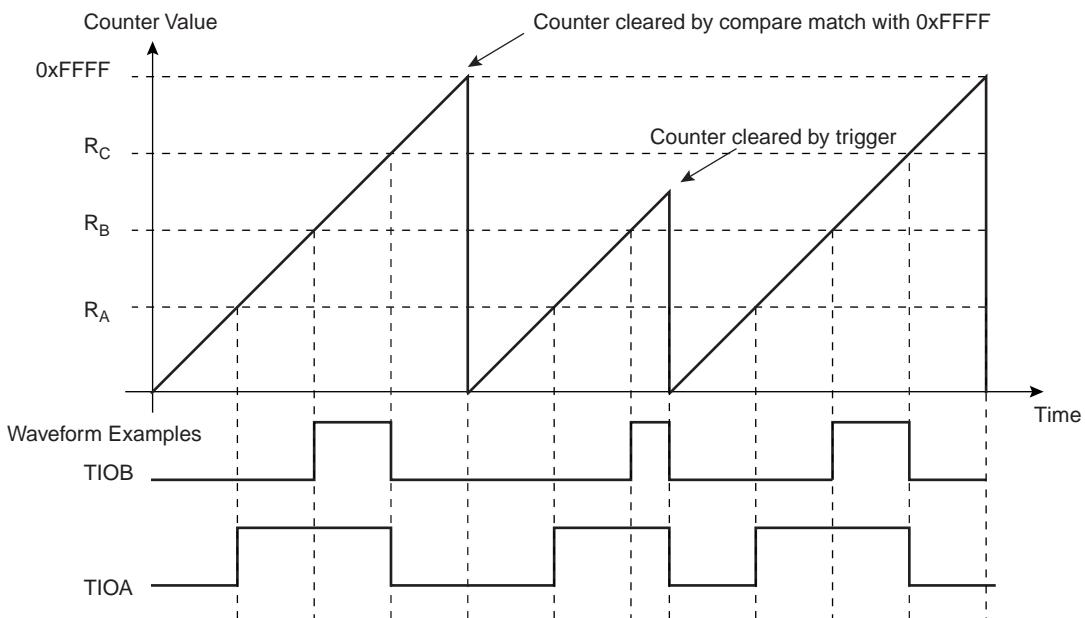
An external event trigger or a software trigger can reset the value of TC\_CV. It is important to note that the trigger may occur at any time. See [Figure 38-9](#).

RC Compare cannot be programmed to generate a trigger in this configuration. At the same time, RC Compare can stop the counter clock (CPCSTOP = 1 in TC\_CMRR) and/or disable the counter clock (CPCDIS = 1 in TC\_CMRR).

**Figure 38-8. WAVSEL= 00 without trigger**



**Figure 38-9. WAVSEL= 00 with trigger**



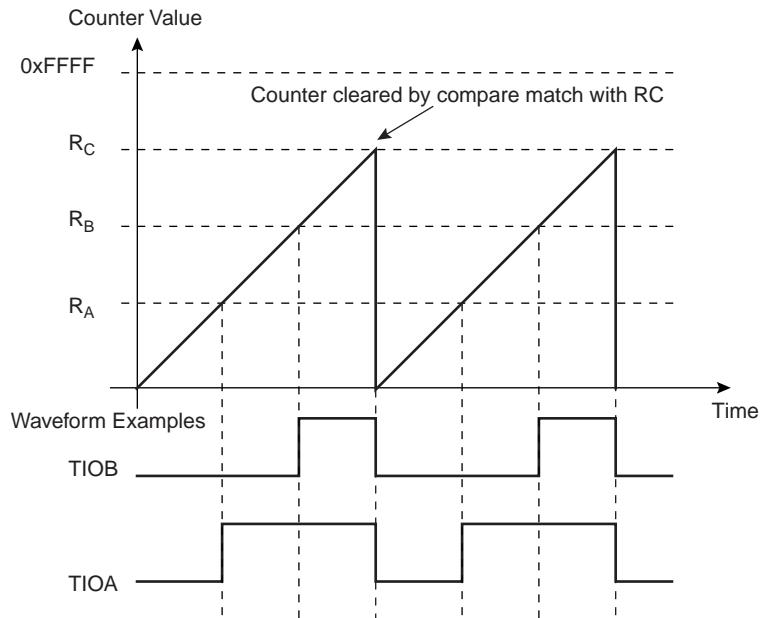
### 38.6.12.2 WAVSEL = 10

When WAVSEL = 10, the value of TC\_CV is incremented from 0 to the value of RC, then automatically reset on a RC Compare. Once the value of TC\_CV has been reset, it is then incremented and so on. See [Figure 38-10](#).

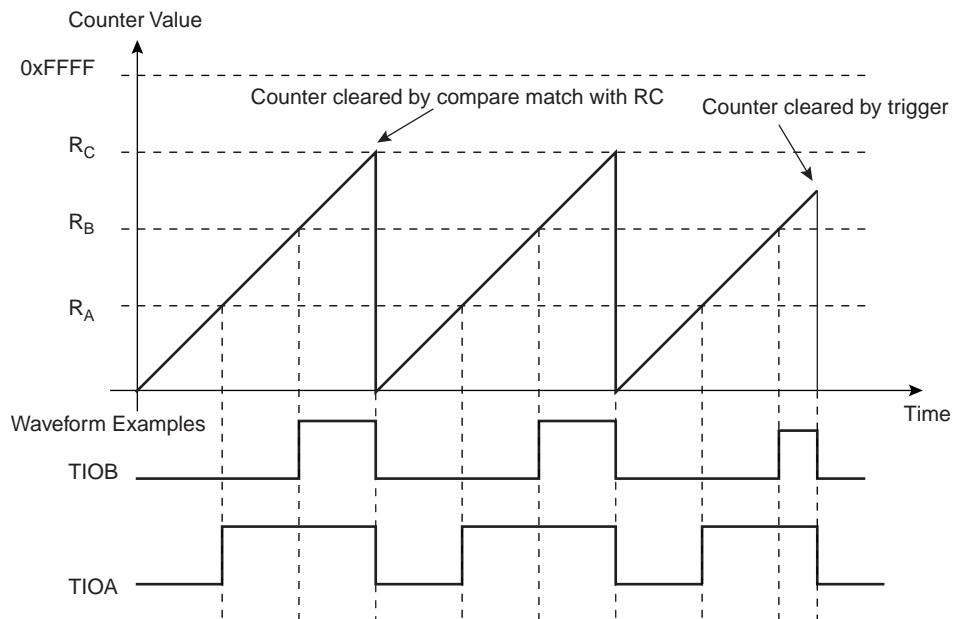
It is important to note that TC\_CV can be reset at any time by an external event or a software trigger if both are programmed correctly. See [Figure 38-11](#).

In addition, RC Compare can stop the counter clock (CPCSTOP = 1 in TC\_CMR) and/or disable the counter clock (CPCDIS = 1 in TC\_CMR).

**Figure 38-10. WAVSEL = 10 Without Trigger**



**Figure 38-11. WAVSEL = 10 With Trigger**



### 38.6.12.3 WAVSEL = 01

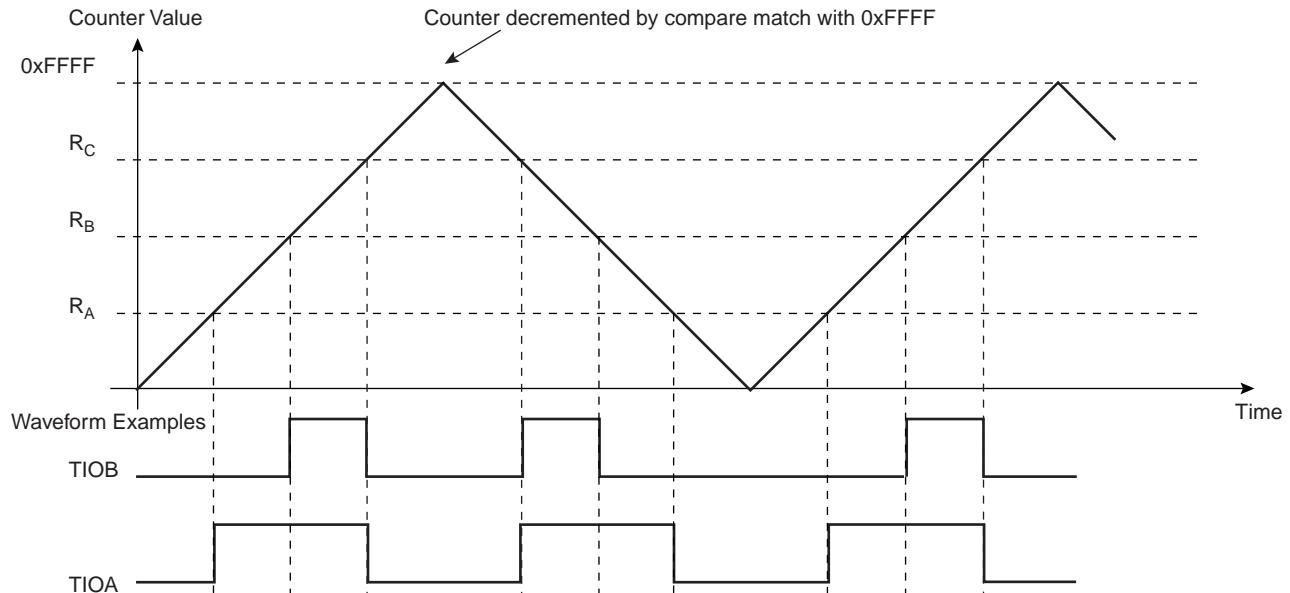
When WAVSEL = 01, the value of TC\_CV is incremented from 0 to 0xFFFF. Once 0xFFFF is reached, the value of TC\_CV is decremented to 0, then re-incremented to 0xFFFF and so on. See [Figure 38-12](#).

A trigger such as an external event or a software trigger can modify TC\_CV at any time. If a trigger occurs while TC\_CV is incrementing, TC\_CV then decrements. If a trigger is received while TC\_CV is decrementing, TC\_CV then increments. See [Figure 38-13](#).

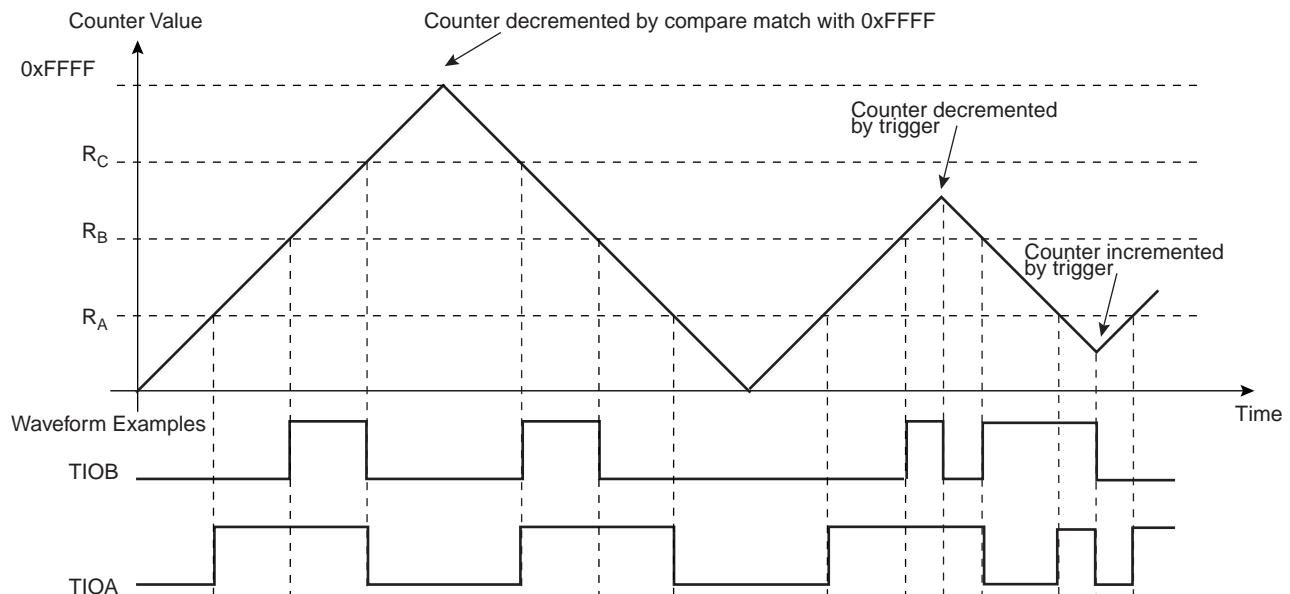
RC Compare cannot be programmed to generate a trigger in this configuration.

At the same time, RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

**Figure 38-12. WAVSEL = 01 Without Trigger**



**Figure 38-13. WAVSEL = 01 With Trigger**



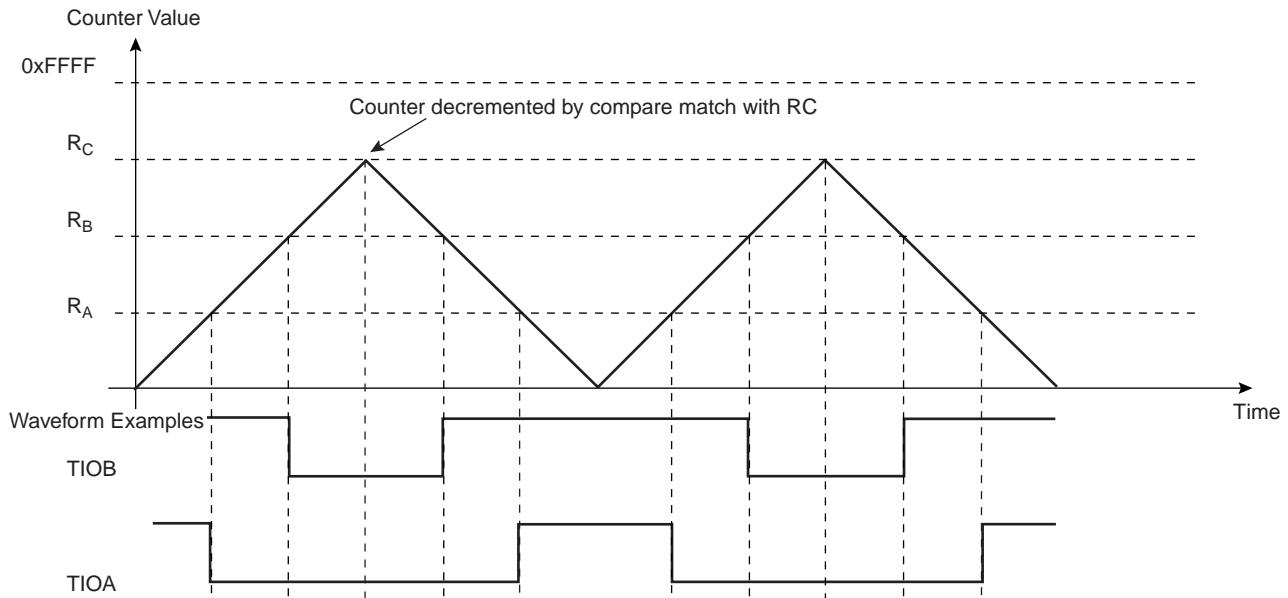
#### 38.6.12.4 WAVSEL = 11

When WAVSEL = 11, the value of TC\_CV is incremented from 0 to RC. Once RC is reached, the value of TC\_CV is decremented to 0, then re-incremented to RC and so on. See [Figure 38-14](#).

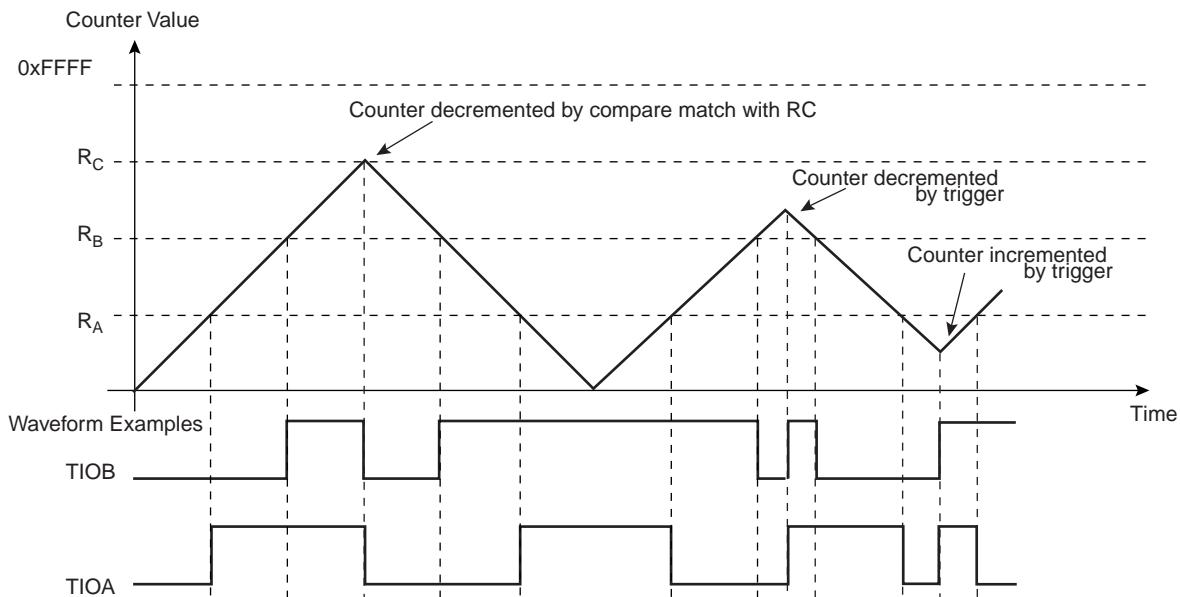
A trigger such as an external event or a software trigger can modify TC\_CV at any time. If a trigger occurs while TC\_CV is incrementing, TC\_CV then decrements. If a trigger is received while TC\_CV is decrementing, TC\_CV then increments. See [Figure 38-15](#).

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

**Figure 38-14. WAVSEL = 11 Without Trigger**



**Figure 38-15. WAVSEL = 11 With Trigger**



### 38.6.13 External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOB. The external event selected can then be used as a trigger.

The EEVT parameter in TC\_CMR selects the external trigger. The EEVTEDG parameter defines the trigger edge for each of the possible external triggers (rising, falling or both). If EEVTEDG is cleared (none), no external event is defined.

If TIOB is defined as an external event signal (EEVT = 0), TIOB is no longer used as an output and the compare register B is not used to generate waveforms and subsequently no IRQs. In this case the TC channel can only generate a waveform on TIOA.

When an external event is defined, it can be used as a trigger by setting bit ENETRG in TC\_CMR.

As in Capture Mode, the SYNC signal and the software trigger are also available as triggers. RC Compare can also be used as a trigger depending on the parameter WAVSEL.

### 38.6.14 Synchronization with PWM

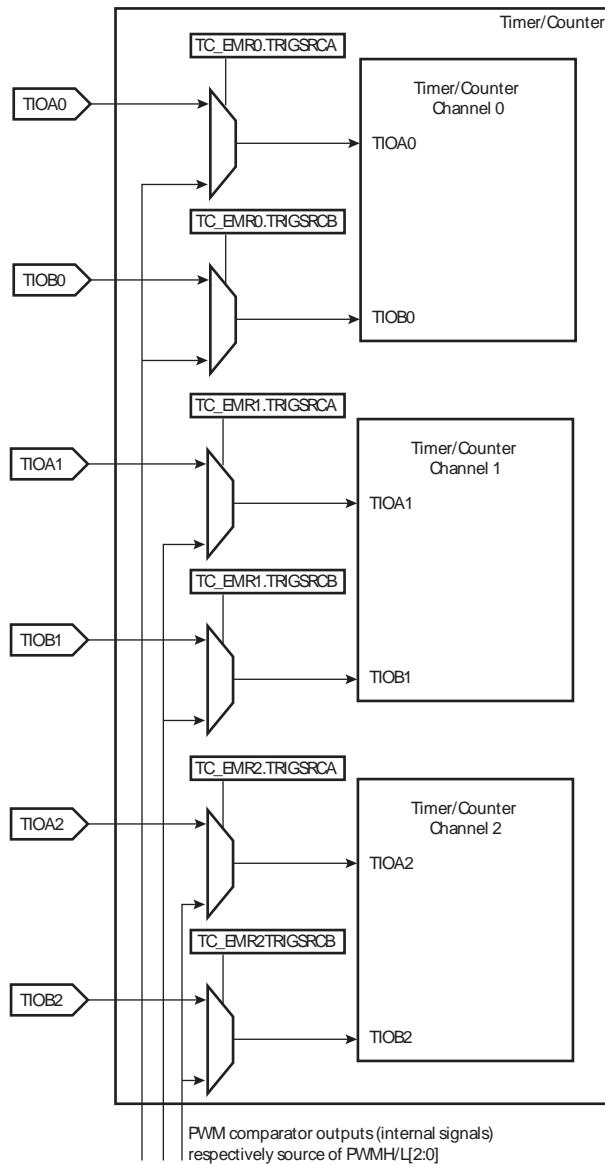
The inputs TIOAB[2:0] can be bypassed, and thus channel trigger/capture events can be directly driven by the independent PWM module.

PWM comparator outputs (internal signals without dead-time insertion), respectively source of the PWMH/L[2:0] outputs, are routed to the internal TC inputs. These specific TC inputs are multiplexed with TIOA/B input signal to drive the internal trigger/capture events.

The selection can be programmed in the extended mode register TC\_EMR fields TRGSRCA and TRGSRCB (see “[TC Extended Mode Register](#)” on page 943).

Each channel of the TC module can be synchronized by a different PWM channel as described in [Figure 38-16 “Synchronization with PWM”](#).

**Figure 38-16. Synchronization with PWM**



### 38.6.15 Output Controller

The output controller defines the output level changes on TIOA and TIOB following an event. TIOB control is used only if TIOB is defined as output (not as an external event).

The following events control TIOA and TIOB: software trigger, external event and RC compare. RA compare controls TIOA and RB compare controls TIOB. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC\_CMR.

### 38.6.16 Quadrature Decoder Logic

#### 38.6.16.1 Description

The quadrature decoder logic is driven by TIOA0, TIOB0, TIOB1 input pins and drives the timer/counter of channel 0 and 1. Channel 2 can be used as a time base in case of speed measurement requirements (refer to [Figure 38-17 "Predefined Connection of the Quadrature Decoder with Timer Counters"](#)).

When writing 0 in the QDEN field of the TC\_BMR register, the quadrature decoder logic is totally transparent.

TIOA0 and TIOB0 are to be driven by the 2 dedicated quadrature signals from a rotary sensor mounted on the shaft of the off-chip motor.

A third signal from the rotary sensor can be processed through pin TIOB1 and is typically dedicated to be driven by an index signal if it is provided by the sensor. This signal is not required to decode the quadrature signals PHA, PHB.

TCCLKS field of TC\_CMR channels must be configured to select XC0 input (i.e. 0x101). TC0XC0S field has no effect as soon as quadrature decoder is enabled.

Either speed or position/revolution can be measured. Position channel 0 accumulates the edges of PHA, PHB input signals giving a high accuracy on motor position whereas channel 1 accumulates the index pulses of the sensor, therefore the number of rotations. Concatenation of both values provides a high level of precision on motion system position.

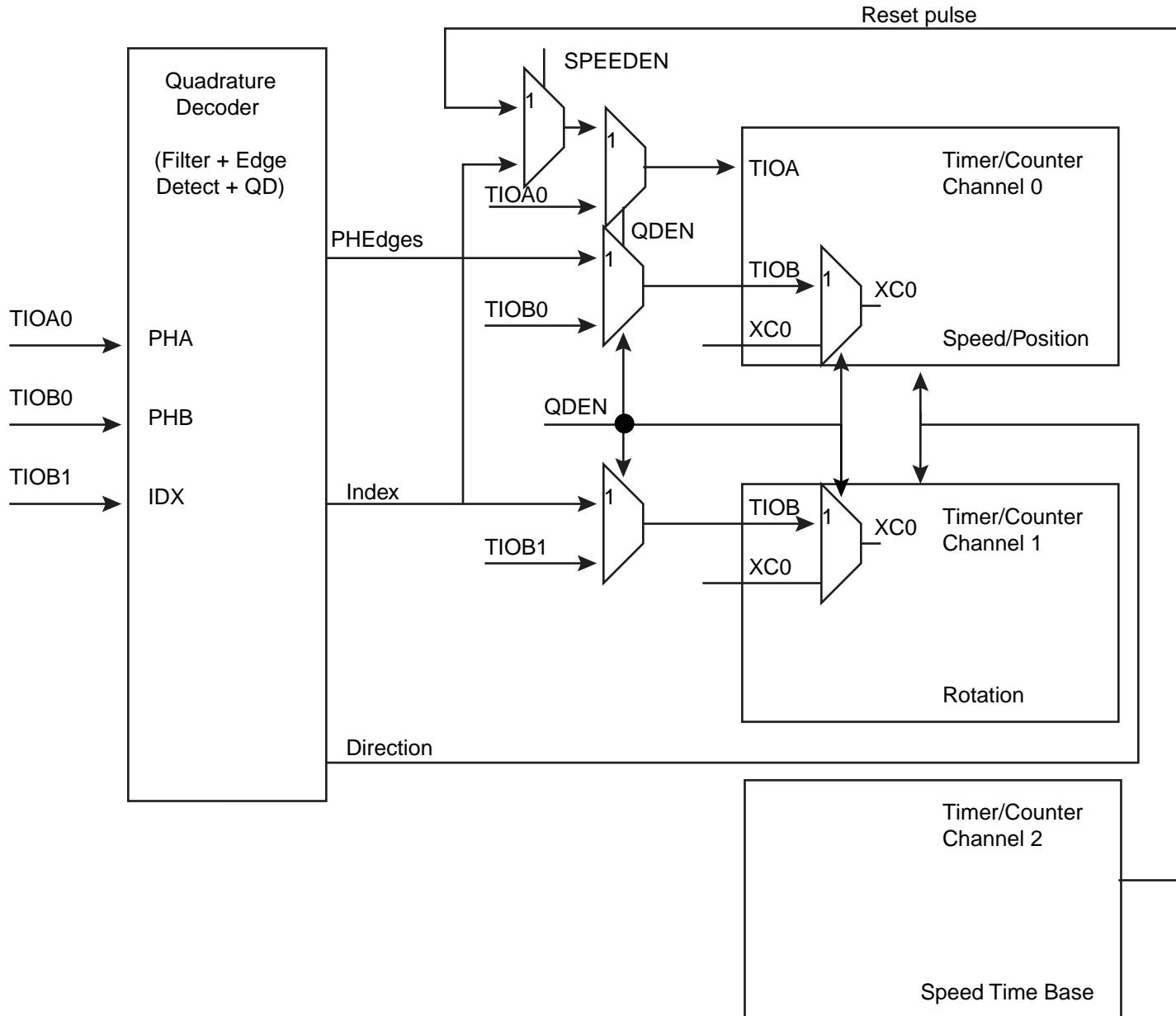
In speed mode, position cannot be measured but revolution can be measured.

Inputs from the rotary sensor can be filtered prior to down-stream processing. Accommodation of input polarity, phase definition and other factors are configurable.

Interruptions can be generated on different events.

A compare function (using TC\_RC register) is available on channel 0 (speed/position) or channel 1 (rotation) and can generate an interrupt by means of the CPCS flag in the TC\_SR registers.

Figure 38-17. Predefined Connection of the Quadrature Decoder with Timer Counters



### 38.6.16.2 Input Pre-processing

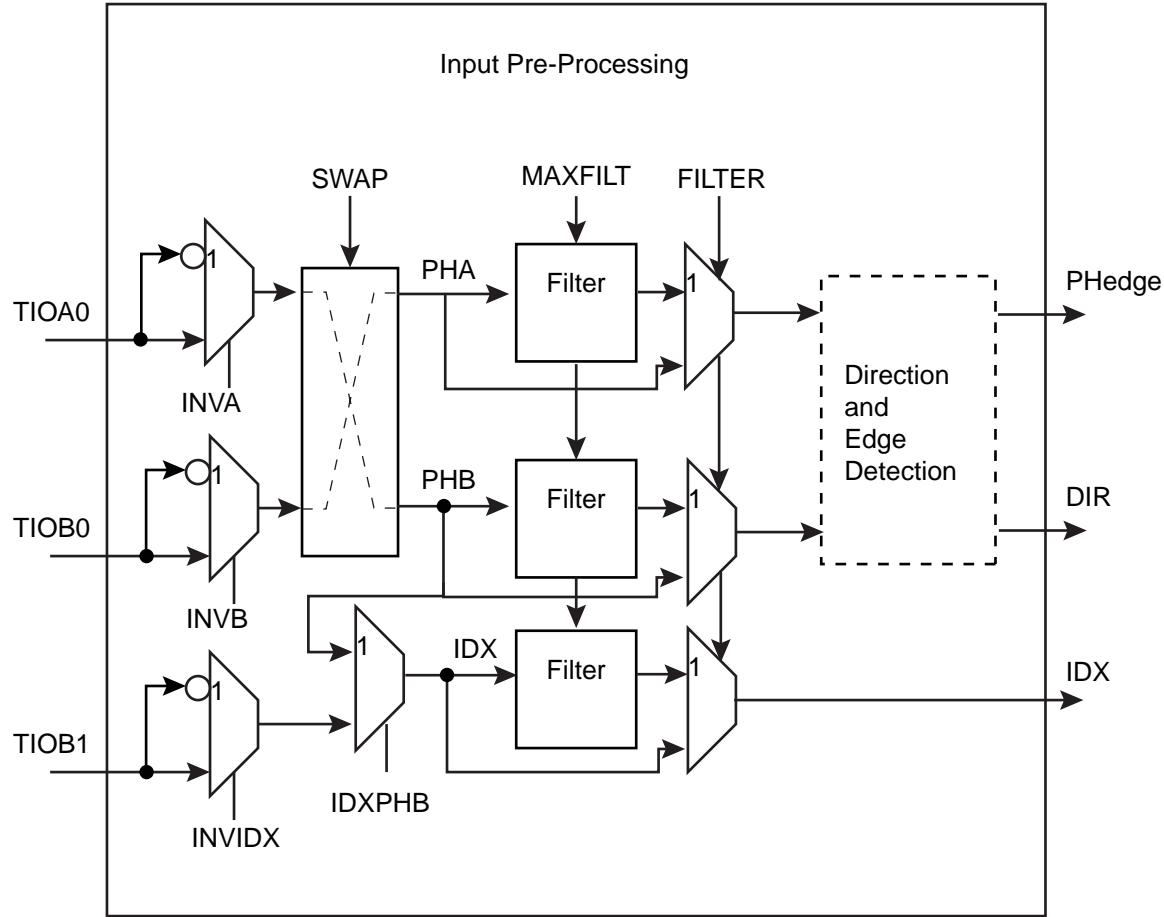
Input pre-processing consists of capabilities to take into account rotary sensor factors such as polarities and phase definition followed by configurable digital filtering.

Each input can be negated and swapped PHA, PHB is also configurable.

By means of the MAXFILT field in TC\_BMR, it is possible to configure a minimum duration for which the pulse is stated as valid. When the filter is active, pulses with a duration lower than MAXFILT+1 \* tMCK ns are not passed to downstream logic.

Filters can be disabled using the FILTER field in the TC\_BMR register.

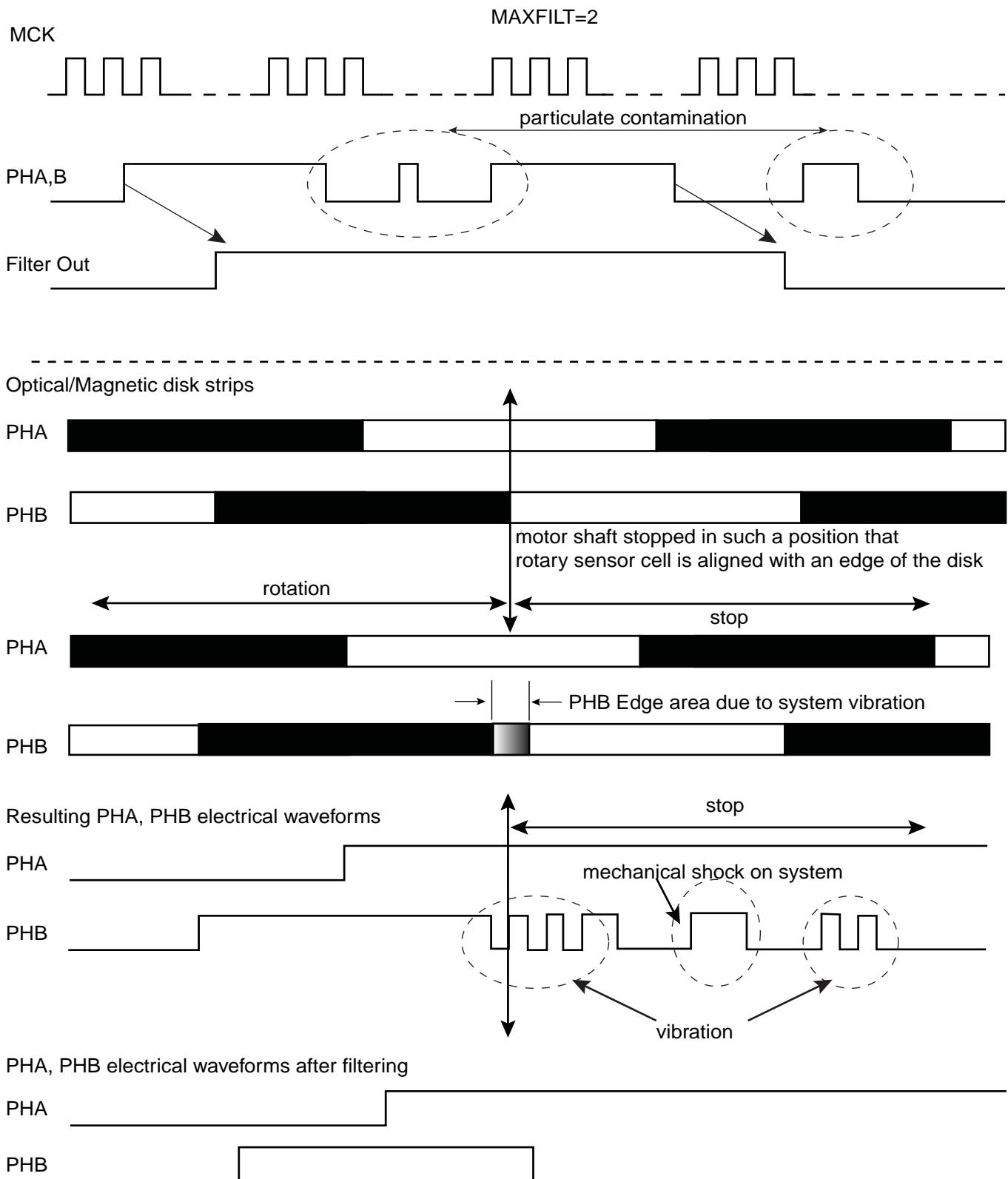
Figure 38-18. Input Stage



Input filtering can efficiently remove spurious pulses that might be generated by the presence of particulate contamination on the optical or magnetic disk of the rotary sensor.

Spurious pulses can also occur in environments with high levels of electro-magnetic interference. Or, simply if vibration occurs even when rotation is fully stopped and the shaft of the motor is in such a position that the beginning of one of the reflective or magnetic bars on the rotary sensor disk is aligned with the light or magnetic (Hall) receiver cell of the rotary sensor. Any vibration can make the PHA, PHB signals toggle for a short duration.

### Figure 38-19. Filtering Examples



### 38.6.16.3 Direction Status and Change Detection

After filtering, the quadrature signals are analyzed to extract the rotation direction and edges of the 2 quadrature signals detected in order to be counted by timer/counter logic downstream.

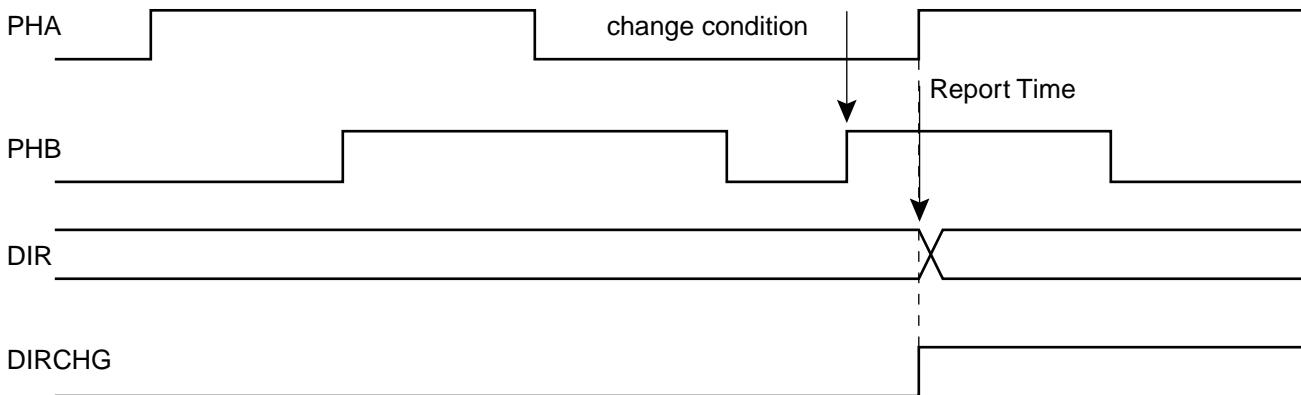
The direction status can be directly read at anytime on TC\_QISR register. The polarity of the direction flag status depends on the configuration written in TC\_BMR register. INVA, INVB, INVIDX, SWAP modify the polarity of DIR flag.

Any change in rotation direction is reported on TC\_QISR register and can generate an interrupt.

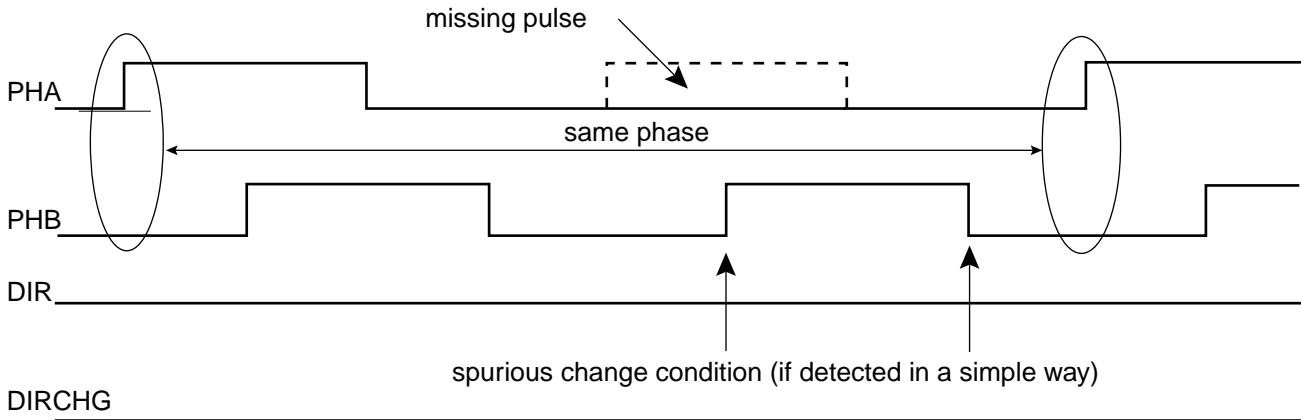
The direction change condition is reported as soon as 2 consecutive edges on a phase signal have sampled the same value on the other phase signal and there is an edge on the other signal. The 2 consecutive edges of 1 phase signal sampling the same value on other phase signal is not sufficient to declare a direction change, for the reason that particulate contamination may mask one or more reflective bar on the optical or magnetic disk of the sensor. (Refer to Figure 38-20 "Rotation Change Detection" for waveforms.)

**Figure 38-20. Rotation Change Detection**

Direction Change under normal conditions



No direction change due to particulate contamination masking a reflective bar

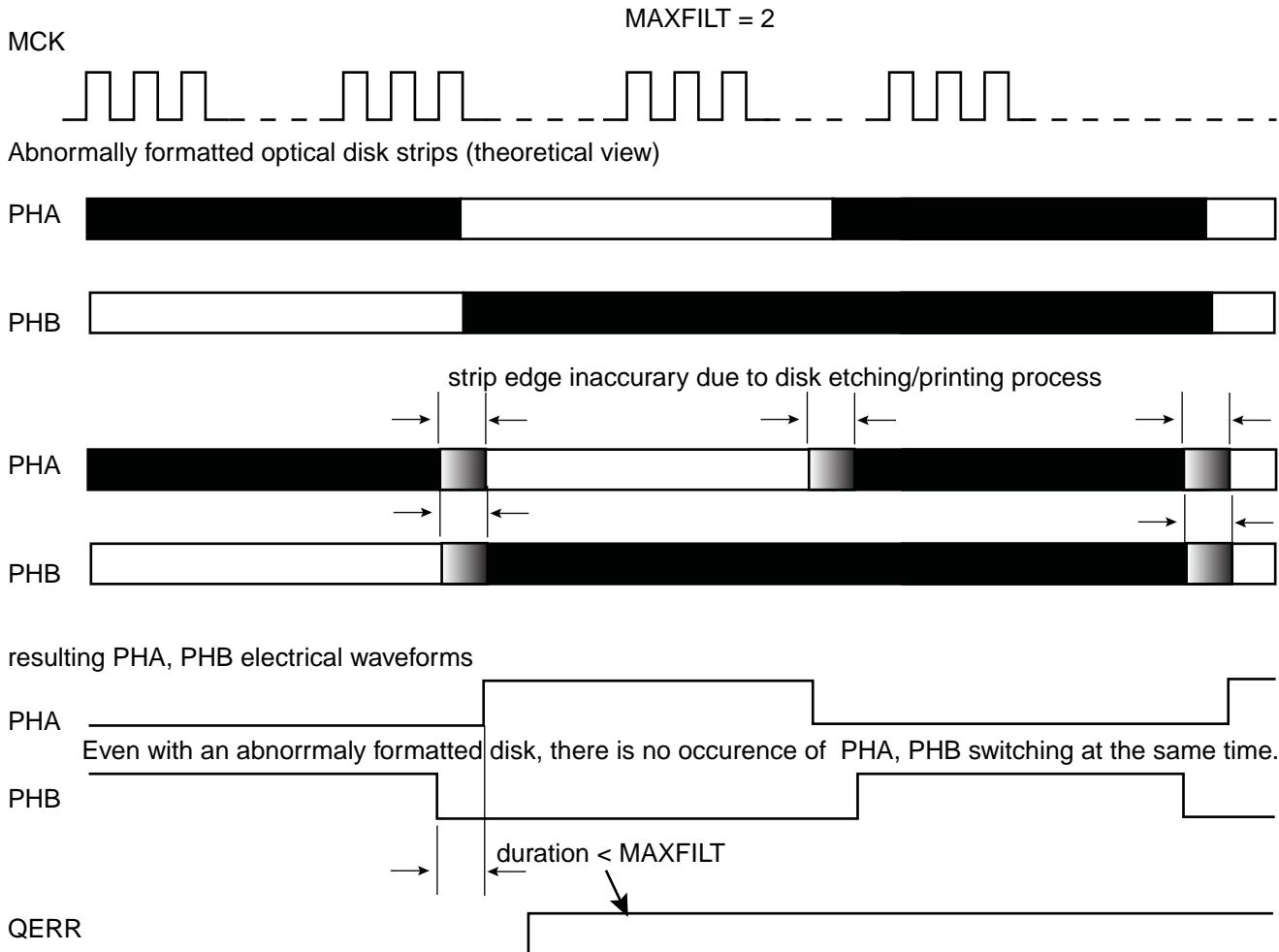


The direction change detection is disabled when QDTRANS is set to 1 in TC\_BMR. In this case the DIR flag report must not be used.

A quadrature error is also reported by the quadrature decoder logic. Rather than reporting an error only when 2 edges occur at the same time on PHA and PHB, which is unlikely to occur in real life, there is a report if the time difference between 2 edges on PHA, PHB is lower than a predefined value. This predefined value is configurable and corresponds

to  $(\text{MAXFILT}+1) * \text{tMCK}$  ns. After being filtered there is no reason to have 2 edges closer than  $(\text{MAXFILT}+1) * \text{tMCK}$  ns under normal mode of operation. In the instance an anomaly occurs, a quadrature error is reported on QERR flag on TC\_QISR register.

**Figure 38-21. Quadrature Error Detection**



MAXFILT must be tuned according to several factors such as the system clock frequency (MCK), type of rotary sensor and rotation speed to be achieved.

#### 38.6.16.4 Position and Rotation Measurement

When POSEN is set in TC\_BMR register, position is processed on channel 0 (by means of the PHA,PHB edge detections) and motor revolutions are accumulated in channel 1 timer/counter and can be read through TC\_CV0 and/or TC\_CV1 register if the IDX signal is provided on TIOB1 input.

Channel 0 and 1 must be configured in capture mode (WAVE = 0 in TC\_CMRO).

In parallel, the number of edges are accumulated on timer/counter channel 0 and can be read on the TC\_CV0 register.

Therefore, the accurate position can be read on both TC\_CV registers and concatenated to form a 32-bit word.

The timer/counter channel 0 is cleared for each increment of IDX count value.

Depending on the quadrature signals, the direction is decoded and allows to count up or down in timer/counter channels 0 and 1. The direction status is reported on TC\_QISR register.

#### 38.6.16.5 Speed Measurement

When SPEEDEN is set in TC\_BMR register, the speed measure is enabled on channel 0.

A time base must be defined on channel 2 by writing the TC\_RC2 period register. Channel 2 must be configured in waveform mode (WAVE bit field set) in TC\_CMRR2 register. WAVSEL bit field must be defined with 0x10 to clear the counter by comparison and matching with TC\_RC value. ACPC field must be defined at 0x11 to toggle TIOA output.

This time base is automatically fed back to TIOA of channel 0 when QDEN and SPEEDEN are set.

Channel 0 must be configured in capture mode (WAVE = 0 in TC\_CMRR0). ABETRG bit field of TC\_CMRR0 must be configured at 1 to get TIOA as a trigger for this channel.

EDGTRG can be set to 0x01, to clear the counter on a rising edge of the TIOA signal and LDRA field must be set accordingly to 0x01, to load TC\_RA0 at the same time as the counter is cleared (LDRB must be set to 0x01). As a consequence, at the end of each time base period the differentiation required for the speed calculation is performed.

The process must be started by configuring the TC\_CR register with CLKEN and SWTRG.

The speed can be read on TC\_RA0 register in TC\_CMRR0.

Channel 1 can still be used to count the number of revolutions of the motor.

### 38.6.17 2-bit Gray Up/Down Counter for Stepper Motor

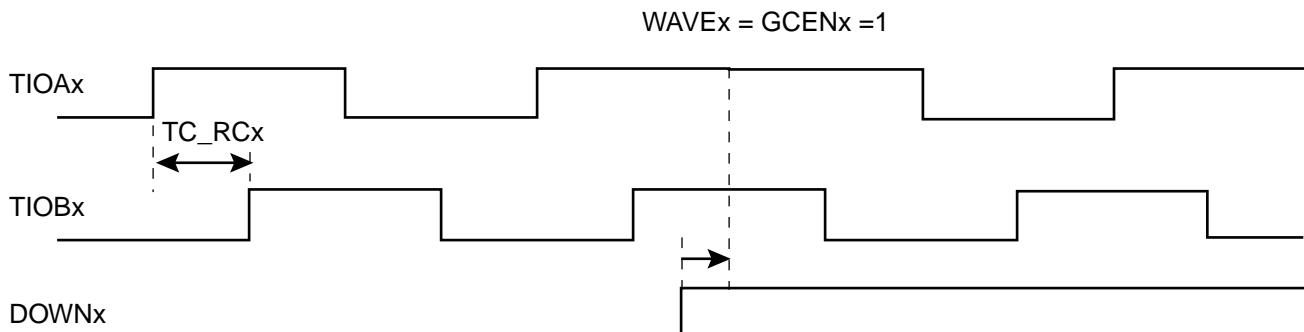
Each channel can be independently configured to generate a 2-bit gray count waveform on corresponding TIOA,TIOB outputs by means of GCEN bit in TC\_SMMRx registers.

Up or Down count can be defined by writing bit DOWN in TC\_SMMRx registers.

It is mandatory to configure the channel in WAVE mode in TC\_CMRR register.

The period of the counters can be programmed on TC\_RCx registers.

**Figure 38-22. 2-bit Gray Up/Down Counter.**



### 38.6.18 Write Protection System

In order to bring security to the Timer Counter, a write protection system has been implemented.

The write protection mode prevent the write of TC\_BMR, TC\_FMR, TC\_CMRRx, TC\_SMMRx, TC\_RAx, TC\_RBx, TC\_RCx registers. When this mode is enabled and one of the protected registers write, the register write request canceled.

Due to the nature of the write protection feature, enabling and disabling the write protection mode requires the use of a security code. Thus when enabling or disabling the write protection mode the WPKEY field of the TC\_WPMR register must be filled with the "TIM" ASCII code (corresponding to 0x54494D) otherwise the register write will be canceled.

### 38.6.19 Fault Mode

At anytime, the TC\_RCx registers can be used to perform a comparison on the respective current channel counter value (TC\_CVx) with the value of TC\_RCx register.

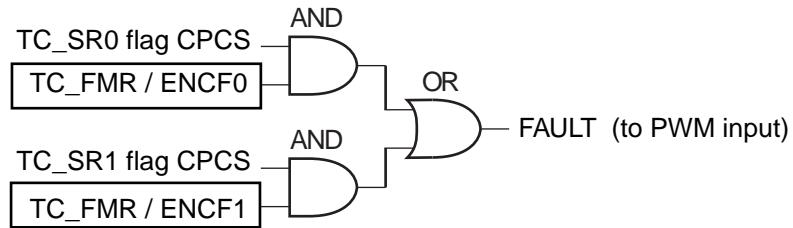
The CPCSx flags can be set accordingly and an interrupt can be generated.

This interrupt is processed but requires an unpredictable amount of time to be achieve the required action.

It is possible to trigger the FAULT output of the TIMER1 with CPCS from TC\_SR0 register and/or CPCS from TC\_SR1 register. Each source can be independently enabled/disabled by means of TC\_FMR register.

This can be useful to detect an overflow on speed and/or position when QDEC is processed and to act immediately by using the FAULT output.

**Figure 38-23. Fault Output Generation**



## 38.7 Timer Counter (TC) User Interface

Table 38-5. Register Mapping

Offset <sup>(1)</sup>	Register	Name	Access	Reset
0x00 + channel * 0x40 + 0x00	Channel Control Register	TC_CCR	Write-only	–
0x00 + channel * 0x40 + 0x04	Channel Mode Register	TC_CMRR	Read-write	0
0x00 + channel * 0x40 + 0x08	Stepper Motor Mode Register	TC_SMMR	Read-write	0
0x00 + channel * 0x40 + 0x0C	Register AB	TC_RAB	Read-only	0
0x00 + channel * 0x40 + 0x10	Counter Value	TC_CV	Read-only	0
0x00 + channel * 0x40 + 0x14	Register A	TC_RA	Read-write <sup>(2)</sup>	0
0x00 + channel * 0x40 + 0x18	Register B	TC_RB	Read-write <sup>(2)</sup>	0
0x00 + channel * 0x40 + 0x1C	Register C	TC_RC	Read-write	0
0x00 + channel * 0x40 + 0x20	Status Register	TC_SR	Read-only	0
0x00 + channel * 0x40 + 0x24	Interrupt Enable Register	TC_IER	Write-only	–
0x00 + channel * 0x40 + 0x28	Interrupt Disable Register	TC_IDR	Write-only	–
0x00 + channel * 0x40 + 0x2C	Interrupt Mask Register	TC_IMR	Read-only	0
0x00 + channel * 0x40 + 0x30	Extended Mode Register	TC_EMRR	Read-write	0
0xC0	Block Control Register	TC_BCR	Write-only	–
0xC4	Block Mode Register	TC_BMR	Read-write	0
0xC8	QDEC Interrupt Enable Register	TC_QIER	Write-only	–
0xCC	QDEC Interrupt Disable Register	TC_QIDR	Write-only	–
0xD0	QDEC Interrupt Mask Register	TC_QIMR	Read-only	0
0xD4	QDEC Interrupt Status Register	TC_QISR	Read-only	0
0xD8	Fault Mode Register	TC_FMR	Read-write	0
0xE4	Write Protect Mode Register	TC_WPMR	Read-write	0
0xE8 - 0xFC	Reserved	–	–	–
0x100 - 0x1A4	Reserved for PDC Registers	–	–	–

Notes: 1. Channel index ranges from 0 to 2.

2. Read-only if WAVE = 0

### 38.7.1 TC Channel Control Register

**Name:** TC\_CCRx [x=0..2]

**Address:** 0x40090000 (0)[0], 0x40090040 (0)[1], 0x40090080 (0)[2], 0x40094000 (1)[0], 0x40094040 (1)[1], 0x40094080 (1)[2], 0x40098000 (2)[0], 0x40098040 (2)[1], 0x40098080 (2)[2]

**Access:** Write-only

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
-	-	-	-	-	SWTRG	CLKDIS	CLKEN

- CLKEN: Counter Clock Enable Command**

0 = No effect.

1 = Enables the clock if CLKDIS is not 1.

- CLKDIS: Counter Clock Disable Command**

0 = No effect.

1 = Disables the clock.

- SWTRG: Software Trigger Command**

0 = No effect.

1 = A software trigger is performed: the counter is reset and the clock is started.

### 38.7.2 TC Channel Mode Register: Capture Mode

**Name:** TC\_CMRx [x=0..2] (WAVE = 0)

**Address:** 0x40090004 (0)[0], 0x40090044 (0)[1], 0x40090084 (0)[2], 0x40094004 (1)[0], 0x40094044 (1)[1], 0x40094084 (1)[2], 0x40098004 (2)[0], 0x40098044 (2)[1], 0x40098084 (2)[2]

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-		SBSMPLR		LDRB		LDRA	
15	14	13	12	11	10	9	8
WAVE	CPCTRG	-	-	-	ABETRG	ETRGEDG	
7	6	5	4	3	2	1	0
LDBDIS	LDBSTOP	BURST		CLKI		TCCLKS	

This register can only be written if the WPEN bit is cleared in “[TC Write Protect Mode Register](#)” on page 952

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: TCLK1
1	TIMER_CLOCK2	Clock selected: TCLK2
2	TIMER_CLOCK3	Clock selected: TCLK3
3	TIMER_CLOCK4	Clock selected: TCLK4
4	TIMER_CLOCK5	Clock selected: TCLK5
5	XC0	Clock selected: XC0
6	XC1	Clock selected: XC1
7	XC2	Clock selected: XC2

To operate at maximum clock frequency (MCK) please refer to “[TC Extended Mode Register](#)” on page 943.

- **CLKI: Clock Invert**

0 = Counter is incremented on rising edge of the clock.

1 = Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	XC0	XC0 is ANDed with the selected clock.
2	XC1	XC1 is ANDed with the selected clock.
3	XC2	XC2 is ANDed with the selected clock.

- **LDBSTOP: Counter Clock Stopped with RB Loading**

0 = Counter clock is not stopped when RB loading occurs.

1 = Counter clock is stopped when RB loading occurs.

- **LDBDIS: Counter Clock Disable with RB Loading**

0 = Counter clock is not disabled when RB loading occurs.

1 = Counter clock is disabled when RB loading occurs.

- **ETRGEDG: External Trigger Edge Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	RISING	Rising edge
2	FALLING	Falling edge
3	EDGE	Each edge

- **ABETRG: TIOA or TIOB External Trigger Selection**

0 = TIOB is used as an external trigger.

1 = TIOA is used as an external trigger.

- **CPCTRG: RC Compare Trigger Enable**

0 = RC Compare has no effect on the counter and its clock.

1 = RC Compare resets the counter and starts the counter clock.

- **WAVE: Waveform Mode**

0 = Capture Mode is enabled.

1 = Capture Mode is disabled (Waveform Mode is enabled).

- **LDRA: RA Loading Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge of TIOA
2	FALLING	Falling edge of TIOA
3	EDGE	Each edge of TIOA

- **LDRB: RB Loading Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge of TIOA
2	FALLING	Falling edge of TIOA
3	EDGE	Each edge of TIOA

- **SBSMPLR: Loading Edge Subsampling Ratio**

Value	Name	Description
0	ONE	Load a Capture Register each selected edge
1	HALF	Load a Capture Register every 2 selected edges
2	FOURTH	Load a Capture Register every 4 selected edges
3	EIGHTH	Load a Capture Register every 8 selected edges
4	SIXTEENTH	Load a Capture Register every 16 selected edges

### 38.7.3 TC Channel Mode Register: Waveform Mode

**Name:** TC\_CMRx [x=0..2] (WAVE = 1)

**Access:** Read-write

31	30	29	28	27	26	25	24
BSWTRG		BEEVT		BCPC		BCPB	
23	22	21	20	19	18	17	16
ASWTRG		AEEVT		ACPC		ACPA	
15	14	13	12	11	10	9	8
WAVE	WAVSEL		ENETRG	EEVT		EEVTEDG	
7	6	5	4	3	2	1	0
CPCDIS	CPCSTOP	BURST		CLKI		TCCLKS	

This register can only be written if the WPEN bit is cleared in “[TC Write Protect Mode Register](#)” on page 952

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: TCLK1
1	TIMER_CLOCK2	Clock selected: TCLK2
2	TIMER_CLOCK3	Clock selected: TCLK3
3	TIMER_CLOCK4	Clock selected: TCLK4
4	TIMER_CLOCK5	Clock selected: TCLK5
5	XC0	Clock selected: XC0
6	XC1	Clock selected: XC1
7	XC2	Clock selected: XC2

To operate at maximum clock frequency (MCK) please refer to “[TC Extended Mode Register](#)” on page 943.

- **CLKI: Clock Invert**

0 = Counter is incremented on rising edge of the clock.

1 = Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	XC0	XC0 is ANDed with the selected clock.
2	XC1	XC1 is ANDed with the selected clock.
3	XC2	XC2 is ANDed with the selected clock.

- **CPCSTOP: Counter Clock Stopped with RC Compare**

0 = Counter clock is not stopped when counter reaches RC.

1 = Counter clock is stopped when counter reaches RC.

- **CPCDIS: Counter Clock Disable with RC Compare**

0 = Counter clock is not disabled when counter reaches RC.

1 = Counter clock is disabled when counter reaches RC.

- **EEVTEDG: External Event Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge
2	FALLING	Falling edge
3	EDGE	Each edge

- **EEVT: External Event Selection**

Signal selected as external event.

Value	Name	Description	TIOB Direction
0	TIOB	TIOB <sup>(1)</sup>	input
1	XC0	XC0	output
2	XC1	XC1	output
3	XC2	XC2	output

Note: 1. If TIOB is chosen as the external event signal, it is configured as an input and no longer generates waveforms and subsequently no IRQs.

- **ENETRG: External Event Trigger Enable**

0 = The external event has no effect on the counter and its clock.

1 = The external event resets the counter and starts the counter clock.

Note: Whatever the value programmed in ENETRG, the selected external event only controls the TIOA output and TIOB if not used as input (trigger event input or other input used).

- **WAVSEL: Waveform Selection**

Value	Name	Description
0	UP	UP mode without automatic trigger on RC Compare
1	UPDOWN	UPDOWN mode without automatic trigger on RC Compare
2	UP_RC	UP mode with automatic trigger on RC Compare
3	UPDOWN_RC	UPDOWN mode with automatic trigger on RC Compare

- **WAVE: Waveform Mode**

0 = Waveform Mode is disabled (Capture Mode is enabled).

1 = Waveform Mode is enabled.

- **ACPA: RA Compare Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ACPC: RC Compare Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **AEEVT: External Event Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ASWTRG: Software Trigger Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPB: RB Compare Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPC: RC Compare Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BEEVT: External Event Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BSWTRG: Software Trigger Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

### 38.7.4 TC Stepper Motor Mode Register

**Name:** TC\_SMMRx [x=0..2]

**Address:** 0x40090008 (0)[0], 0x40090048 (0)[1], 0x40090088 (0)[2], 0x40094008 (1)[0], 0x40094048 (1)[1], 0x40094088 (1)[2], 0x40098008 (2)[0], 0x40098048 (2)[1], 0x40098088 (2)[2]

**Access:** Read-write

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
-	-	-	-	-	-	DOWN	GCEN

This register can only be written if the WPEN bit is cleared in “TC Write Protect Mode Register” on page 952

- **GCEN: Gray Count Enable**

0 = TIOAx [x=0..2] and TIOBx [x=0..2] are driven by internal counter of channel x.

1 = TIOAx [x=0..2] and TIOBx [x=0..2] are driven by a 2-bit gray counter.

- **DOWN: DOWN Count**

0 = Up counter.

1 = Down counter.

### 38.7.5 TC Register AB

**Name:** TC\_RABx [x=0..2]

**Address:** 0x4009000C (0)[0], 0x4009004C (0)[1], 0x4009008C (0)[2], 0x4009400C (1)[0], 0x4009404C (1)[1], 0x4009408C (1)[2], 0x4009800C (2)[0], 0x4009804C (2)[1], 0x4009808C (2)[2]

**Access:** Read-only

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

- **RAB: Register A or Register B**

RAB contains the next unread capture Register A or Register B value in real time. It is usually read by the **DMA** after a request due to a valid load edge on TIOA.

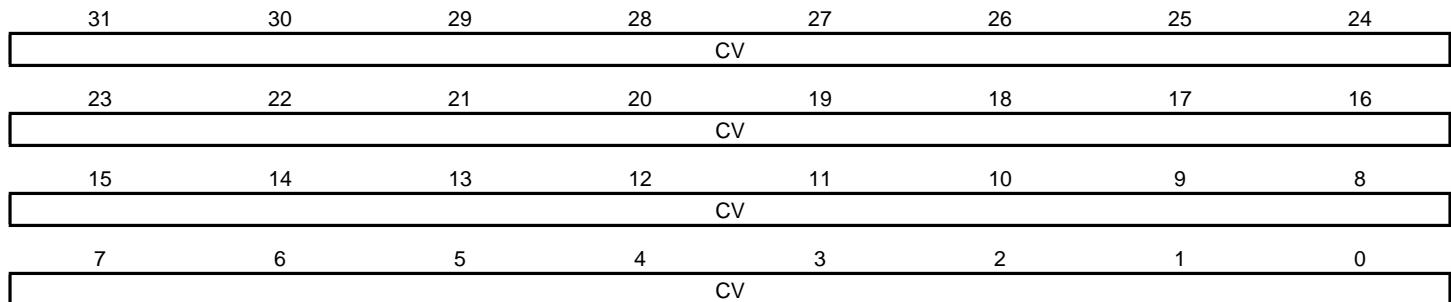
When DMA is used, the RAB register address must be configured as source address of the transfer.

### 38.7.6 TC Counter Value Register

**Name:** TC\_CVx [x=0..2]

**Address:** 0x40090010 (0)[0], 0x40090050 (0)[1], 0x40090090 (0)[2], 0x40094010 (1)[0], 0x40094050 (1)[1], 0x40094090 (1)[2], 0x40098010 (2)[0], 0x40098050 (2)[1], 0x40098090 (2)[2]

**Access:** Read-only



- **CV: Counter Value**

CV contains the counter value in real time.

### 38.7.7 TC Register A

**Name:** TC\_RAx [x=0..2]

**Address:** 0x40090014 (0)[0], 0x40090054 (0)[1], 0x40090094 (0)[2], 0x40094014 (1)[0], 0x40094054 (1)[1], 0x40094094 (1)[2], 0x40098014 (2)[0], 0x40098054 (2)[1], 0x40098094 (2)[2]

**Access:** Read-only if WAVE = 0, Read-write if WAVE = 1

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

This register can only be written if the WPEN bit is cleared in “TC Write Protect Mode Register” on page 952

- **RA: Register A**

RA contains the Register A value in real time.

### 38.7.8 TC Register B

**Name:** TC\_RBx [x=0..2]

**Address:** 0x40090018 (0)[0], 0x40090058 (0)[1], 0x40090098 (0)[2], 0x40094018 (1)[0], 0x40094058 (1)[1], 0x40094098 (1)[2], 0x40098018 (2)[0], 0x40098058 (2)[1], 0x40098098 (2)[2]

**Access:** Read-only if WAVE = 0, Read-write if WAVE = 1

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

This register can only be written if the WPEN bit is cleared in “TC Write Protect Mode Register” on page 952

- **RB: Register B**

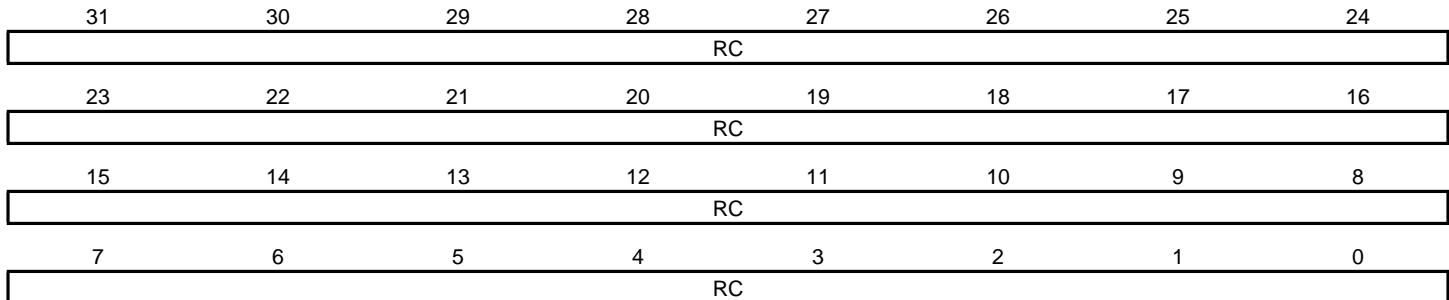
RB contains the Register B value in real time.

### 38.7.9 TC Register C

**Name:** TC\_RCx [x=0..2]

**Address:** 0x4009001C (0)[0], 0x4009005C (0)[1], 0x4009009C (0)[2], 0x4009401C (1)[0], 0x4009405C (1)[1], 0x4009409C (1)[2], 0x4009801C (2)[0], 0x4009805C (2)[1], 0x4009809C (2)[2]

**Access:** Read-write



This register can only be written if the WPEN bit is cleared in “[TC Write Protect Mode Register](#)” on page 952

- **RC: Register C**

RC contains the Register C value in real time.

### 38.7.10 TC Status Register

**Name:** TC\_SR<sub>x</sub> [x=0..2]

**Address:** 0x40090020 (0)[0], 0x40090060 (0)[1], 0x400900A0 (0)[2], 0x40094020 (1)[0], 0x40094060 (1)[1], 0x400940A0 (1)[2], 0x40098020 (2)[0], 0x40098060 (2)[1], 0x400980A0 (2)[2]

**Access:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	MTIOB	MTIOA	CLKSTA
15	14	13	12	11	10	9	8
-	-	-	-	-	-	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow Status**

0 = No counter overflow has occurred since the last read of the Status Register.

1 = A counter overflow has occurred since the last read of the Status Register.

- **LOVRS: Load Overrun Status**

0 = Load overrun has not occurred since the last read of the Status Register or WAVE = 1.

1 = RA or RB have been loaded at least twice without any read of the corresponding register since the last read of the Status Register, if WAVE = 0.

- **CPAS: RA Compare Status**

0 = RA Compare has not occurred since the last read of the Status Register or WAVE = 0.

1 = RA Compare has occurred since the last read of the Status Register, if WAVE = 1.

- **CPBS: RB Compare Status**

0 = RB Compare has not occurred since the last read of the Status Register or WAVE = 0.

1 = RB Compare has occurred since the last read of the Status Register, if WAVE = 1.

- **CPCS: RC Compare Status**

0 = RC Compare has not occurred since the last read of the Status Register.

1 = RC Compare has occurred since the last read of the Status Register.

- **LDRAS: RA Loading Status**

0 = RA Load has not occurred since the last read of the Status Register or WAVE = 1.

1 = RA Load has occurred since the last read of the Status Register, if WAVE = 0.

- **LDRBS: RB Loading Status**

0 = RB Load has not occurred since the last read of the Status Register or WAVE = 1.

1 = RB Load has occurred since the last read of the Status Register, if WAVE = 0.

- **ETRGS: External Trigger Status**

0 = External trigger has not occurred since the last read of the Status Register.

1 = External trigger has occurred since the last read of the Status Register.

- **ENDRX: End of Receiver Transfer**

0 = The Receive End of Transfer signal from the PDC channel is inactive.

1 = The Receive End of Transfer signal from the PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0 = The Receive Buffer Full signal from the PDC channel is inactive.

1 = The Receive Buffer Full signal from the PDC channel is active.

- **CLKSTA: Clock Enabling Status**

0 = Clock is disabled.

1 = Clock is enabled.

- **MTIOA: TIOA Mirror**

0 = TIOA is low. If WAVE = 0, this means that TIOA pin is low. If WAVE = 1, this means that TIOA is driven low.

1 = TIOA is high. If WAVE = 0, this means that TIOA pin is high. If WAVE = 1, this means that TIOA is driven high.

- **MTIOB: TIOB Mirror**

0 = TIOB is low. If WAVE = 0, this means that TIOB pin is low. If WAVE = 1, this means that TIOB is driven low.

1 = TIOB is high. If WAVE = 0, this means that TIOB pin is high. If WAVE = 1, this means that TIOB is driven high.

### 38.7.11 TC Interrupt Enable Register

**Name:** TC\_IERx [x=0..2]

**Address:** 0x40090024 (0)[0], 0x40090064 (0)[1], 0x400900A4 (0)[2], 0x40094024 (1)[0], 0x40094064 (1)[1], 0x400940A4 (1)[2], 0x40098024 (2)[0], 0x40098064 (2)[1], 0x400980A4 (2)[2]

**Access:** Write-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0 = No effect.

1 = Enables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0 = No effect.

1 = Enables the Load Overrun Interrupt.

- **CPAS: RA Compare**

0 = No effect.

1 = Enables the RA Compare Interrupt.

- **CPBS: RB Compare**

0 = No effect.

1 = Enables the RB Compare Interrupt.

- **CPCS: RC Compare**

0 = No effect.

1 = Enables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0 = No effect.

1 = Enables the RA Load Interrupt.

- **LDRBS: RB Loading**

0 = No effect.

1 = Enables the RB Load Interrupt.

- **ETRGS: External Trigger**

0 = No effect.

1 = Enables the External Trigger Interrupt.

- **ENDRX: End of Receiver Transfer**

0 = No effect.

1 = Enables the PDC Receive End of Transfer Interrupt.

- **RXBUFF: Reception Buffer Full**

0 = No effect.

1 = Enables the PDC Receive Buffer Full Interrupt.

### 38.7.12 TC Interrupt Disable Register

**Name:** TC\_IDRx [x=0..2]

**Address:** 0x40090028 (0)[0], 0x40090068 (0)[1], 0x400900A8 (0)[2], 0x40094028 (1)[0], 0x40094068 (1)[1], 0x400940A8 (1)[2], 0x40098028 (2)[0], 0x40098068 (2)[1], 0x400980A8 (2)[2]

**Access:** Write-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0 = No effect.

1 = Disables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0 = No effect.

1 = Disables the Load Overrun Interrupt (if WAVE = 0).

- **CPAS: RA Compare**

0 = No effect.

1 = Disables the RA Compare Interrupt (if WAVE = 1).

- **CPBS: RB Compare**

0 = No effect.

1 = Disables the RB Compare Interrupt (if WAVE = 1).

- **CPCS: RC Compare**

0 = No effect.

1 = Disables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0 = No effect.

1 = Disables the RA Load Interrupt (if WAVE = 0).

- **LDRBS: RB Loading**

0 = No effect.

1 = Disables the RB Load Interrupt (if WAVE = 0).

- **ETRGS: External Trigger**

0 = No effect.

1 = Disables the External Trigger Interrupt.

- **ENDRX: End of Receiver Transfer**

0 = No effect.

1 = Disables the PDC Receive End of Transfer Interrupt.

- **RXBUFF: Reception Buffer Full**

0 = No effect.

1 = Disables the PDC Receive Buffer Full Interrupt.

### 38.7.13 TC Interrupt Mask Register

**Name:** TC\_IMRx [x=0..2]

**Address:** 0x4009002C (0)[0], 0x4009006C (0)[1], 0x400900AC (0)[2], 0x4009402C (1)[0], 0x4009406C (1)[1], 0x400940AC (1)[2], 0x4009802C (2)[0], 0x4009806C (2)[1], 0x400980AC (2)[2]

**Access:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0 = The Counter Overflow Interrupt is disabled.

1 = The Counter Overflow Interrupt is enabled.

- **LOVRS: Load Overrun**

0 = The Load Overrun Interrupt is disabled.

1 = The Load Overrun Interrupt is enabled.

- **CPAS: RA Compare**

0 = The RA Compare Interrupt is disabled.

1 = The RA Compare Interrupt is enabled.

- **CPBS: RB Compare**

0 = The RB Compare Interrupt is disabled.

1 = The RB Compare Interrupt is enabled.

- **CPCS: RC Compare**

0 = The RC Compare Interrupt is disabled.

1 = The RC Compare Interrupt is enabled.

- **LDRAS: RA Loading**

0 = The Load RA Interrupt is disabled.

1 = The Load RA Interrupt is enabled.

- **LDRBS: RB Loading**

0 = The Load RB Interrupt is disabled.

1 = The Load RB Interrupt is enabled.

- **ETRGS: External Trigger**

0 = The External Trigger Interrupt is disabled.

1 = The External Trigger Interrupt is enabled.

- **ENDRX: End of Receiver Transfer**

0 = The PDC Receive End of Transfer Interrupt is disabled.

1 = The PDC Receive End of Transfer Interrupt is enabled.

- **RXBUFF: Reception Buffer Full**

0 = The PDC Receive Buffer Full Interrupt is disabled.

1 = The PDC Receive Buffer Full Interrupt is enabled.

### 38.7.14 TC Extended Mode Register

**Name:** TC\_EMRx [x=0..2]

**Address:** 0x40090030 (0)[0], 0x40090070 (0)[1], 0x400900B0 (0)[2], 0x40094030 (1)[0], 0x40094070 (1)[1], 0x400940B0 (1)[2], 0x40098030 (2)[0], 0x40098070 (2)[1], 0x400980B0 (2)[2]

**Access:** Read-write

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

- **TRIGSRCA:** TRIGger SouRCe for input A

Value	Name	Description
0	EXTERNAL_TIOAx	the trigger/capture input A is driven by external pin TIOAx
1	PWMx	the trigger/capture input A is driven internally by PWMx

- **TRIGSRCB:** TRIGger SouRCe for input B

Value	Name	Description
0	EXTERNAL_TIOBx	the trigger/capture input B is driven by external pin TIOBx
1	PWMx	the trigger/capture input B is driven internally by PWMx

#### • **NODIVCLK: NO DIVided CLocK**

0 = The selected clock is defined by field TCCLKS in TC\_CMRx register.

1 = The selected clock is MCK and TCCLKS field (TC\_CMRx register) has no effect.

### 38.7.15 TC Block Control Register

**Name:** TC\_BCR

**Address:** 0x400900C0 (0), 0x400940C0 (1), 0x400980C0 (2)

**Access:** Write-only

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
-	-	-	-	-	-	-	SYNC

- **SYNC: Synchro Command**

0 = No effect.

1 = Asserts the SYNC signal which generates a software trigger simultaneously for each of the channels.

### 38.7.16 TC Block Mode Register

**Name:** TC\_BMR

**Address:** 0x400900C4 (0), 0x400940C4 (1), 0x400980C4 (2)

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	MAXFILT	
23	22	21	20	19	18	17	16
	MAXFILT		FILTER	–	IDXPHB	SWAP	
15	14	13	12	11	10	9	8
INVIDX	INVB	INVA	EDGPHA	QDTRANS	SPEEDEN	POSEN	QDEN
7	6	5	4	3	2	1	0
–	–	TC2XC2S		TC1XC1S		TC0XC0S	

This register can only be written if the WPEN bit is cleared in “[TC Write Protect Mode Register](#)” on page 952.

- **TC0XC0S: External Clock Signal 0 Selection**

Value	Name	Description
0	TCLK0	Signal connected to XC0: TCLK0
1	–	Reserved
2	TIOA1	Signal connected to XC0: TIOA1
3	TIOA2	Signal connected to XC0: TIOA2

- **TC1XC1S: External Clock Signal 1 Selection**

Value	Name	Description
0	TCLK1	Signal connected to XC1: TCLK1
1	–	Reserved
2	TIOA0	Signal connected to XC1: TIOA0
3	TIOA2	Signal connected to XC1: TIOA2

- **TC2XC2S: External Clock Signal 2 Selection**

Value	Name	Description
0	TCLK2	Signal connected to XC2: TCLK2
1	–	Reserved
2	TIOA1	Signal connected to XC2: TIOA1
3	TIOA2	Signal connected to XC2: TIOA2

- **QDEN: Quadrature Decoder ENabled**

0 = Disabled.

1 = Enables the quadrature decoder logic (filter, edge detection and quadrature decoding).

Quadrature decoding (direction change) can be disabled using QDTRANS bit.

One of the POSEN or SPEEDEN bits must be also enabled.

- **POSEN: POSition ENabled**

0 = Disable position.

1 = Enables the position measure on channel 0 and 1.

- **SPEEDEN: SPEED ENabled**

0 = Disabled.

1 = Enables the speed measure on channel 0, the time base being provided by channel 2.

- **QDTRANS: Quadrature Decoding TRANSparent**

0 = Full quadrature decoding logic is active (direction change detected).

1 = Quadrature decoding logic is inactive (direction change inactive) but input filtering and edge detection are performed.

- **EDGPHA: EDGe on PHA count mode**

0 = Edges are detected on PHA only.

1 = Edges are detected on both PHA and PHB.

- **INVA: INVerted phA**

0 = PHA (TIOA0) is directly driving quadrature decoder logic.

1 = PHA is inverted before driving quadrature decoder logic.

- **INVB: INVerted phB**

0 = PHB (TIOB0) is directly driving quadrature decoder logic.

1 = PHB is inverted before driving quadrature decoder logic.

- **SWAP: SWAP PHA and PHB**

0 = No swap between PHA and PHB.

1 = Swap PHA and PHB internally, prior to driving quadrature decoder logic.

- **INVIDX: INVerted InDeX**

0 = IDX (TIOA1) is directly driving quadrature logic.

1 = IDX is inverted before driving quadrature logic.

- **IDXPHB: InDeX pin is PHB pin**

0 = IDX pin of the rotary sensor must drive TIOA1.

1 = IDX pin of the rotary sensor must drive TIOB0.

- **FILTER:**

0 = IDX,PHA, PHB input pins are not filtered.

1 = IDX,PHA, PHB input pins are filtered using MAXFILT value.

- **MAXFILT: MAXimum FILTer**

1.. 63: Defines the filtering capabilities.

Pulses with a period shorter than MAXFILT+1 MCK clock cycles are discarded.

### 38.7.17 TC QDEC Interrupt Enable Register

**Name:** TC\_QIER

**Address:** 0x400900C8 (0), 0x400940C8 (1), 0x400980C8 (2)

**Access:** Write-only

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
-	-	-	-	-	QERR	DIRCHG	IDX

- **IDX: InDeX**

0 = No effect.

1 = Enables the interrupt when a rising edge occurs on IDX input.

- **DIRCHG: DIRection CHanGe**

0 = No effect.

1 = Enables the interrupt when a change on rotation direction is detected.

- **QERR: Quadrature ERRor**

0 = No effect.

1 = Enables the interrupt when a quadrature error occurs on PHA,PHB.

### 38.7.18 TC QDEC Interrupt Disable Register

**Name:** TC\_QIDR

**Address:** 0x400900CC (0), 0x400940CC (1), 0x400980CC (2)

**Access:** Write-only

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
-	-	-	-	-	QERR	DIRCHG	IDX

- **IDX: InDeX**

0 = No effect.

1 = Disables the interrupt when a rising edge occurs on IDX input.

- **DIRCHG: DIRection CHanGe**

0 = No effect.

1 = Disables the interrupt when a change on rotation direction is detected.

- **QERR: Quadrature ERRor**

0 = No effect.

1 = Disables the interrupt when a quadrature error occurs on PHA, PHB.

### 38.7.19 TC QDEC Interrupt Mask Register

**Name:** TC\_QIMR

**Address:** 0x400900D0 (0), 0x400940D0 (1), 0x400980D0 (2)

**Access:** Read-only

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
-	-	-	-	-	QERR	DIRCHG	IDX

- **IDX: InDeX**

0 = The interrupt on IDX input is disabled.

1 = The interrupt on IDX input is enabled.

- **DIRCHG: DIRection CHanGe**

0 = The interrupt on rotation direction change is disabled.

1 = The interrupt on rotation direction change is enabled.

- **QERR: Quadrature ERRor**

0 = The interrupt on quadrature error is disabled.

1 = The interrupt on quadrature error is enabled.

### 38.7.20 TC QDEC Interrupt Status Register

**Name:** TC\_QISR

**Address:** 0x400900D4 (0), 0x400940D4 (1), 0x400980D4 (2)

**Access:** Read-only

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

- **IDX: InDeX**

0 = No Index input change since the last read of TC\_QISR.

1 = The IDX input has changed since the last read of TC\_QISR.

- **DIRCHG: DIRection CHanGe**

0 = No change on rotation direction since the last read of TC\_QISR.

1 = The rotation direction changed since the last read of TC\_QISR.

- **QERR: Quadrature ERRoR**

0 = No quadrature error since the last read of TC\_QISR.

1 = A quadrature error occurred since the last read of TC\_QISR.

- **DIR: DIRection**

Returns an image of the actual rotation direction.

### 38.7.21 TC Fault Mode Register

**Name:** TC\_FMR

**Address:** 0x400900D8 (0), 0x400940D8 (1), 0x400980D8 (2)

**Access:** Read-write

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
—	—	—	—	—	—	ENCF1	ENCF0

This register can only be written if the WPEN bit is cleared in “TC Write Protect Mode Register” on page 952

- **ENCF0: ENable Compare Fault Channel 0**

0 = Disables the FAULT output source (CPCS flag) from channel 0.

1 = Enables the FAULT output source (CPCS flag) from channel 0.

- **ENCF1: ENable Compare Fault Channel 1**

0 = Disables the FAULT output source (CPCS flag) from channel 1.

1 = Enables the FAULT output source (CPCS flag) from channel 1.

### 38.7.22 TC Write Protect Mode Register

**Name:** TC\_WPMR

**Address:** 0x400900E4 (0), 0x400940E4 (1), 0x400980E4 (2)

**Access:** Read-write

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x54494D ("TIM" in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x54494D ("TIM" in ASCII).

Protects the registers:

["TC Block Mode Register"](#)

["TC Channel Mode Register: Capture Mode"](#)

["TC Channel Mode Register: Waveform Mode"](#)

["TC Fault Mode Register"](#)

["TC Stepper Motor Mode Register"](#)

["TC Register A"](#)

["TC Register B"](#)

["TC Register C"](#)

["TC Extended Mode Register"](#)

- **WPKEY: Write Protect KEY**

Value	Name	Description
0x54494D	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

## 39. High Speed MultiMedia Card Interface (HSMCI)

### 39.1 Description

The High Speed Multimedia Card Interface (HSMCI) supports the MultiMedia Card (MMC) Specification V4.3, the SD Memory Card Specification V2.0, the SDIO V2.0 specification and CE-ATA V1.1.

The HSMCI includes a command register, response registers, data registers, timeout counters and error detection logic that automatically handle the transmission of commands and, when required, the reception of the associated responses and data with a limited processor overhead.

The HSMCI supports stream, block and multi block data read and write, and is compatible with the Peripheral DMA Controller (PDC) Channels, minimizing processor intervention for large buffer transfers.

The HSMCI operates at a rate of up to Master Clock divided by 2 and supports the interfacing of 1 slot(s). Each slot may be used to interface with a High Speed MultiMedia Card bus (up to 30 Cards) or with an SD Memory Card. Only one slot can be selected at a time (slots are multiplexed). A bit field in the SD Card Register performs this selection.

The SD Memory Card communication is based on a 9-pin interface (clock, command, four data and three power lines) and the High Speed MultiMedia Card on a 7-pin interface (clock, command, one data, three power lines and one reserved for future use).

The SD Memory Card interface also supports High Speed MultiMedia Card operations. The main differences between SD and High Speed MultiMedia Cards are the initialization process and the bus topology.

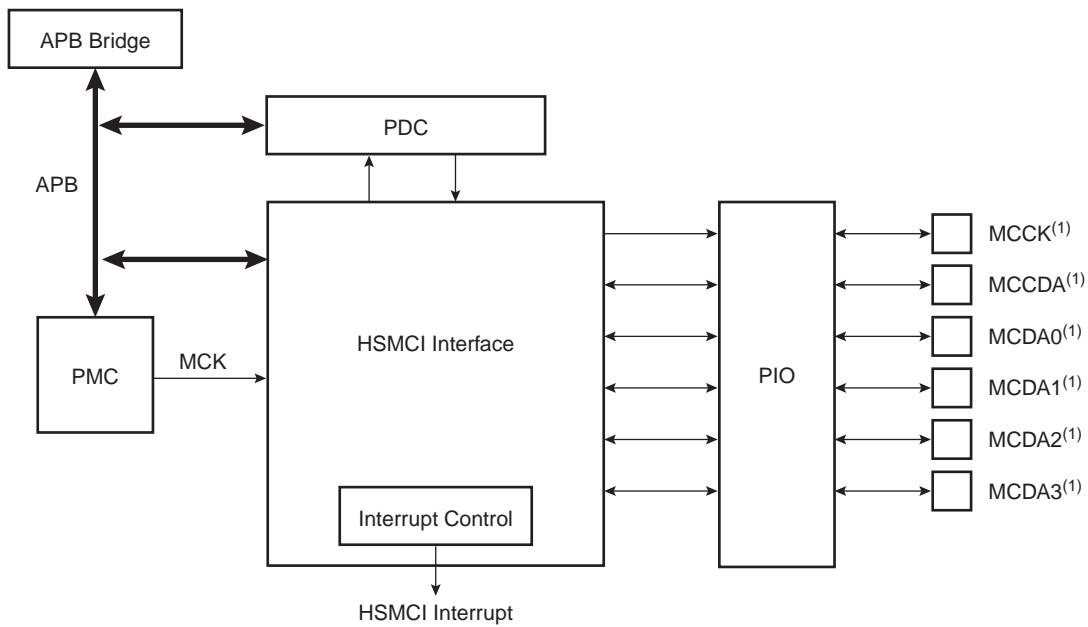
HSMCI fully supports CE-ATA Revision 1.1, built on the MMC System Specification v4.0. The module includes dedicated hardware to issue the command completion signal and capture the host command completion signal disable.

### 39.2 Embedded Characteristics

- Compatible with MultiMedia Card Specification Version 4.3
- Compatible with SD Memory Card Specification Version 2.0
- Compatible with SDIO Specification Version 2.0
- Compatible with CE-ATA Specification 1.1
- Cards Clock Rate Up to Master Clock Divided by 2
- Boot Operation Mode Support
- High Speed Mode Support
- Embedded Power Management to Slow Down Clock Rate When Not Used
- Supports 1 Multiplexed Slot(s)
  - Each Slot for either a High Speed MultiMedia Card Bus (Up to 30 Cards) or an SD Memory Card
- Support for Stream, Block and Multi-block Data Read and Write
- Supports Connection to Peripheral DMA Controller (PDC)
  - Minimizes Processor Intervention for Large Buffer Transfers
- Built in FIFO (from 16 to 256 bytes) with Large Memory Aperture Supporting Incremental Access
- Support for CE-ATA Completion Signal Disable Command
- Protection Against Unexpected Modification On-the-Fly of the Configuration Registers

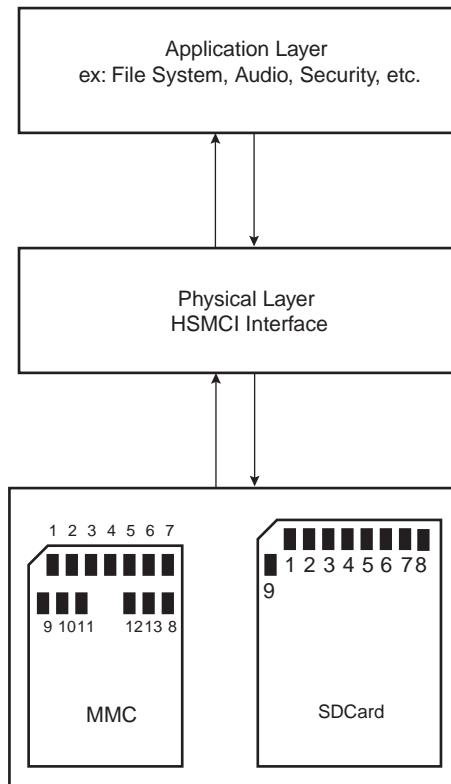
### 39.3 Block Diagram

Figure 39-1. Block Diagram



## 39.4 Application Block Diagram

Figure 39-2. Application Block Diagram



## 39.5 Pin Name List

Table 39-1. I/O Lines Description for 4-bit Configuration

Pin Name <sup>(2)</sup>	Pin Description	Type <sup>(1)</sup>	Comments
MCCDA	Command/response	I/O/PP/OD	CMD of an MMC or SDCard/SDIO
MCCK	Clock	I/O	CLK of an MMC or SD Card/SDIO
MCDA0 - MCDA3	Data 0..3 of Slot A	I/O/PP	DAT[0..3] of an MMC DAT[0..3] of an SD Card/SDIO

Notes:

1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.

2. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx\_CK, MCCDA to HSMCIx\_CDA, MCDAy to HSMCIx\_DAY.

## 39.6 Product Dependencies

### 39.6.1 I/O Lines

The pins used for interfacing the High Speed MultiMedia Cards or SD Cards are multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the peripheral functions to HSMCI pins.

Table 39-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
HSMCI	MCCDA	PA28	C
HSMCI	MCCK	PA29	C
HSMCI	MCDA0	PA30	C
HSMCI	MCDA1	PA31	C
HSMCI	MCDA2	PA26	C
HSMCI	MCDA3	PA27	C

### 39.6.2 Power Management

The HSMCI is clocked through the Power Management Controller (PMC), so the programmer must first configure the PMC to enable the HSMCI clock.

### 39.6.3 Interrupt

The HSMCI interface has an interrupt line connected to the interrupt controller.

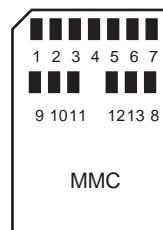
Handling the HSMCI interrupt requires programming the interrupt controller before configuring the HSMCI.

Table 39-3. Peripheral IDs

Instance	ID
HSMCI	16

## 39.7 Bus Topology

Figure 39-3. High Speed MultiMedia Memory Card Bus Topology



The High Speed MultiMedia Card communication is based on a 13-pin serial bus interface. It has three communication lines and four supply lines.

**Table 39-4. Bus Topology**

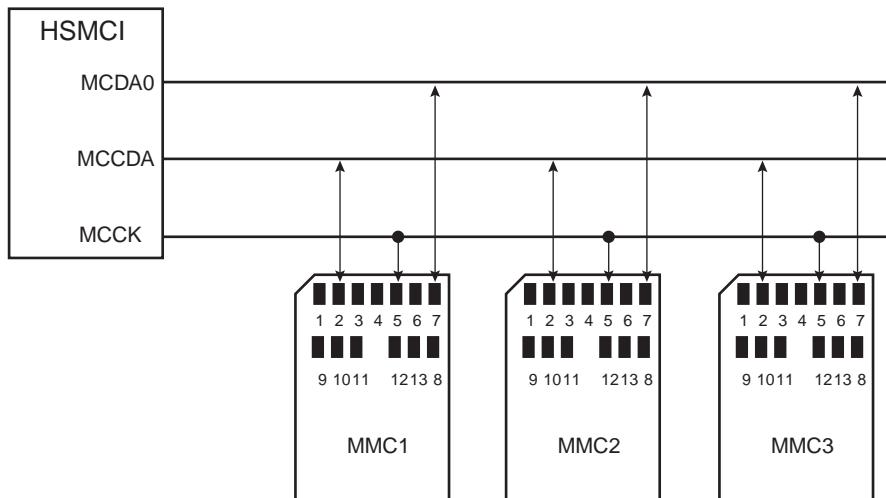
Pin Number	Name	Type <sup>(1)</sup>	Description	HSMCI Pin Name <sup>(2)</sup> (Slot z)
1	DAT[3]	I/O/PP	Data	MCDz3
2	CMD	I/O/PP/OD	Command/response	MCCDz
3	VSS1	S	Supply voltage ground	VSS
4	VDD	S	Supply voltage	VDD
5	CLK	I/O	Clock	MCCK
6	VSS2	S	Supply voltage ground	VSS
7	DAT[0]	I/O/PP	Data 0	MCDz0
8	DAT[1]	I/O/PP	Data 1	MCDz1
9	DAT[2]	I/O/PP	Data 2	MCDz2
10	DAT[4]	I/O/PP	Data 4	MCDz4
11	DAT[5]	I/O/PP	Data 5	MCDz5
12	DAT[6]	I/O/PP	Data 6	MCDz6
13	DAT[7]	I/O/PP	Data 7	MCDz7

Notes:

1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.

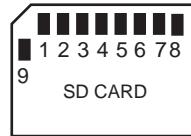
2. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx\_CK, MCCDA to HSMCIx\_CDA, MCDAy to HSMCIx\_DAy.

**Figure 39-4. MMC Bus Connections (One Slot)**



Note: When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx\_CK, MCCDA to HSMCIx\_CDA MCDAy to HSMCIx\_DAy.

**Figure 39-5. SD Memory Card Bus Topology**



The SD Memory Card bus includes the signals listed in [Table 39-5](#).

**Table 39-5. SD Memory Card Bus Signals**

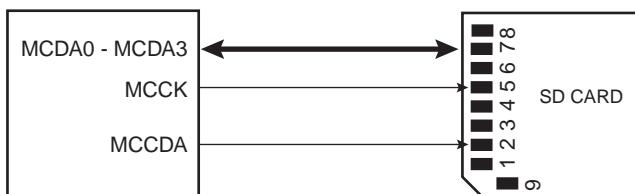
Pin Number	Name	Type <sup>(1)</sup>	Description	HSMCI Pin Name <sup>(2)</sup> (Slot z)
1	CD/DAT[3]	I/O/PP	Card detect/ Data line Bit 3	MCDz3
2	CMD	PP	Command/response	MCCDz
3	VSS1	S	Supply voltage ground	VSS
4	VDD	S	Supply voltage	VDD
5	CLK	I/O	Clock	MCCK
6	VSS2	S	Supply voltage ground	VSS
7	DAT[0]	I/O/PP	Data line Bit 0	MCDz0
8	DAT[1]	I/O/PP	Data line Bit 1 or Interrupt	MCDz1
9	DAT[2]	I/O/PP	Data line Bit 2	MCDz2

Notes:

1. I: input, O: output, PP: Push Pull, OD: Open Drain.

2. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx\_CK, MCCDA to HSMCIx\_CDA, MCDAy to HSMCIx\_DAy.

**Figure 39-6. SD Card Bus Connections with One Slot**



Note: When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx\_CK, MCCDA to HSMCIx\_CDA MCDAy to HSMCIx\_DAy.

When the HSMCI is configured to operate with SD memory cards, the width of the data bus can be selected in the HSMCI\_SDCR register. Clearing the SDCBUS bit in this register means that the width is one bit; setting it means that the width is four bits. In the case of High Speed MultiMedia cards, only the data line 0 is used. The other data lines can be used as independent PIOs.

## 39.8 High Speed MultiMedia Card Operations

After a power-on reset, the cards are initialized by a special message-based High Speed MultiMedia Card bus protocol. Each message is represented by one of the following tokens:

- Command: A command is a token that starts an operation. A command is sent from the host either to a single card (addressed command) or to all connected cards (broadcast command). A command is transferred serially on the CMD line.
- Response: A response is a token which is sent from an addressed card or (synchronously) from all connected cards to the host as an answer to a previously received command. A response is transferred serially on the CMD line.
- Data: Data can be transferred from the card to the host or vice versa. Data is transferred via the data line.

Card addressing is implemented using a session address assigned during the initialization phase by the bus controller to all currently connected cards. Their unique CID number identifies individual cards.

The structure of commands, responses and data blocks is described in the High Speed MultiMedia Card System Specification. See also [Table 39-6 on page 960](#).

High Speed MultiMedia Card bus data transfers are composed of these tokens.

There are different types of operations. Addressed operations always contain a command and a response token. In addition, some operations have a data token; the others transfer their information directly within the command or response structure. In this case, no data token is present in an operation. The bits on the DAT and the CMD lines are transferred synchronous to the clock HSMCI Clock.

Two types of data transfer commands are defined:

- Sequential commands: These commands initiate a continuous data stream. They are terminated only when a stop command follows on the CMD line. This mode reduces the command overhead to an absolute minimum.
- Block-oriented commands: These commands send a data block succeeded by CRC bits.

Both read and write operations allow either single or multiple block transmission. A multiple block transmission is terminated when a stop command follows on the CMD line similarly to the sequential read or when a multiple block transmission has a pre-defined block count ([See “Data Transfer Operation” on page 962](#).).

The HSMCI provides a set of registers to perform the entire range of High Speed MultiMedia Card operations.

### 39.8.1 Command - Response Operation

After reset, the HSMCI is disabled and becomes valid after setting the MCIEN bit in the HSMCI\_CR Control Register.

The PWSEN bit saves power by dividing the HSMCI clock by  $2^{PWSDIV} + 1$  when the bus is inactive.

The two bits, RDPROOF and WRPROOF in the HSMCI Mode Register (HSMCI\_MR) allow stopping the HSMCI Clock during read or write access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

All the timings for High Speed MultiMedia Card are defined in the High Speed MultiMedia Card System Specification.

The two bus modes (open drain and push/pull) needed to process all the operations are defined in the HSMCI Command Register. The HSMCI\_CMDR allows a command to be carried out.

For example, to perform an ALL\_SEND\_CID command:

CMD	Host Command				N <sub>ID</sub> Cycles				CID					
	S	T	Content	CRC	E	Z	*****	Z	S	T	Content	Z	Z	Z

The command ALL\_SEND\_CID and the fields and values for the HSMCI\_CMDR Control Register are described in [Table 39-6](#) and [Table 39-7](#).

**Table 39-6. ALL\_SEND\_CID Command Description**

CMD Index	Type	Argument	Resp	Abbreviation	Command Description
CMD2	bcr <sup>(1)</sup>	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the CMD line

Note: 1. bcr means broadcast command with response.

**Table 39-7. Fields and Values for HSMCI\_CMDR Command Register**

Field	Value
CMDNB (command number)	2 (CMD2)
RSPTYP (response type)	2 (R2: 136 bits response)
SPCMD (special command)	0 (not a special command)
OPCMD (open drain command)	1
MAXLAT (max latency for command to response)	0 (NID cycles ==> 5 cycles)
TRCMD (transfer command)	0 (No transfer)
TRDIR (transfer direction)	X (available only in transfer command)
TRTYP (transfer type)	X (available only in transfer command)
IOSPCMD (SDIO special command)	0 (not a special command)

The HSMCI\_ARGR contains the argument field of the command.

To send a command, the user must perform the following steps:

- Fill the argument register (HSMCI\_ARGR) with the command argument.
- Set the command register (HSMCI\_CMDR) (see [Table 39-7](#)).

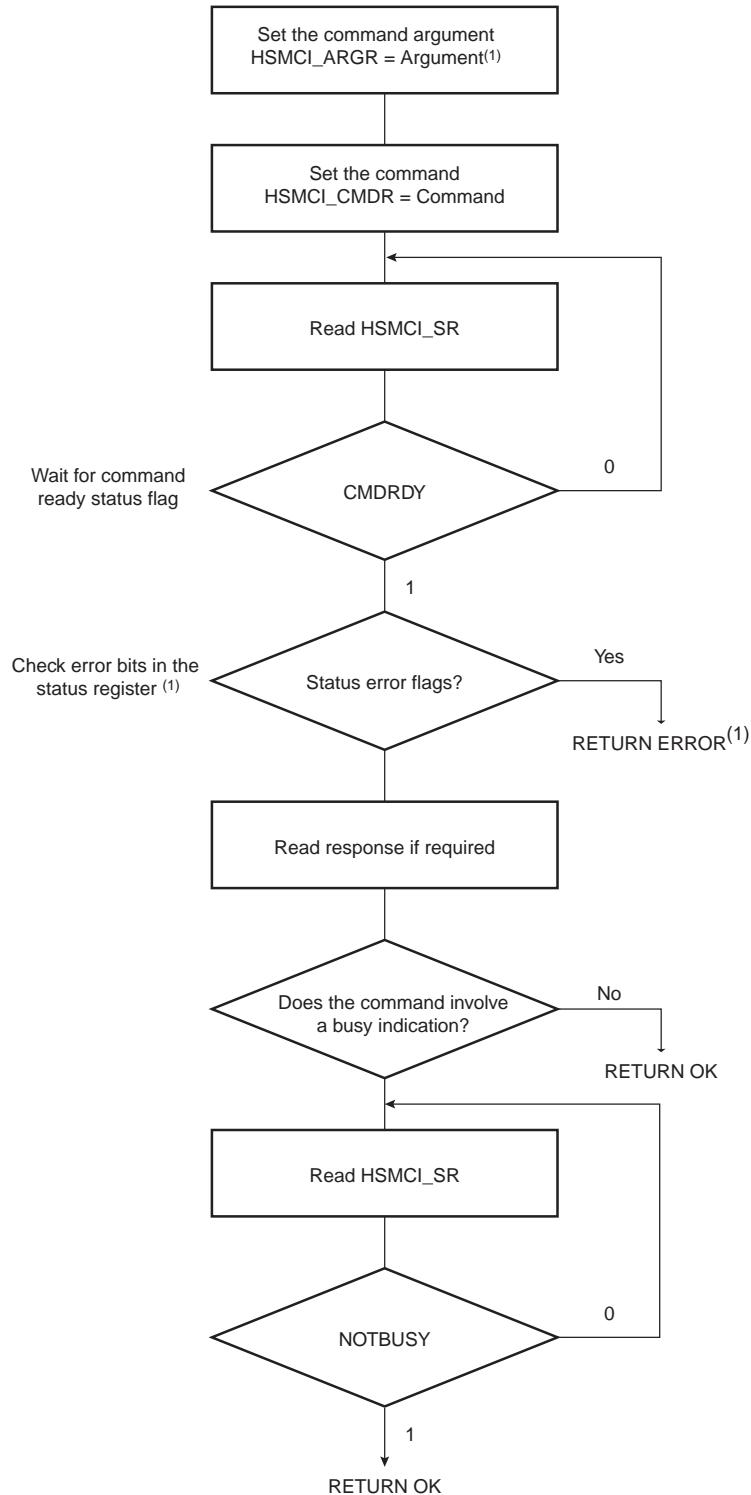
The command is sent immediately after writing the command register.

While the card maintains a busy indication (at the end of a STOP\_TRANSMISSION command CMD12, for example), a new command shall not be sent. The NOTBUSY flag in the status register (HSMCI\_SR) is asserted when the card releases the busy indication.

If the command requires a response, it can be read in the HSMCI Response Register (HSMCI\_RSPR). The response size can be from 48 bits up to 136 bits depending on the command. The HSMCI embeds an error detection to prevent any corrupted data during the transfer.

The following flowchart shows how to send a command to the card and read the response if needed. In this example, the status register bits are polled but setting the appropriate bits in the Interrupt Enable Register (HSMCI\_IER) allows using an interrupt method.

**Figure 39-7. Command/Response Functional Flow Diagram**



Note: 1. If the command is SEND\_OP\_COND, the CRC error flag is always present (refer to R3 response in the High Speed MultiMedia Card specification).

### 39.8.2 Data Transfer Operation

The High Speed MultiMedia Card allows several read/write operations (single block, multiple blocks, stream, etc.). These kinds of transfer can be selected setting the Transfer Type (TRTYP) field in the HSMCI Command Register (HSMCI\_CMDR).

These operations can be done using the features of the Peripheral DMA Controller (PDC). If the PDCMODE bit is set in HSMCI\_MR, then all reads and writes use the PDC facilities.

In all cases, the block length (BLKLEN field) must be defined either in the Mode Register HSMCI\_MR, or in the Block Register HSMCI\_BLKR. This field determines the size of the data block.

Consequent to MMC Specification 3.1, two types of multiple block read (or write) transactions are defined (the host can use either one at any time):

- Open-ended/Infinite Multiple block read (or write):

The number of blocks for the read (or write) multiple block operation is not defined. The card will continuously transfer (or program) data blocks until a stop transmission command is received.

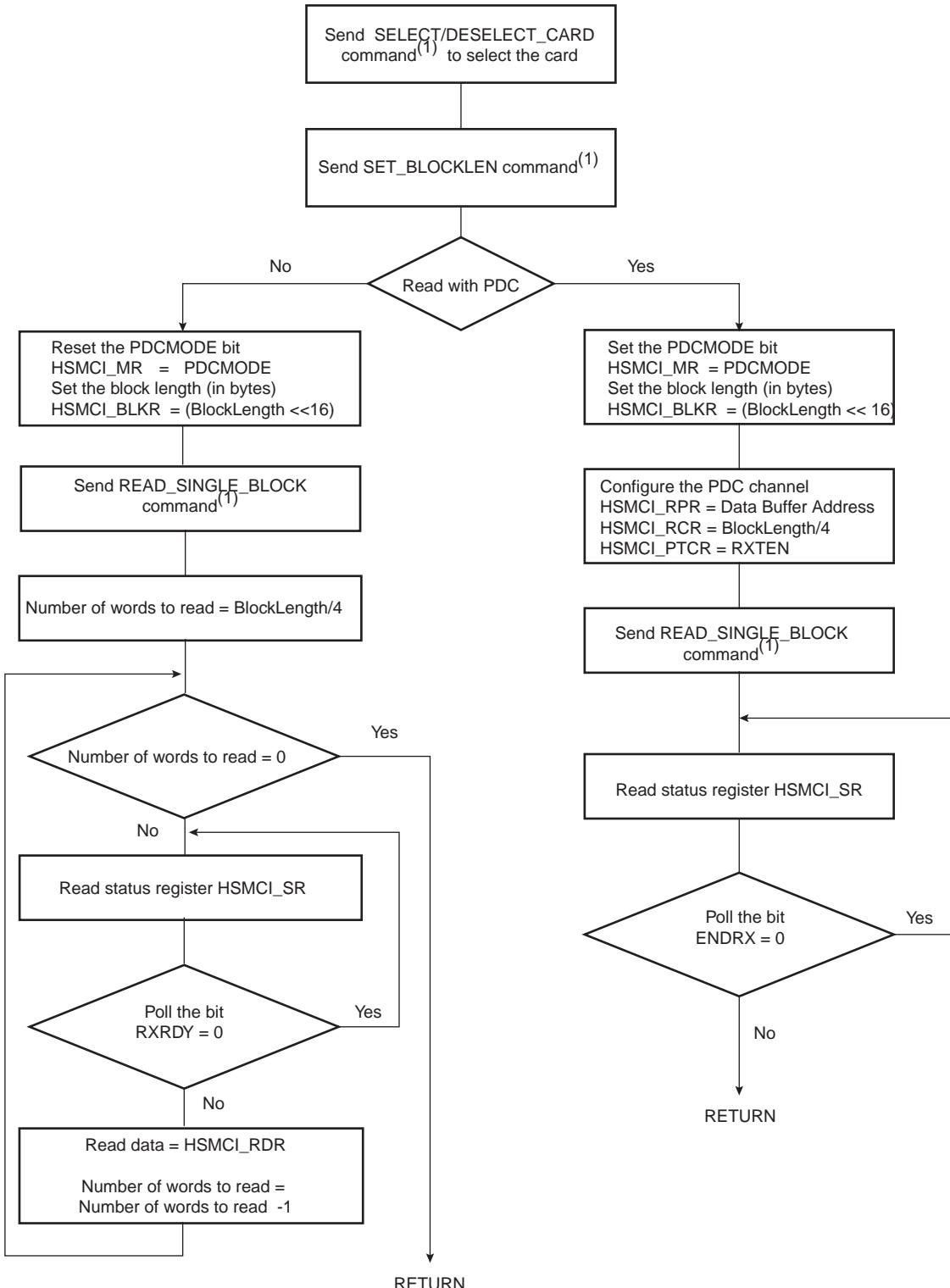
- Multiple block read (or write) with pre-defined block count (since version 3.1 and higher):

The card will transfer (or program) the requested number of data blocks and terminate the transaction. The stop command is not required at the end of this type of multiple block read (or write), unless terminated with an error. In order to start a multiple block read (or write) with pre-defined block count, the host must correctly program the HSMCI Block Register (HSMCI\_BLKR). Otherwise the card will start an open-ended multiple block read. The BCNT field of the Block Register defines the number of blocks to transfer (from 1 to 65535 blocks). Programming the value 0 in the BCNT field corresponds to an infinite block transfer.

### 39.8.3 Read Operation

The following flowchart shows how to read a single block with or without use of PDC facilities. In this example (see [Figure 39-8](#)), a polling method is used to wait for the end of read. Similarly, the user can configure the Interrupt Enable Register (HSMCI\_IER) to trigger an interrupt at the end of read.

**Figure 39-8. Read Functional Flow Diagram**



Note: 1. It is assumed that this command has been correctly sent (see [Figure 39-7](#)).

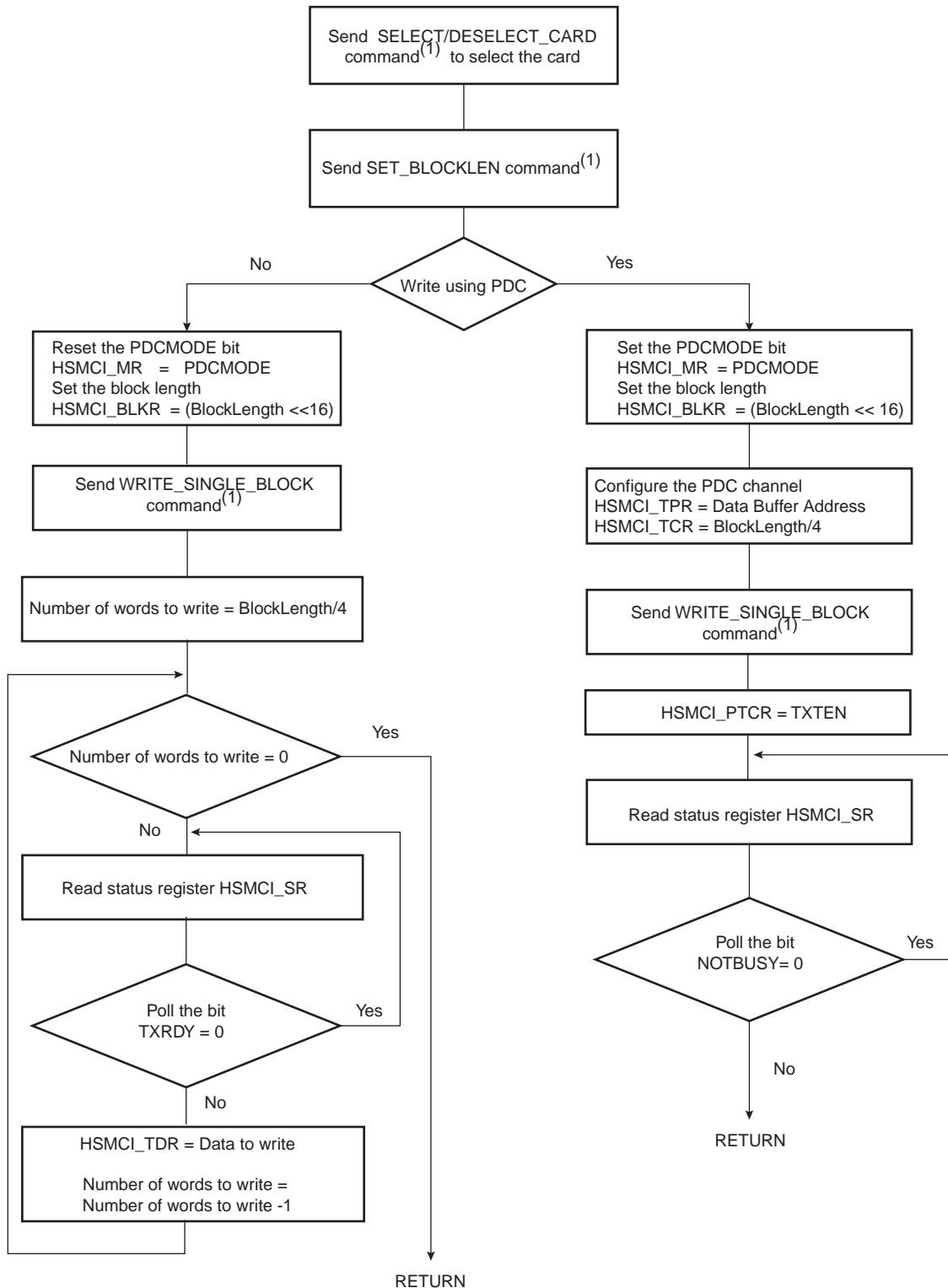
### 39.8.4 Write Operation

In write operation, the HSMCI Mode Register (HSMCI\_MR) is used to define the padding value when writing non-multiple block size. If the bit PDCPADV is 0, then 0x00 value is used when padding data, otherwise 0xFF is used.

If set, the bit PDCMODE enables PDC transfer.

The following flowchart ([Figure 39-9](#)) shows how to write a single block with or without use of PDC facilities. Polling or interrupt method can be used to wait for the end of write according to the contents of the Interrupt Mask Register (HSMCI\_IMR).

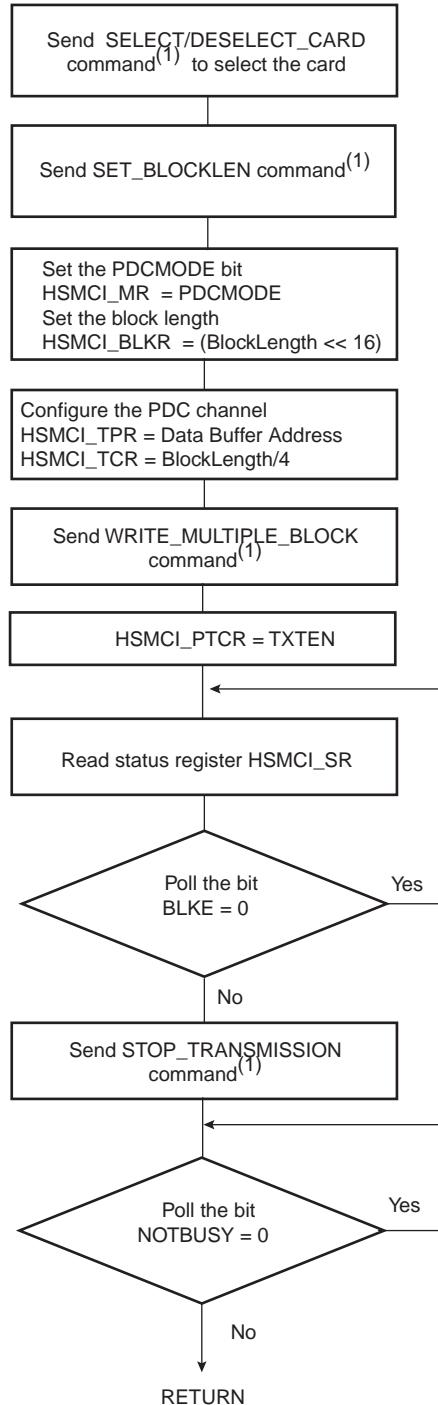
**Figure 39-9. Write Functional Flow Diagram**



Note: 1. It is assumed that this command has been correctly sent (see [Figure 39-7](#)).

The following flowchart ([Figure 39-10](#)) shows how to manage a multiple write block transfer with the PDC. Polling or interrupt method can be used to wait for the end of write according to the contents of the Interrupt Mask Register (HSMCI\_IMR).

**Figure 39-10. Multiple Write Functional Flow Diagram**



Note: 1. It is assumed that this command has been correctly sent (see [Figure 39-7](#)).

## 39.9 SD/SDIO Card Operation

The High Speed MultiMedia Card Interface allows processing of SD Memory (Secure Digital Memory Card) and SDIO (SD Input Output) Card commands.

SD/SDIO cards are based on the MultiMedia Card (MMC) format, but are physically slightly thicker and feature higher data transfer rates, a lock switch on the side to prevent accidental overwriting and security features. The physical form factor, pin assignment and data transfer protocol are forward-compatible with the High Speed MultiMedia Card with some additions. SD slots can actually be used for more than flash memory cards. Devices that support SDIO can use small devices designed for the SD form factor, such as GPS receivers, Wi-Fi or Bluetooth adapters, modems, barcode readers, IrDA adapters, FM radio tuners, RFID readers, digital cameras and more.

SD/SDIO is covered by numerous patents and trademarks, and licensing is only available through the Secure Digital Card Association.

The SD/SDIO Card communication is based on a 9-pin interface (Clock, Command, 4 x Data and 3 x Power lines). The communication protocol is defined as a part of this specification. The main difference between the SD/SDIO Card and the High Speed MultiMedia Card is the initialization process.

The SD/SDIO Card Register (HSMCI\_SDCR) allows selection of the Card Slot and the data bus width.

The SD/SDIO Card bus allows dynamic configuration of the number of data lines. After power up, by default, the SD/SDIO Card uses only DAT0 for data transfer. After initialization, the host can change the bus width (number of active data lines).

### 39.9.1 SDIO Data Transfer Type

SDIO cards may transfer data in either a multi-byte (1 to 512 bytes) or an optional block format (1 to 511 blocks), while the SD memory cards are fixed in the block transfer mode. The TRTYP field in the HSMCI Command Register (HSMCI\_CMDR) allows to choose between SDIO Byte or SDIO Block transfer.

The number of bytes/blocks to transfer is set through the BCNT field in the HSMCI Block Register (HSMCI\_BLKR). In SDIO Block mode, the field BLKLEN must be set to the data block size while this field is not used in SDIO Byte mode.

An SDIO Card can have multiple I/O or combined I/O and memory (called Combo Card). Within a multi-function SDIO or a Combo card, there are multiple devices (I/O and memory) that share access to the SD bus. In order to allow the sharing of access to the host among multiple devices, SDIO and combo cards can implement the optional concept of suspend/resume (Refer to the SDIO Specification for more details). To send a suspend or a resume command, the host must set the SDIO Special Command field (IOSPCMD) in the HSMCI Command Register.

### 39.9.2 SDIO Interrupts

Each function within an SDIO or Combo card may implement interrupts (Refer to the SDIO Specification for more details). In order to allow the SDIO card to interrupt the host, an interrupt function is added to a pin on the DAT[1] line to signal the card's interrupt to the host. An SDIO interrupt on each slot can be enabled through the HSMCI Interrupt Enable Register. The SDIO interrupt is sampled regardless of the currently selected slot.

## 39.10 CE-ATA Operation

CE-ATA maps the streamlined ATA command set onto the MMC interface. The ATA task file is mapped onto MMC register space.

CE-ATA utilizes five MMC commands:

- GO\_IDLE\_STATE (CMD0): used for hard reset.
- STOP\_TRANSMISSION (CMD12): causes the ATA command currently executing to be aborted.
- FAST\_IO (CMD39): Used for single register access to the ATA taskfile registers, 8 bit access only.
- RW\_MULTIPLE\_REGISTERS (CMD60): used to issue an ATA command or to access the control/status registers.
- RW\_MULTIPLE\_BLOCK (CMD61): used to transfer data for an ATA command.

CE-ATA utilizes the same MMC command sequences for initialization as traditional MMC devices.

### 39.10.1 Executing an ATA Polling Command

1. Issue READ\_DMA\_EXT with RW\_MULTIPLE\_REGISTER (CMD60) for 8kB of DATA.
2. Read the ATA status register until DRQ is set.
3. Issue RW\_MULTIPLE\_BLOCK (CMD61) to transfer DATA.
4. Read the ATA status register until DRQ && BSY are set to 0.

### 39.10.2 Executing an ATA Interrupt Command

1. Issue READ\_DMA\_EXT with RW\_MULTIPLE\_REGISTER (CMD60) for 8kB of DATA with nIEN field set to zero to enable the command completion signal in the device.
2. Issue RW\_MULTIPLE\_BLOCK (CMD61) to transfer DATA.
3. Wait for Completion Signal Received Interrupt.

### 39.10.3 Aborting an ATA Command

If the host needs to abort an ATA command prior to the completion signal it must send a special command to avoid potential collision on the command line. The SPCMD field of the HSMCI\_CMDR must be set to 3 to issue the CE-ATA completion Signal Disable Command.

### 39.10.4 CE-ATA Error Recovery

Several methods of ATA command failure may occur, including:

- No response to an MMC command, such as RW\_MULTIPLE\_REGISTER (CMD60).
- CRC is invalid for an MMC command or response.
- CRC16 is invalid for an MMC data packet.
- ATA Status register reflects an error by setting the ERR bit to one.
- The command completion signal does not arrive within a host specified time out period.

Error conditions are expected to happen infrequently. Thus, a robust error recovery mechanism may be used for each error event. The recommended error recovery procedure after a timeout is:

- Issue the command completion signal disable if nIEN was cleared to zero and the RW\_MULTIPLE\_BLOCK (CMD61) response has been received.
- Issue STOP\_TRANSMISSION (CMD12) and successfully receive the R1 response.
- Issue a software reset to the CE-ATA device using FAST\_IO (CMD39).

If STOP\_TRANSMISSION (CMD12) is successful, then the device is again ready for ATA commands. However, if the error recovery procedure does not work as expected or there is another timeout, the next step is to issue GO\_IDLE\_STATE (CMD0) to the device. GO\_IDLE\_STATE (CMD0) is a hard reset to the device and completely resets all device states.

Note that after issuing GO\_IDLE\_STATE (CMD0), all device initialization needs to be completed again. If the CE-ATA device completes all MMC commands correctly but fails the ATA command with the ERR bit set in the ATA Status register, no error recovery action is required. The ATA command itself failed implying that the device could not complete the action requested, however, there was no communication or protocol failure. After the device signals an error by setting the ERR bit to one in the ATA Status register, the host may attempt to retry the command.

## 39.11 HSMCI Boot Operation Mode

In boot operation mode, the processor can read boot data from the slave (MMC device) by keeping the CMD line low after power-on before issuing CMD1. The data can be read from either the boot area or user area, depending on register setting. As it is not possible to boot directly on SD-CARD, a preliminary boot code must be stored in internal Flash.

### 39.11.1 Boot Procedure, Processor Mode

1. Configure the HSMCI data bus width programming SDCBUS Field in the HSMCI\_SDCR register. The BOOT\_BUS\_WIDTH field located in the device Extended CSD register must be set accordingly.
2. Set the byte count to 512 bytes and the block count to the desired number of blocks, writing BLKLEN and BCNT fields of the HSMCI\_BLKR Register.
3. Issue the Boot Operation Request command by writing to the HSMCI\_CMDR register with SPCMD field set to BOOTREQ, TRDIR set to READ and TRCMD set to "start data transfer".
4. The BOOT\_ACK field located in the HSMCI\_CMDR register must be set to one, if the BOOT\_ACK field of the MMC device located in the Extended CSD register is set to one.
5. Host processor can copy boot data sequentially as soon as the RXRDY flag is asserted.
6. When Data transfer is completed, host processor shall terminate the boot stream by writing the HSMCI\_CMDR register with SPCMD field set to BOOTEND.

## 39.12 HSMCI Transfer Done Timings

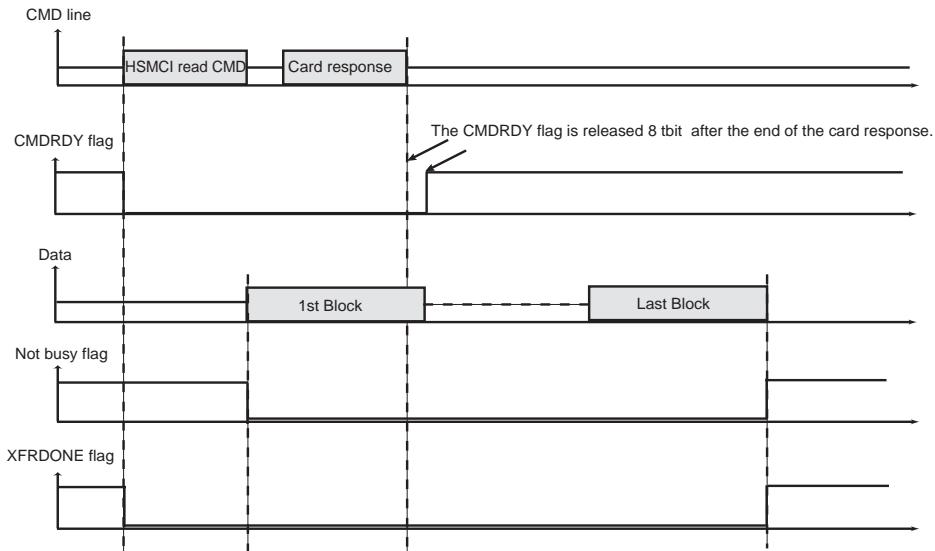
### 39.12.1 Definition

The XFRDONE flag in the HSMCI\_SR indicates exactly when the read or write sequence is finished.

### 39.12.2 Read Access

During a read access, the XFRDONE flag behaves as shown in [Figure 39-11](#).

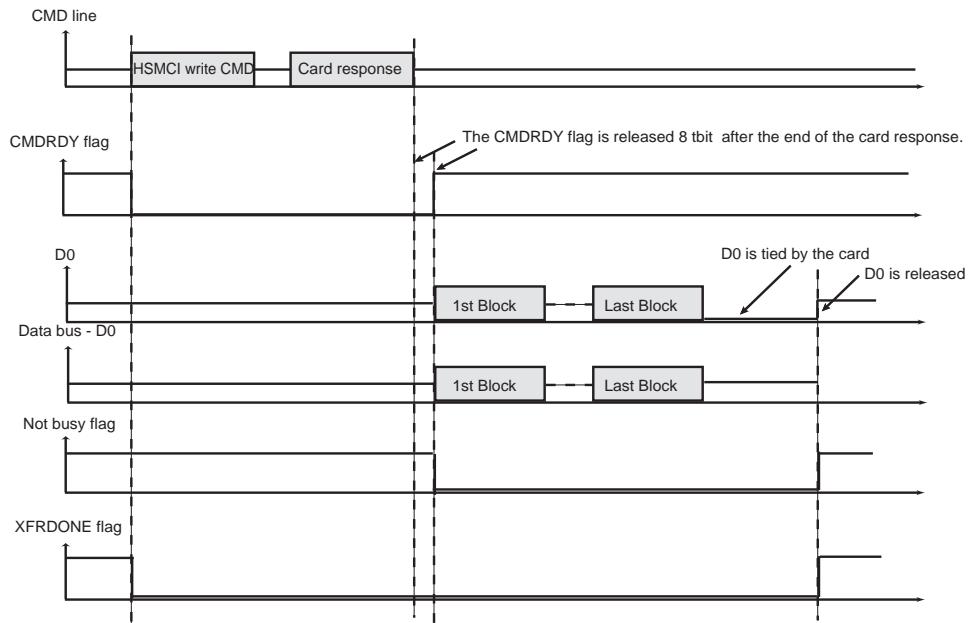
**Figure 39-11. XFRDONE During a Read Access**



### 39.12.3 Write Access

During a write access, the XFRDONE flag behaves as shown in [Figure 39-12](#).

**Figure 39-12. XFRDONE During a Write Access**



## 39.13 Write Protection Registers

To prevent any single software error that may corrupt HSMCI behavior, the entire HSMCI address space from address offset 0x000 to 0x00FC can be write-protected by setting the WPEN bit in the “[HSMCI Write Protect Mode Register](#)” (HSMCI\_WPMR).

If a write access to anywhere in the HSMCI address space from address offset 0x000 to 0x00FC is detected, then the WPVS flag in the HSMCI Write Protect Status Register (HSMCI\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the HSMCI Write Protect Mode Register (HSMCI\_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- [“HSMCI Mode Register” on page 974](#)
- [“HSMCI Data Timeout Register” on page 976](#)
- [“HSMCI SDCard/SDIO Register” on page 977](#)
- [“HSMCI Completion Signal Timeout Register” on page 982](#)
- [“HSMCI Configuration Register” on page 996](#)

## 39.14 High Speed MultiMedia Card Interface (HSMCI) User Interface

Table 39-8. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	HSMCI_CR	Write	–
0x04	Mode Register	HSMCI_MR	Read-write	0x0
0x08	Data Timeout Register	HSMCI_DTOR	Read-write	0x0
0x0C	SD/SDIO Card Register	HSMCI_SDCR	Read-write	0x0
0x10	Argument Register	HSMCI_ARGR	Read-write	0x0
0x14	Command Register	HSMCI_CMDR	Write	–
0x18	Block Register	HSMCI_BLKR	Read-write	0x0
0x1C	Completion Signal Timeout Register	HSMCI_CSTOR	Read-write	0x0
0x20	Response Register <sup>(1)</sup>	HSMCI_RSPPR	Read	0x0
0x24	Response Register <sup>(1)</sup>	HSMCI_RSPPR	Read	0x0
0x28	Response Register <sup>(1)</sup>	HSMCI_RSPPR	Read	0x0
0x2C	Response Register <sup>(1)</sup>	HSMCI_RSPPR	Read	0x0
0x30	Receive Data Register	HSMCI_RDR	Read	0x0
0x34	Transmit Data Register	HSMCI_TDR	Write	–
0x38 - 0x3C	Reserved	–	–	–
0x40	Status Register	HSMCI_SR	Read	0xC0E5
0x44	Interrupt Enable Register	HSMCI_IER	Write	–
0x48	Interrupt Disable Register	HSMCI_IDR	Write	–
0x4C	Interrupt Mask Register	HSMCI_IMR	Read	0x0
0x50	Reserved	–	–	–
0x54	Configuration Register	HSMCI_CFG	Read-write	0x00
0x58-0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	HSMCI_WPMR	Read-write	–
0xE8	Write Protection Status Register	HSMCI_WPSR	Read-only	–
0xEC - 0xFC	Reserved	–	–	–
0x100-0x128	Reserved for PDC registers	–	–	–
0x12Cx1FC	Reserved	–	–	–
0x200	FIFO Memory Aperture0	HSMCI_FIFO0	Read-write	0x0
...	...	...	...	...
0x5FC	FIFO Memory Aperture255	HSMCI_FIFO255	Read-write	0x0

Notes: 1. The Response Register can be read by N accesses at the same HSMCI\_RSPPR or at consecutive addresses (0x20 to 0x2C). N depends on the size of the response.

### 39.14.1 HSMCI Control Register

**Name:** HSMCI\_CR

**Address:** 0x40080000

**Access:** Write-only

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
SWRST	—	—	—	PWSDIS	PWSEN	MCIDIS	MCIEN

- **MCIEN: Multi-Media Interface Enable**

0 = No effect.

1 = Enables the Multi-Media Interface if MCDIS is 0.

- **MCIDIS: Multi-Media Interface Disable**

0 = No effect.

1 = Disables the Multi-Media Interface.

- **PWSEN: Power Save Mode Enable**

0 = No effect.

1 = Enables the Power Saving Mode if PWSDIS is 0.

**Warning:** Before enabling this mode, the user must set a value different from 0 in the PWSDIV field (Mode Register, HSMCI\_MR).

- **PWSDIS: Power Save Mode Disable**

0 = No effect.

1 = Disables the Power Saving Mode.

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the HSMCI. A software triggered hardware reset of the HSMCI interface is performed.

### 39.14.2 HSMCI Mode Register

**Name:** HSMCI\_MR  
**Address:** 0x40080004  
**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	CLKODD
15	14	13	12	11	10	9	8
PDCMODE	PADV	FBYTE	WRPROOF	RDPROOF	PWSDIV		
7	6	5	4	3	2	1	0
CLKDIV							

This register can only be written if the WPEN bit is cleared in “[HSMCI Write Protect Mode Register](#)” on page 997.

- **CLKDIV: Clock Divider**

High Speed MultiMedia Card Interface clock (MCCK or HSMCI\_CK) is Master Clock (MCK) divider by  $(\{CLKDIV,CLKODD\}+2)$ .

- **PWSDIV: Power Saving Divider**

High Speed MultiMedia Card Interface clock is divided by  $2^{(PWSDIV)} + 1$  when entering Power Saving Mode.

**Warning:** This value must be different from 0 before enabling the Power Save Mode in the HSMCI\_CR (HSMCI\_PWSEN bit).

- **RDPROOF: Read Proof Enable**

Enabling Read Proof allows to stop the HSMCI Clock during read access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

0 = Disables Read Proof.

1 = Enables Read Proof.

- **WRPROOF: Write Proof Enable**

Enabling Write Proof allows to stop the HSMCI Clock during write access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

0 = Disables Write Proof.

1 = Enables Write Proof.

- **FBYTE: Force Byte Transfer**

Enabling Force Byte Transfer allow byte transfers, so that transfer of blocks with a size different from modulo 4 can be supported.

**Warning:** BLKLEN value depends on FBYTE.

0 = Disables Force Byte Transfer.

1 = Enables Force Byte Transfer.

- **PADV: Padding Value**

0 = 0x00 value is used when padding data in write transfer.

1 = 0xFF value is used when padding data in write transfer.

PADV may be only in manual transfer.

- **PDCMODE: PDC-oriented Mode**

0 = Disables PDC transfer

1 = Enables PDC transfer. In this case, UNRE and OVRE flags in the HSMCI Status Register (HSMCI\_SR) are deactivated after the PDC transfer has been completed.

- **CLKODD: Clock divider is odd**

This field is the least significant bit of the clock divider and indicates the clock divider parity.

### 39.14.3 HSMCI Data Timeout Register

**Name:** HSMCI\_DTOR

**Address:** 0x40080008

**Access:** Read-write

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
–	DTOMUL			DTOCYC			

This register can only be written if the WPEN bit is cleared in “[HSMCI Write Protect Mode Register](#)” on page 997.

- **DTOCYC: Data Timeout Cycle Number**

These fields determine the maximum number of Master Clock cycles that the HSMCI waits between two data block transfers. It equals (DTOCYC x Multiplier).

- **DTOMUL: Data Timeout Multiplier**

Multiplier is defined by DTOMUL as shown in the following table:

Value	Name	Description
0	1	DTOCYC
1	16	DTOCYC x 16
2	128	DTOCYC x 128
3	256	DTOCYC x 256
4	1024	DTOCYC x 1024
5	4096	DTOCYC x 4096
6	65536	DTOCYC x 65536
7	1048576	DTOCYC x 1048576

If the data time-out set by DTOCYC and DTOMUL has been exceeded, the Data Time-out Error flag (DTOE) in the HSMCI Status Register (HSMCI\_SR) rises.

### 39.14.4 HSMCI SDCard/SDIO Register

**Name:** HSMCI\_SDCR

**Address:** 0x4008000C

**Access:** Read-write

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
SDCBUS	–	–	–	–	–	–	SDCSEL

This register can only be written if the WPEN bit is cleared in “HSMCI Write Protect Mode Register” on page 997.

- **SDCSEL: SDCard/SDIO Slot**

Value	Name	Description
0	SLOTA	Slot A is selected.
1	SLOTB	–
2	SLOTC	–
3	SLOTD	–

- **SDCBUS: SDCard/SDIO Bus Width**

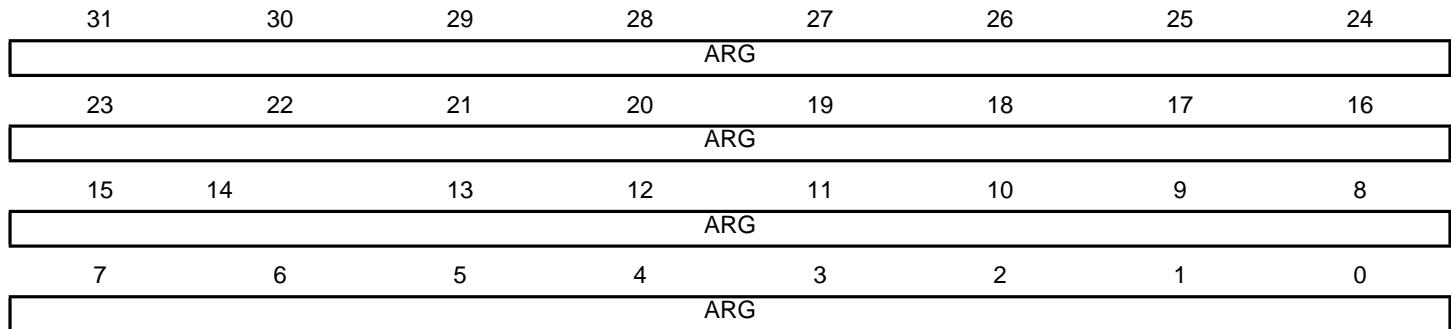
Value	Name	Description
0	1	1 bit
1	–	Reserved
2	4	4 bit
3	8	8 bit

### 39.14.5 HSMCI Argument Register

**Name:** HSMCI\_ARGR

**Address:** 0x40080010

**Access:** Read-write



- **ARG: Command Argument**

### 39.14.6 HSMCI Command Register

**Name:** HSMCI\_CMDR

**Address:** 0x40080014

**Access:** Write-only

31	30	29	28	27	26	25	24
-	-	-	-	BOOT_ACK	ATACS	IOPSPCMD	
23	22	21	20	19	18	17	16
-	-	TRTYP		TRDIR		TRCMD	
15	14	13	12	11	10	9	8
-	-	-	MAXLAT	OPDCMD		SPCMD	
7	6	5	4	3	2	1	0
RSPTYP		CMDNB					

This register is write-protected while CMDRDY is 0 in HSMCI\_SR. If an interrupt command is sent, this register is only writable by an interrupt response (field SPCMD). This means that the current command execution cannot be interrupted or modified.

- **CMDBN: Command Number**

This is the command index.

- **RSPTYP: Response Type**

Value	Name	Description
0	NORESP	No response.
1	48_BIT	48-bit response.
2	136_BIT	136-bit response.
3	R1B	R1b response type

- **SPCMD: Special Command**

Value	Name	Description
0	STD	Not a special CMD.
1	INIT	Initialization CMD: 74 clock cycles for initialization sequence.
2	SYNC	Synchronized CMD: Wait for the end of the current data block transfer before sending the pending command.
3	CE_ATA	CE-ATA Completion Signal disable Command. The host cancels the ability for the device to return a command completion signal on the command line.
4	IT_CMD	Interrupt command: Corresponds to the Interrupt Mode (CMD40).
5	IT_RESP	Interrupt response: Corresponds to the Interrupt Mode (CMD40).
6	BOR	Boot Operation Request. Start a boot operation mode, the host processor can read boot data from the MMC device directly.
7	EBO	End Boot Operation. This command allows the host processor to terminate the boot operation mode.

- **OPDCMD: Open Drain Command**

0 (PUSHPULL) = Push pull command.

1 (OPENDRAIN) = Open drain command.

- **MAXLAT: Max Latency for Command to Response**

0 (5) = 5-cycle max latency.

1 (64) = 64-cycle max latency.

- **TRCMD: Transfer Command**

Value	Name	Description
0	NO_DATA	No data transfer
1	START_DATA	Start data transfer
2	STOP_DATA	Stop data transfer
3	-	Reserved

- **TRDIR: Transfer Direction**

0 (WRITE) = Write.

1 (READ) = Read.

- **TRTYP: Transfer Type**

Value	Name	Description
0	SINGLE	MMC/SD Card Single Block
1	MULTIPLE	MMC/SD Card Multiple Block
2	STREAM	MMC Stream
4	BYTE	SDIO Byte
5	BLOCK	SDIO Block

- **IOSPCMD: SDIO Special Command**

Value	Name	Description
0	STD	Not an SDIO Special Command
1	SUSPEND	SDIO Suspend Command
2	RESUME	SDIO Resume Command

- **ATACS: ATA with Command Completion Signal**

0 (NORMAL) = Normal operation mode.

1 (COMPLETION) = This bit indicates that a completion signal is expected within a programmed amount of time (HSMCI\_CSTOR).

- **BOOT\_ACK: Boot Operation Acknowledge.**

The master can choose to receive the boot acknowledge from the slave when a Boot Request command is issued. When set to one this field indicates that a Boot acknowledge is expected within a programmable amount of time defined with DTOMUL and DTOCYC fields located in the HSMCI\_DTOR register. If the acknowledge pattern is not received then an acknowledge timeout error is raised. If the acknowledge pattern is corrupted then an acknowledge pattern error is set.

### 39.14.7 HSMCI Block Register

**Name:** HSMCI\_BLKR

**Address:** 0x40080018

**Access:** Read-write

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

#### • BCNT: MMC/SDIO Block Count - SDIO Byte Count

This field determines the number of data byte(s) or block(s) to transfer.

The transfer data type and the authorized values for BCNT field are determined by the TRTYP field in the HSMCI Command Register (HSMCI\_CMDR).

When TRTYP=1 (MMC/SDCARD Multiple Block), BCNT can be programmed from 1 to 65535, 0 corresponds to an infinite block transfer.

When TRTYP=4 (SDIO Byte), BCNT can be programmed from 1 to 511, 0 corresponds to 512-byte transfer. Values in range 512 to 65536 are forbidden.

When TRTYP=5 (SDIO Block), BCNT can be programmed from 1 to 511, 0 corresponds to an infinite block transfer. Values in range 512 to 65536 are forbidden.

**Warning:** In SDIO Byte and Block modes (TRTYP=4 or 5), writing the 7 last bits of BCNT field with a value which differs from 0 is forbidden and may lead to unpredictable results.

#### • BLKLEN: Data Block Length

This field determines the size of the data block.

Bits 16 and 17 must be set to 0 if FBYTE is disabled.

Note: In SDIO Byte mode, BLKLEN field is not used.

### 39.14.8 HSMCI Completion Signal Timeout Register

**Name:** HSMCI\_CSTOR

**Address:** 0x4008001C

**Access:** Read-write

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
—	CSTOMUL			CSTOCYC			

This register can only be written if the WPEN bit is cleared in “[HSMCI Write Protect Mode Register](#)” on page 997.

- **CSTOCYC: Completion Signal Timeout Cycle Number**

These fields determine the maximum number of Master Clock cycles that the HSMCI waits between two data block transfers. Its value is calculated by (CSTOCYC x Multiplier).

- **CSTOMUL: Completion Signal Timeout Multiplier**

These fields determine the maximum number of Master Clock cycles that the HSMCI waits between two data block transfers. Its value is calculated by (CSTOCYC x Multiplier).

These fields determine the maximum number of Master Clock cycles that the HSMCI waits between the end of the data transfer and the assertion of the completion signal. The data transfer comprises data phase and the optional busy phase. If a non-DATA ATA command is issued, the HSMCI starts waiting immediately after the end of the response until the completion signal.

Multiplier is defined by CSTOMUL as shown in the following table:

Value	Name	Description
0	1	CSTOCYC x 1
1	16	CSTOCYC x 16
2	128	CSTOCYC x 128
3	256	CSTOCYC x 256
4	1024	CSTOCYC x 1024
5	4096	CSTOCYC x 4096
6	65536	CSTOCYC x 65536
7	1048576	CSTOCYC x 1048576

If the data time-out set by CSTOCYC and CSTOMUL has been exceeded, the Completion Signal Time-out Error flag (CSTOE) in the HSMCI Status Register (HSMCI\_SR) rises.

### 39.14.9 HSMCI Response Register

Name: HSMCI\_RSPR

Address: 0x40080020

Access: Read-only

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

- **RSP: Response**

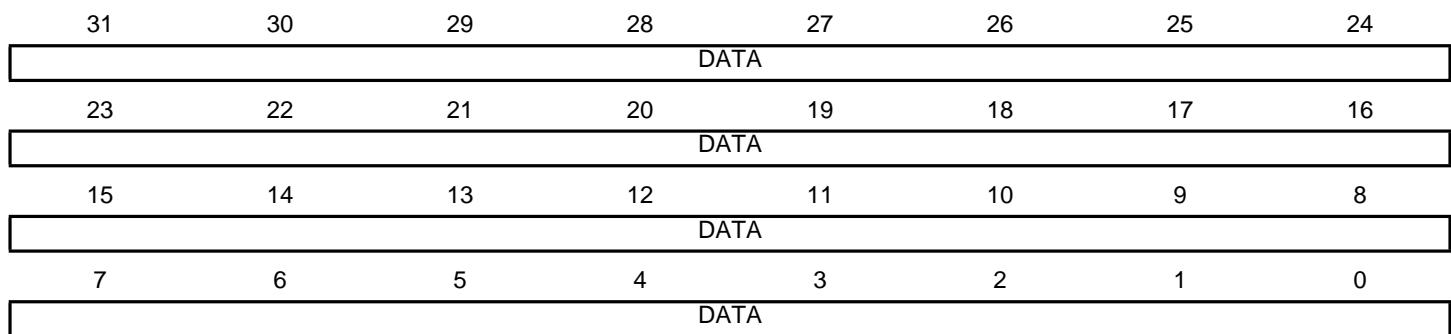
Note: 1. The response register can be read by N accesses at the same HSMCI\_RSPR or at consecutive addresses (0x20 to 0x2C).  
N depends on the size of the response.

### 39.14.10HSMCI Receive Data Register

**Name:** HSMCI\_RDR

**Address:** 0x40080030

**Access:** Read-only



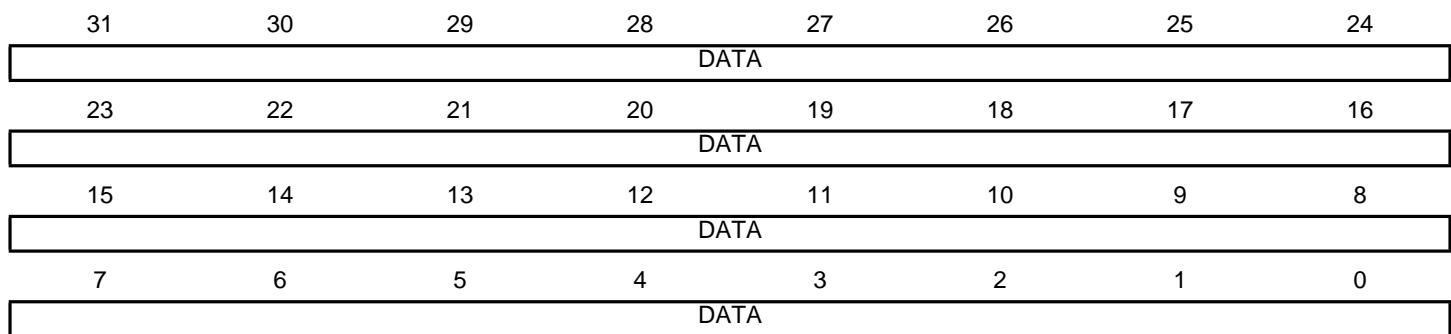
- **DATA: Data to Read**

### 39.14.11 HSMCI Transmit Data Register

**Name:** HSMCI\_TDR

**Address:** 0x40080034

**Access:** Write-only



- **DATA: Data to Write**

### 39.14.12HSMCI Status Register

**Name:** HSMCI\_SR

**Address:** 0x40080040

**Access:** Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	-	-
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	CSRCV	SDIOWAIT	-	-	-	SDIOIRQA
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready**

0 = A command is in progress.

1 = The last command has been sent. Cleared when writing in the HSMCI\_CMDR.

- **RXRDY: Receiver Ready**

0 = Data has not yet been received since the last read of HSMCI\_RDR.

1 = Data has been received since the last read of HSMCI\_RDR.

- **TXRDY: Transmit Ready**

0= The last data written in HSMCI\_TDR has not yet been transferred in the Shift Register.

1= The last data written in HSMCI\_TDR has been transferred in the Shift Register.

- **BLKE: Data Block Ended**

This flag must be used only for Write Operations.

0 = A data block transfer is not yet finished. Cleared when reading the HSMCI\_SR.

1 = A data block transfer has ended, including the CRC16 Status transmission.

the flag is set for each transmitted CRC Status.

Refer to the MMC or SD Specification for more details concerning the CRC Status.

- **DTIP: Data Transfer in Progress**

0 = No data transfer in progress.

1 = The current data transfer is still in progress, including CRC16 calculation. Cleared at the end of the CRC16 calculation.

- **NOTBUSY: HSMCI Not Busy**

A block write operation uses a simple busy signalling of the write operation duration on the data (DAT0) line: during a data transfer block, if the card does not have a free data receive buffer, the card indicates this condition by pulling down the data line (DAT0) to LOW. The card stops pulling down the data line as soon as at least one receive buffer for the defined data transfer block length becomes free.

Refer to the MMC or SD Specification for more details concerning the busy behavior.

For all the read operations, the NOTBUSY flag is cleared at the end of the host command.

For the Infinite Read Multiple Blocks, the NOTBUSY flag is set at the end of the STOP\_TRANSMISSION host command (CMD12).

For the Single Block Reads, the NOTBUSY flag is set at the end of the data read block.

For the Multiple Block Reads with pre-defined block count, the NOTBUSY flag is set at the end of the last received data block.

The NOTBUSY flag allows to deal with these different states.

0 = The HSMCI is not ready for new data transfer. Cleared at the end of the card response.

1 = The HSMCI is ready for new data transfer. Set when the busy state on the data line has ended. This corresponds to a free internal data receive buffer of the card.

- **ENDRX: End of RX Buffer**

0 = The Receive Counter Register has not reached 0 since the last write in HSMCI\_RCR or HSMCI\_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in HSMCI\_RCR or HSMCI\_RNCR.

- **ENDTX: End of TX Buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in HSMCI\_TCR or HSMCI\_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in HSMCI\_TCR or HSMCI\_TNCR.

Note: BLKE and NOTBUSY flags can be used to check that the data has been successfully transmitted on the data lines and not only transferred from the PDC to the HSMCI Controller.

- **SDIOIRQA: SDIO Interrupt for Slot A**

0 = No interrupt detected on SDIO Slot A.

1 = An SDIO Interrupt on Slot A occurred. Cleared when reading the HSMCI\_SR.

- **SDIOWAIT: SDIO Read Wait Operation Status**

0 = Normal Bus operation.

1 = The data bus has entered IO wait state.

- **CSRCV: CE-ATA Completion Signal Received**

0 = No completion signal received since last status read operation.

1 = The device has issued a command completion signal on the command line. Cleared by reading in the HSMCI\_SR register.

- **RXBUFF: RX Buffer Full**

0 = HSMCI\_RCR or HSMCI\_RNCR has a value other than 0.

1 = Both HSMCI\_RCR and HSMCI\_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = HSMCI\_TCR or HSMCI\_TNCR has a value other than 0.

1 = Both HSMCI\_TCR and HSMCI\_TNCR have a value of 0.

Note: BLKE and NOTBUSY flags can be used to check that the data has been successfully transmitted on the data lines and not only transferred from the PDC to the HSMCI Controller.

- **RINDE: Response Index Error**

0 = No error.

1 = A mismatch is detected between the command index sent and the response index received. Cleared when writing in the HSMCI\_CMDR.

- **RDIRE: Response Direction Error**

0 = No error.

1 = The direction bit from card to host in the response has not been detected.

- **RCRCE: Response CRC Error**

0 = No error.

1 = A CRC7 error has been detected in the response. Cleared when writing in the HSMCI\_CMDR.

- **RENDE: Response End Bit Error**

0 = No error.

1 = The end bit of the response has not been detected. Cleared when writing in the HSMCI\_CMDR.

- **RTOE: Response Time-out Error**

0 = No error.

1 = The response time-out set by MAXLAT in the HSMCI\_CMDR has been exceeded. Cleared when writing in the HSMCI\_CMDR.

- **DCRCE: Data CRC Error**

0 = No error.

1 = A CRC16 error has been detected in the last data block. Cleared by reading in the HSMCI\_SR register.

- **DTOE: Data Time-out Error**

0 = No error.

1 = The data time-out set by DTOCYC and DTOMUL in HSMCI\_DTOR has been exceeded. Cleared by reading in the HSMCI\_SR register.

- **CSTOE: Completion Signal Time-out Error**

0 = No error.

1 = The completion signal time-out set by CSTOCYC and CSTOMUL in HSMCI\_CSTOR has been exceeded. Cleared by reading in the HSMCI\_SR register. Cleared by reading in the HSMCI\_SR register.

- **FIFOEMPTY: FIFO empty flag**

0 = FIFO contains at least one byte.

1 = FIFO is empty.

- **XFRDONE: Transfer Done flag**

0 = A transfer is in progress.

1 = Command Register is ready to operate and the data bus is in the idle state.

- **ACKRCV: Boot Operation Acknowledge Received**

0 = No Boot acknowledge received since the last read of the status register.

1 = A Boot acknowledge signal has been received. Cleared by reading the HSMCI\_SR register.

- **ACKRCVE: Boot Operation Acknowledge Error**

0 = No error

1 = Corrupted Boot Acknowledge signal received.

- **OVRE: Overrun**

0 = No error.

1 = At least one 8-bit received data has been lost (not read). Cleared when sending a new data transfer command.

When FERRCTRL in HSMCI\_CFG is set to 1, OVRE becomes reset after read.

- **UNRE: Underrun**

0 = No error.

1 = At least one 8-bit data has been sent without valid information (not written). Cleared when sending a new data transfer command or when setting FERRCTRL in HSMCI\_CFG to 1.

When FERRCTRL in HSMCI\_CFG is set to 1, UNRE becomes reset after read.

### 39.14.13HSMCI Interrupt Enable Register

**Name:** HSMCI\_IER

**Address:** 0x40080044

**Access:** Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	-	-
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	CSRCV	SDIOWAIT	-	-	-	SDIOIRQA
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY:** Command Ready Interrupt Enable
- **RXRDY:** Receiver Ready Interrupt Enable
- **TXRDY:** Transmit Ready Interrupt Enable
- **BLKE:** Data Block Ended Interrupt Enable
- **DTIP:** Data Transfer in Progress Interrupt Enable
- **NOTBUSY:** Data Not Busy Interrupt Enable
- **ENDRX:** End of Receive Buffer Interrupt Enable
- **ENDTX:** End of Transmit Buffer Interrupt Enable
- **SDIOIRQA:** SDIO Interrupt for Slot A Interrupt Enable
- **SDIOWAIT:** SDIO Read Wait Operation Status Interrupt Enable
- **CSRCV:** Completion Signal Received Interrupt Enable
- **RXBUFF:** Receive Buffer Full Interrupt Enable
- **TXBUFE:** Transmit Buffer Empty Interrupt Enable
- **RINDE:** Response Index Error Interrupt Enable
- **RDIRE:** Response Direction Error Interrupt Enable
- **RCRCE:** Response CRC Error Interrupt Enable
- **RENDE:** Response End Bit Error Interrupt Enable
- **RTOE:** Response Time-out Error Interrupt Enable
- **DCRCE:** Data CRC Error Interrupt Enable

- **DTOE:** Data Time-out Error Interrupt Enable
- **CSTOE:** Completion Signal Timeout Error Interrupt Enable
- **FIFOEMPTY:** FIFO empty Interrupt enable
- **XFRDONE:** Transfer Done Interrupt enable
- **ACKRCV:** Boot Acknowledge Interrupt Enable
- **ACKRCVE:** Boot Acknowledge Error Interrupt Enable
- **OVRE:** Overrun Interrupt Enable
- **UNRE:** Underrun Interrupt Enable

0 = No effect.

1 = Enables the corresponding interrupt.

### 39.14.14HSMCI Interrupt Disable Register

**Name:** HSMCI\_IDR

**Address:** 0x40080048

**Access:** Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	-	-
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	CSRCV	SDIOWAIT	-	-	-	SDIOIRQA
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY:** Command Ready Interrupt Disable
- **RXRDY:** Receiver Ready Interrupt Disable
- **TXRDY:** Transmit Ready Interrupt Disable
- **BLKE:** Data Block Ended Interrupt Disable
- **DTIP:** Data Transfer in Progress Interrupt Disable
- **NOTBUSY:** Data Not Busy Interrupt Disable
- **ENDRX:** End of Receive Buffer Interrupt Disable
- **ENDTX:** End of Transmit Buffer Interrupt Disable
- **SDIOIRQA:** SDIO Interrupt for Slot A Interrupt Disable
- **SDIOWAIT:** SDIO Read Wait Operation Status Interrupt Disable
- **CSRCV:** Completion Signal received interrupt Disable
- **RXBUFF:** Receive Buffer Full Interrupt Disable
- **TXBUFE:** Transmit Buffer Empty Interrupt Disable
- **RINDE:** Response Index Error Interrupt Disable
- **RDIRE:** Response Direction Error Interrupt Disable
- **RCRCE:** Response CRC Error Interrupt Disable
- **RENDE:** Response End Bit Error Interrupt Disable
- **RTOE:** Response Time-out Error Interrupt Disable
- **DCRCE:** Data CRC Error Interrupt Disable

- **DTOE: Data Time-out Error Interrupt Disable**
- **CSTOE: Completion Signal Time out Error Interrupt Disable**
- **FIFOEMPTY: FIFO empty Interrupt Disable**
- **XFRDONE: Transfer Done Interrupt Disable**
- **ACKRCV: Boot Acknowledge Interrupt Disable**
- **ACKRCVE: Boot Acknowledge Error Interrupt Disable**
- **OVRE: Overrun Interrupt Disable**
- **UNRE: Underrun Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

### 39.14.15HSMCI Interrupt Mask Register

**Name:** HSMCI\_IMR

**Address:** 0x4008004C

**Access:** Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	-	-
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	CSRCV	SDIOWAIT	-	-	-	SDIOIRQA
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY:** Command Ready Interrupt Mask
- **RXRDY:** Receiver Ready Interrupt Mask
- **TXRDY:** Transmit Ready Interrupt Mask
- **BLKE:** Data Block Ended Interrupt Mask
- **DTIP:** Data Transfer in Progress Interrupt Mask
- **NOTBUSY:** Data Not Busy Interrupt Mask
- **ENDRX:** End of Receive Buffer Interrupt Mask
- **ENDTX:** End of Transmit Buffer Interrupt Mask
- **SDIOIRQA:** SDIO Interrupt for Slot A Interrupt Mask
- **SDIOWAIT:** SDIO Read Wait Operation Status Interrupt Mask
- **CSRCV:** Completion Signal Received Interrupt Mask
- **RXBUFF:** Receive Buffer Full Interrupt Mask
- **TXBUFE:** Transmit Buffer Empty Interrupt Mask
- **RINDE:** Response Index Error Interrupt Mask
- **RDIRE:** Response Direction Error Interrupt Mask
- **RCRCE:** Response CRC Error Interrupt Mask
- **RENDE:** Response End Bit Error Interrupt Mask
- **RTOE:** Response Time-out Error Interrupt Mask
- **DCRCE:** Data CRC Error Interrupt Mask

- **DTOE: Data Time-out Error Interrupt Mask**
- **CSTOE: Completion Signal Time-out Error Interrupt Mask**
- **FIFOEMPTY: FIFO Empty Interrupt Mask**
- **XFRDONE: Transfer Done Interrupt Mask**
- **ACKRCV: Boot Operation Acknowledge Received Interrupt Mask**
- **ACKRCVE: Boot Operation Acknowledge Error Interrupt Mask**
- **OVRE: Overrun Interrupt Mask**
- **UNRE: Underrun Interrupt Mask**

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

### 39.14.16HSMCI Configuration Register

**Name:** HSMCI\_CFG

**Address:** 0x40080054

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in “[HSMCI Write Protect Mode Register](#)” on page 997.

- **FIFOMODE: HSMCI Internal FIFO control mode**

0 = A write transfer starts when a sufficient amount of data is written into the FIFO.

When the block length is greater than or equal to 3/4 of the HSMCI internal FIFO size, then the write transfer starts as soon as half the FIFO is filled. When the block length is greater than or equal to half the internal FIFO size, then the write transfer starts as soon as one quarter of the FIFO is filled. In other cases, the transfer starts as soon as the total amount of data is written in the internal FIFO.

1 = A write transfer starts as soon as one data is written into the FIFO.

- **FERRCTRL: Flow Error flag reset control mode**

0= When an underflow/overflow condition flag is set, a new Write/Read command is needed to reset the flag.

1= When an underflow/overflow condition flag is set, a read status resets the flag.

- **HSMODE: High Speed Mode**

0= Default bus timing mode.

1= If set to one, the host controller outputs command line and data lines on the rising edge of the card clock. The Host driver shall check the high speed support in the card registers.

- **LSYNC: Synchronize on the last block**

0= The pending command is sent at the end of the current data block.

1= The pending command is sent at the end of the block transfer when the transfer length is not infinite. (block count shall be different from zero)

### 39.14.17 HSMCI Write Protect Mode Register

**Name:** HSMCI\_WPMR

**Address:** 0x400800E4

**Access:** Read-write

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protection if WPKEY corresponds to 0x4D4349 ("MCI" in ASCII).

1 = Enables the Write Protection if WPKEY corresponds to 0x4D4349 ("MCI" in ASCII).

Protects the registers:

- ["HSMCI Mode Register" on page 974](#)
- ["HSMCI Data Timeout Register" on page 976](#)
- ["HSMCI SDCard/SDIO Register" on page 977](#)
- ["HSMCI Completion Signal Timeout Register" on page 982](#)
- ["HSMCI Configuration Register" on page 996](#)

- **WPKEY: Write Protect Key**

Value	Name	Description
0x4D4349	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 39.14.18HSMCI Write Protect Status Register

**Name:** HSMCI\_WPSR

**Address:** 0x400800E8

**Access:** Read-only

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

- **WPVS: Write Protection Violation Status**

Value	Name	Description
0	NONE	No Write Protection Violation occurred since the last read of this register (HSMCI_WPSR)
1	WRITE	Write Protection detected unauthorized attempt to write a control register had occurred (since the last read.)
2	RESET	Software reset had been performed while Write Protection was enabled (since the last read).
3	BOTH	Both Write Protection violation and software reset with Write Protection enabled have occurred since the last read.

- **WPVSR: Write Protection Violation SouRCe**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

## 40. Pulse Width Modulation Controller (PWM)

### 40.1 Description

The PWM macrocell controls 4 channels independently. Each channel controls two complementary square output waveforms. Characteristics of the output waveforms such as period, duty-cycle, polarity and dead-times (also called dead-bands or non-overlapping times) are configured through the user interface. Each channel selects and uses one of the clocks provided by the clock generator. The clock generator provides several clocks resulting from the division of the PWM master clock (MCK).

Each channel includes a register in order to generate an additional edge of the output waveform in addition of those resulting of the duty cycle value.

All PWM macrocell accesses are made through registers mapped on the peripheral bus. All channels integrate a double buffering system in order to prevent an unexpected output waveform while modifying the period, the spread spectrum, the duty-cycle, the additional edge register or the dead-times.

Channels can be linked together as synchronous channels to be able to update their duty-cycle or dead-times at the same time.

The update of duty-cycles of synchronous channels can be performed by the Peripheral DMA Controller Channel (PDC) which offers buffer transfer without processor Intervention.

The PWM macrocell includes a spread-spectrum counter to allow a constantly varying period (only for Channel 0). This counter may be useful to minimize electromagnetic interference or to reduce the acoustic noise of a PWM driven motor.

The PWM macrocell provides 8 independent comparison units capable of comparing a programmed value to the counter of the synchronous channels (counter of channel 0). These comparisons are intended to generate software interrupts, to trigger pulses on the 2 independent event lines (in order to synchronize ADC conversions with a lot of flexibility independently of the PWM outputs) and to trigger PDC transfer requests.

The PWM outputs can be overridden synchronously or asynchronously to their channel counter.

The PWM block provides a fault protection mechanism with 8 fault inputs, capable to detect a fault condition and to override the PWM outputs asynchronously (outputs forced to 0, 1 or Hi-Z).

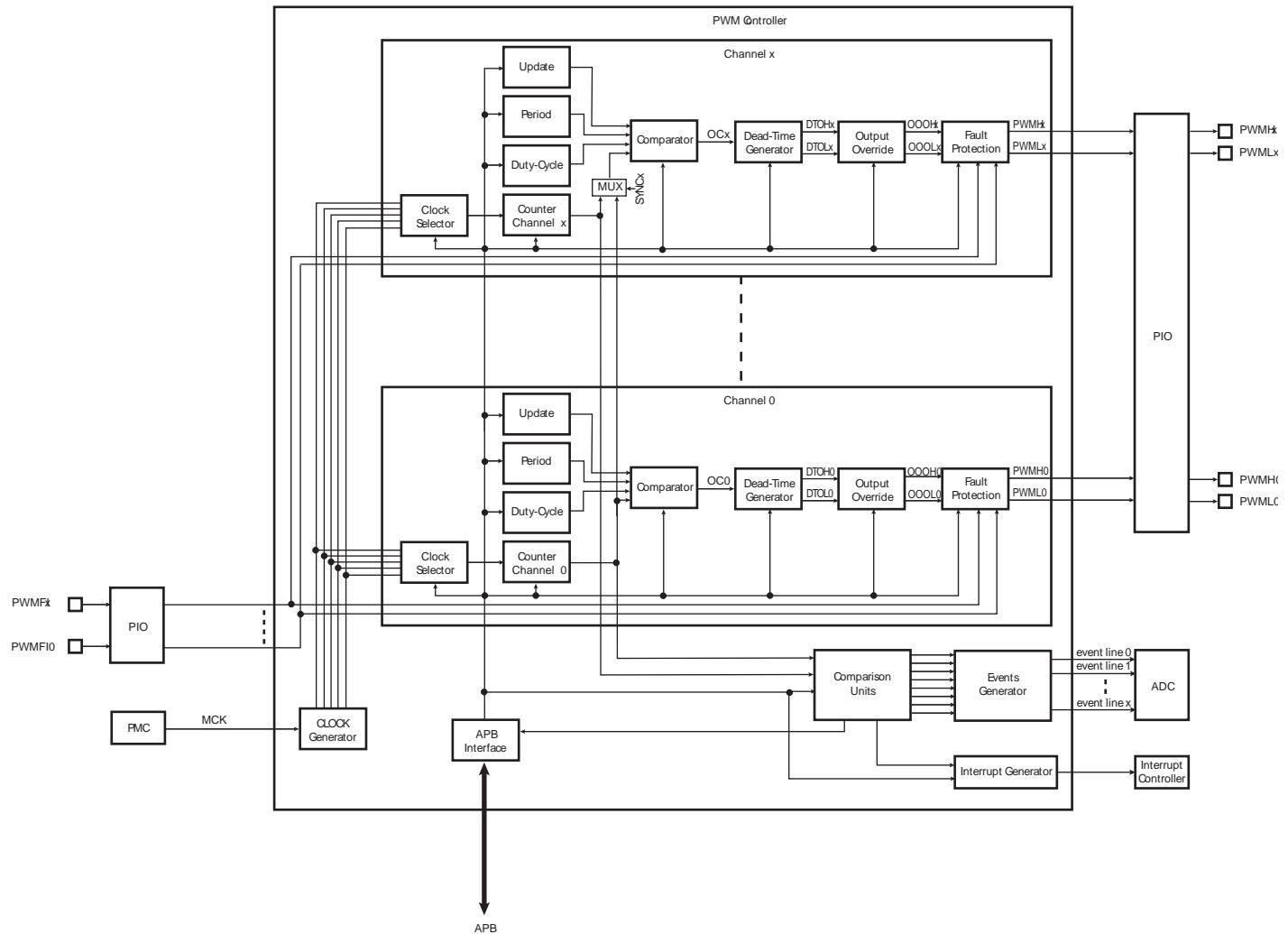
For safety usage, some configuration registers are write-protected.

## 40.2 Embedded Characteristics

- 4 Channels
- Common Clock Generator Providing Thirteen Different Clocks
  - A Modulo n Counter Providing Eleven Clocks
  - Two Independent Linear Dividers Working on Modulo n Counter Outputs
- Independent Channels
  - Independent 16-bit Counter for Each Channel
  - Independent Complementary Outputs with 12-bit Dead-Time Generator (also called Dead-Band or Non-Overlapping Time) for Each Channel
  - Independent Enable Disable Command for Each Channel
  - Independent Clock Selection for Each Channel
  - Independent Period, Duty-Cycle and Dead-Time for Each Channel
  - Independent Double Buffering of Period, Duty-Cycle and Dead-Times for Each Channel
  - Independent Programmable Selection of The Output Waveform Polarity for Each Channel, with Double Buffering
  - Independent Programmable Center or Left Aligned Output Waveform for Each Channel
  - Independent Additional Edge Value for Each Channel, with Double Buffering, in Order to Generate Additional Edges of the Output Waveform
  - Independent Output Override for Each Channel
  - Independent Interrupt for Each Channel, at Each Period for Left-Aligned or Center-Aligned Configuration or at Each Half-Period for Center-Aligned Configuration
  - Independent Update Time Selection of Double Buffering Registers (Polarity, Duty Cycle and Additional Edge Value) for Each Channel, at Each Period for Left-Aligned or Center-Aligned Configuration, at each Half-Period for Center-Aligned Configuration
- Two 2-bit Gray Up/Down Channels for Stepper Motor Control
- Spread Spectrum Counter to allow a constantly varying Duty Cycle (only for Channel 0)
- Synchronous Channel Mode
  - Synchronous Channels share the same Counter
  - Mode to Update the Synchronous Channels Registers after a Programmable Number of Periods
  - Synchronous Channels Supports Connection of one Peripheral DMA Controller Channel (PDC) which offers Buffer Transfer without Processor Intervention to update Duty-Cycle Registers
- 2 Independent Events Lines Intended to Synchronize ADC Conversions
  - Programmable delay for Events Lines to delay ADC measurements
- 8 Comparison Units Intended to Generate Interrupts, Pulses on Event Lines and PDC Transfer Requests
- 8 Programmable Fault/Break Inputs Providing an Asynchronous Protection of PWM Outputs
  - 1 User Driven through PIO inputs
  - PMC Driven when Crystal Oscillator Clock Fails
  - ADC Controller Driven through Configurable Comparison Function
  - Analog Comparator Controller Driven
  - Timer/Counter Driven through Configurable Comparison Function
- Write Protected Registers

## 40.3 Block Diagram

Figure 40-1. Pulse Width Modulation Controller Block Diagram



## 40.4 I/O Lines Description

Each channel outputs two complementary external I/O lines.

Table 40-1. I/O Line Description

Name	Description	Type
PWMHx	PWM Waveform Output High for channel x	Output
PWMLx	PWM Waveform Output Low for channel x	Output
PWMFIx	PWM Fault Input x	Input

## 40.5 Product Dependencies

### 40.5.1 I/O Lines

The pins used for interfacing the PWM are multiplexed with PIO lines. The programmer must first program the PIO controller to assign the desired PWM pins to their peripheral function. If I/O lines of the PWM are not used by the application, they can be used for other purposes by the PIO controller.

All of the PWM outputs may or may not be enabled. If an application requires only four channels, then only four PIO lines will be assigned to PWM outputs.

Table 40-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
PWM	PWMFI0	PA9	C
PWM	PWMH0	PA0	A
PWM	PWMH0	PA11	B
PWM	PWMH0	PA23	B
PWM	PWMH0	PB0	A
PWM	PWMH0	PC18	B
PWM	PWMH0	PD20	A
PWM	PWMH1	PA1	A
PWM	PWMH1	PA12	B
PWM	PWMH1	PA24	B
PWM	PWMH1	PB1	A
PWM	PWMH1	PC19	B
PWM	PWMH1	PD21	A
PWM	PWMH2	PA2	A
PWM	PWMH2	PA13	B
PWM	PWMH2	PA25	B
PWM	PWMH2	PB4	B
PWM	PWMH2	PC20	B
PWM	PWMH2	PD22	A
PWM	PWMH3	PA7	B
PWM	PWMH3	PA14	B
PWM	PWMH3	PA17	C
PWM	PWMH3	PB14	B
PWM	PWMH3	PC21	B
PWM	PWMH3	PD23	A
PWM	PWML0	PA19	B
PWM	PWML0	PB5	B
PWM	PWML0	PC0	B
PWM	PWML0	PC13	B
PWM	PWML0	PD24	A
PWM	PWML1	PA20	B
PWM	PWML1	PB12	A
PWM	PWML1	PC1	B
PWM	PWML1	PC15	B

**Table 40-2. I/O Lines**

PWM	PWML1	PD25	A
PWM	PWML2	PA16	C
PWM	PWML2	PA30	A
PWM	PWML2	PB13	A
PWM	PWML2	PC2	B
PWM	PWML2	PD26	A
PWM	PWML3	PA15	C
PWM	PWML3	PC3	B
PWM	PWML3	PC22	B
PWM	PWML3	PD27	A

#### 40.5.2 Power Management

The PWM is not continuously clocked. The programmer must first enable the PWM clock in the Power Management Controller (PMC) before using the PWM. However, if the application does not require PWM operations, the PWM clock can be stopped when not needed and be restarted later. In this case, the PWM will resume its operations where it left off.

In the PWM description, Master Clock (MCK) is the clock of the peripheral bus to which the PWM is connected.

#### 40.5.3 Interrupt Sources

The PWM interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the PWM interrupt requires the Interrupt Controller to be programmed first. Note that it is not recommended to use the PWM interrupt line in edge sensitive mode.

**Table 40-3. Peripheral IDs**

Instance	ID
PWM	36

#### 40.5.4 Fault Inputs

The PWM has the FAULT inputs connected to the different modules. Please refer to the implementation of these module within the product for detailed information about the fault generation procedure. The PWM receives faults from PIO inputs, PMC, ADC controller, Analog Comparator Controller and Timer/Counters.

**Table 40-4. Fault Inputs**

Fault Inputs	External PWM Fault Input Number	Polarity Level <sup>(1)</sup>	Fault Input ID
PA9	PWMFI0	User Defined	0
Main OSC (PMC)	–	1	1
AFEC0-1	–	1	2
ACC	–	1	3
Timer0	–	1	4
Timer1	–	1	5
Timer2	–	1	6
Temperature Sensor	–	1	7

Note: 1. FPOL bit in PWMC\_FMR.

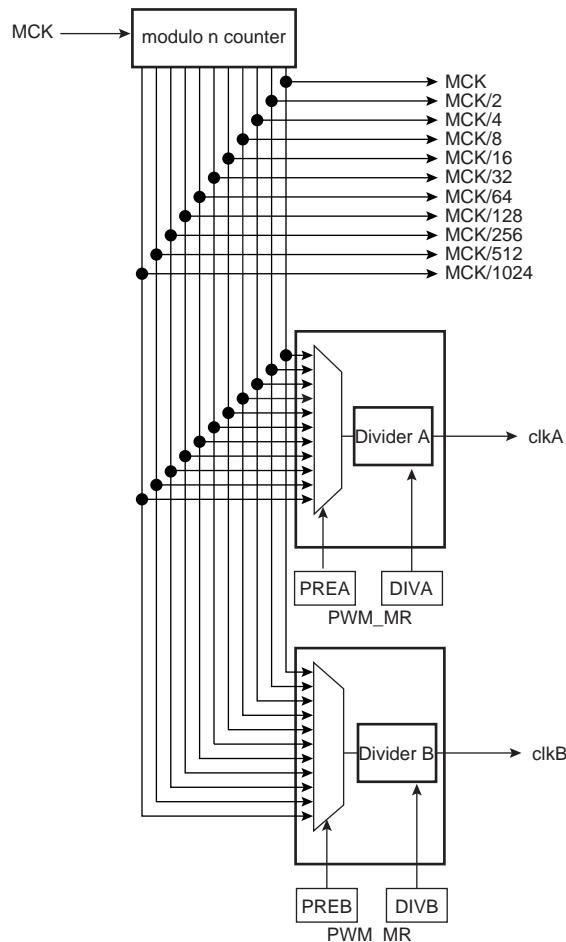
## 40.6 Functional Description

The PWM macrocell is primarily composed of a clock generator module and 4 channels.

- Clocked by the master clock (MCK), the clock generator module provides 13 clocks.
- Each channel can independently choose one of the clock generator outputs.
- Each channel generates an output waveform with attributes that can be defined independently for each channel through the user interface registers.

### 40.6.1 PWM Clock Generator

Figure 40-2. Functional View of the Clock Generator Block Diagram



The PWM master clock (MCK) is divided in the clock generator module to provide different clocks available for all channels. Each channel can independently select one of the divided clocks.

The clock generator is divided in three blocks:

- A modulo n counter which provides 11 clocks:  $F_{MCK}$ ,  $F_{MCK}/2$ ,  $F_{MCK}/4$ ,  $F_{MCK}/8$ ,  $F_{MCK}/16$ ,  $F_{MCK}/32$ ,  $F_{MCK}/64$ ,  $F_{MCK}/128$ ,  $F_{MCK}/256$ ,  $F_{MCK}/512$ ,  $F_{MCK}/1024$
- Two linear dividers (1, 1/2, 1/3, ... 1/255) that provide two separate clocks: clkA and clkB

Each linear divider can independently divide one of the clocks of the modulo n counter. The selection of the clock to be divided is made according to the PREA (PREB) field of the PWM Clock register (PWM\_CLK). The resulting clock clkA (clkB) is the clock selected divided by DIVA (DIVB) field value.

After a reset of the PWM controller, DIVA (DIVB) and PREA (PREB) are set to 0. This implies that after reset clkA (clkB) are turned off.

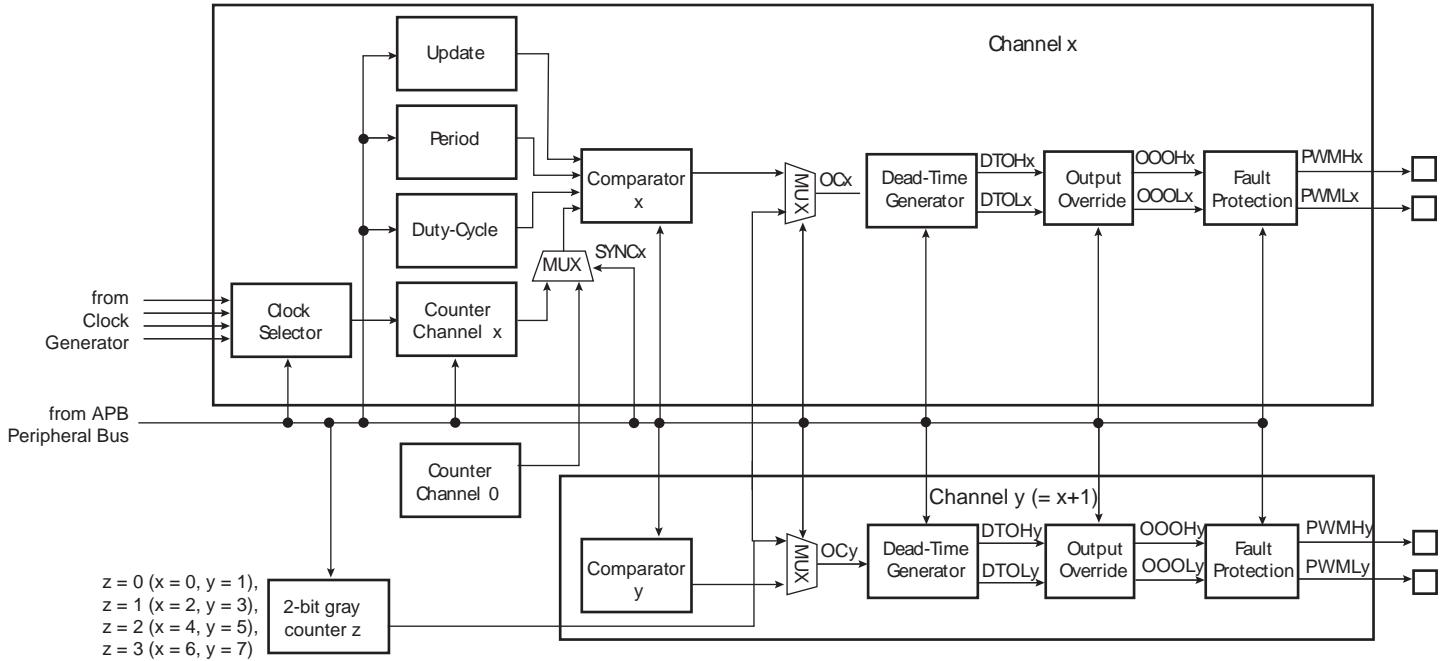
At reset, all clocks provided by the modulo n counter are turned off except clock "MCK". This situation is also true when the PWM master clock is turned off through the Power Management Controller.

**CAUTION:** Before using the PWM macrocell, the programmer must first enable the PWM clock in the Power Management Controller (PMC).

## 40.6.2 PWM Channel

### 40.6.2.1 Channel Block Diagram

Figure 40-3. Functional View of the Channel Block Diagram



Each of the 4 channels is composed of six blocks:

- A clock selector which selects one of the clocks provided by the clock generator (described in [Section 40.6.1 on page 1004](#)).
- A counter clocked by the output of the clock selector. This counter is incremented or decremented according to the channel configuration and comparators matches. The size of the counter is 16 bits.
- A comparator used to compute the OCx output waveform according to the counter value and the configuration. The counter value can be the one of the channel counter or the one of the channel 0 counter according to SYNCx bit in the [“PWM Sync Channels Mode Register” on page 1044 \(PWM\\_SC\)](#).
- A 2-bit configurable gray counter enables the stepper motor driver. One gray counter drives 2 channels.
- A dead-time generator providing two complementary outputs (DTOHx/DTOLx) which allows to drive external power control switches safely.
- An output override block that can force the two complementary outputs to a programmed value (OOOHx/OOOLx).
- An asynchronous fault protection mechanism that has the highest priority to override the two complementary outputs (PWMHx/PWMLx) in case of fault detection (outputs forced to 0, 1 or Hi-Z).

#### 40.6.2.2 Comparator

The comparator continuously compares its counter value with the channel period defined by CPRD in the “[PWM Channel Period Register](#)” on page 1079 (PWM\_CPRDx) and the duty-cycle defined by CDTY in the “[PWM Channel Duty Cycle Register](#)” on page 1077 (PWM\_CDTYx) to generate an output signal OCx accordingly.

The different properties of the waveform of the output OCx are:

- The **clock selection**. The channel counter is clocked by one of the clocks provided by the clock generator described in the previous section. This channel parameter is defined in the CPRE field of the “[PWM Channel Mode Register](#)” on page 1075 (PWM\_CMRx). This field is reset at 0.
- The **waveform period**. This channel parameter is defined in the CPRD field of the PWM\_CPRDx register. If the waveform is left aligned, then the output waveform period depends on the counter source clock and can be calculated:  
By using the PWM master clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024), the resulting period formula will be:

$$\frac{(X \times CPRD)}{MCK}$$

By using the PWM master clock (MCK) divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(CRPD \times DIVA)}{MCK} \text{ or } \frac{(CRPD \times DIVB)}{MCK}$$

If the waveform is center aligned then the output waveform period depends on the counter source clock and can be calculated:

By using the PWM master clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(2 \times X \times CPRD)}{MCK}$$

By using the PWM master clock (MCK) divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(2 \times CRPD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CRPD \times DIVB)}{MCK}$$

- The **waveform duty-cycle**. This channel parameter is defined in the CDTY field of the PWM\_CDTYx register. If the waveform is left aligned then:

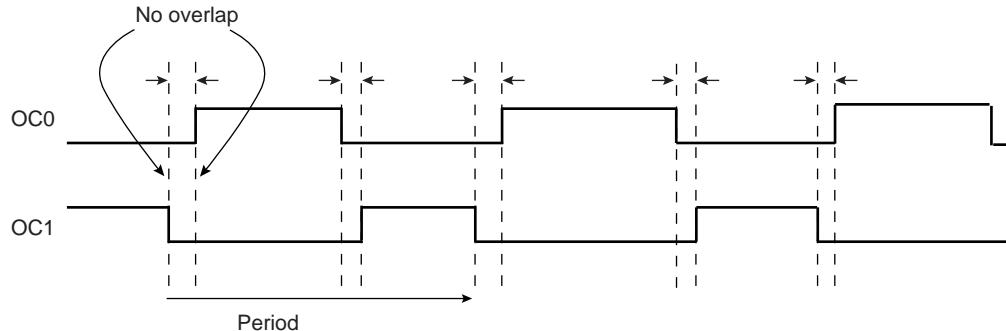
$$\text{duty cycle} = \frac{(\text{period} - 1/\text{fchannel\_x\_clock} \times \text{CDTY})}{\text{period}}$$

If the waveform is center aligned, then:

$$\text{duty cycle} = \frac{((\text{period}/2) - 1/\text{fchannel\_x\_clock} \times \text{CDTY}))}{(\text{period}/2)}$$

- The **waveform polarity**. At the beginning of the period, the signal can be at high or low level. This property is defined in the CPOL field of the PWM\_CMRx register. By default the signal starts by a low level.
- The **waveform alignment**. The output waveform can be left or center aligned. Center aligned waveforms can be used to generate non overlapped waveforms. This property is defined in the CALG field of the PWM\_CMRx register. The default mode is left aligned.

**Figure 40-4. Non Overlapped Center Aligned Waveforms**



Note: 1. See [Figure 40-5 on page 1009](#) for a detailed description of center aligned waveforms.

When center aligned, the channel counter increases up to CPRD and decreases down to 0. This ends the period.

When left aligned, the channel counter increases up to CPRD and is reset. This ends the period.

Thus, for the same CPRD value, the period for a center aligned channel is twice the period for a left aligned channel.

Waveforms are fixed at 0 when:

- CDTY = CPRD and CPOL = 0
- CDTY = 0 and CPOL = 1

Waveforms are fixed at 1 (once the channel is enabled) when:

- CDTY = 0 and CPOL = 0
- CDTY = CPRD and CPOL = 1

The waveform polarity must be set before enabling the channel. This immediately affects the channel output level.

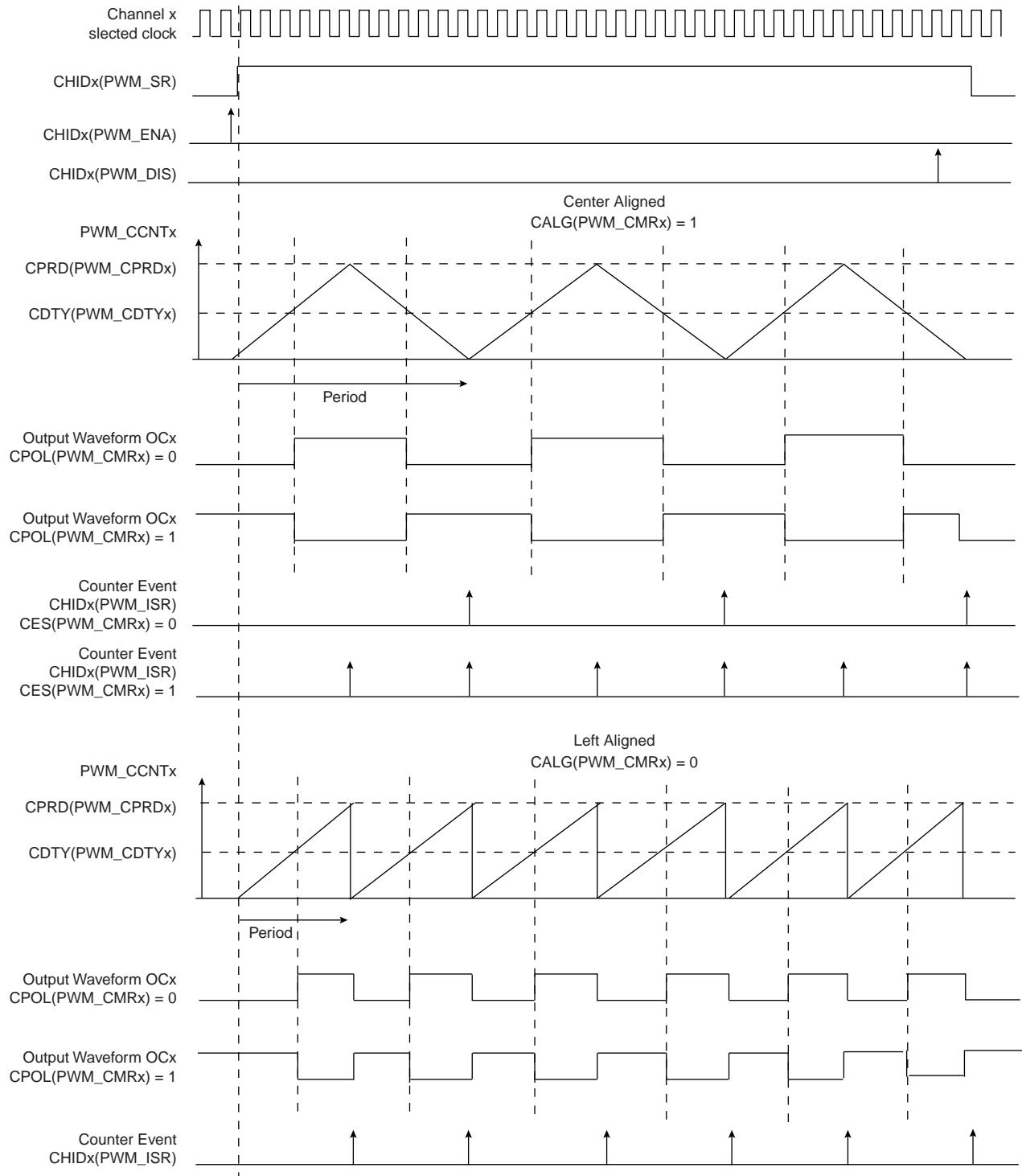
Modifying CPOL in [“PWM Channel Mode Register” on page 1075](#) while the channel is enabled can lead to an unexpected behavior of the device being driven by PWM.

When a channel is enabled, the waveform polarity can be modified at any time by re-programming the bit CPOL, then the waveform polarity is updated immediately. To modify the waveform polarity only at the next PWM period (not immediately), the user must use the bit CPOLUP or the bit CPOLINVUP in [“PWM Comparison x Mode Update Register” on page 1074](#) instead of bit CPOL in [“PWM Channel Mode Register” on page 1075](#).

Besides generating output signals OC<sub>x</sub>, the comparator generates interrupts in function of the counter value. When the output waveform is left aligned, the interrupt occurs at the end of the counter period. When the output waveform is center aligned, the bit CES of the PWM\_CMRx register defines when the channel counter interrupt occurs. If CES is set to 0, the interrupt occurs at the end of the counter period. If CES is set to 1, the interrupt occurs at the end of the counter period and at half of the counter period.

[Figure 40-5 “Waveform Properties”](#) illustrates the counter interrupts in function of the configuration.

**Figure 40-5. Waveform Properties**



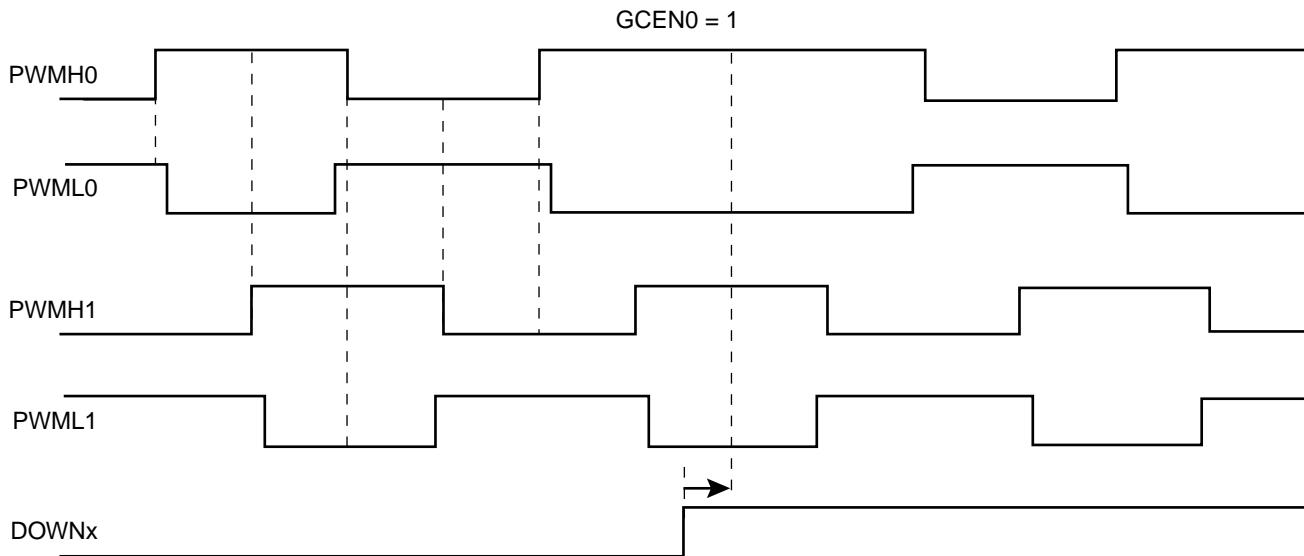
#### 40.6.2.3 2-bit Gray Up/Down Counter for Stepper Motor

It is possible to configure a couple of channels to provide a 2-bit gray count waveform on 2 outputs. Dead-Time Generator and other downstream logic can be configured on these channels.

Up or down count mode can be configured on-the-fly by means of PWM\_SMMR configuration registers.

When GCEN0 is set to 1, channels 0 and 1 outputs are driven with gray counter.

**Figure 40-6. 2-bit Gray Up/Down Counter**



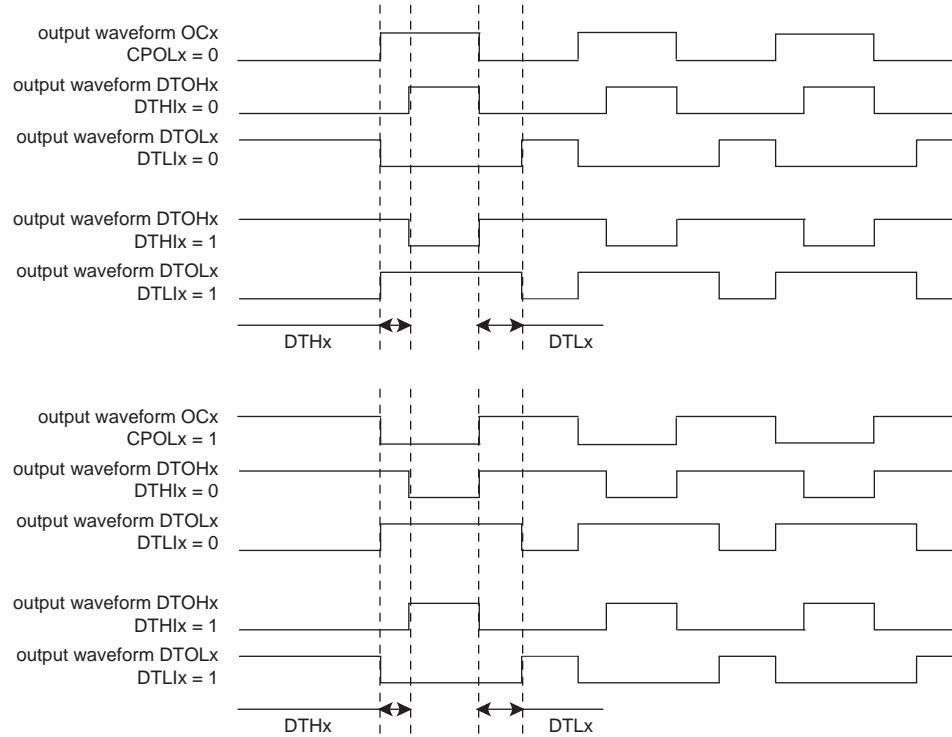
#### 40.6.2.4 Dead-Time Generator

The dead-time generator uses the comparator output OCx to provide the two complementary outputs DTOHx and DTOLx, which allows the PWM macrocell to drive external power control switches safely. When the dead-time generator is enabled by setting the bit DTE to 1 or 0 in the “[PWM Channel Mode Register](#)” (PWM\_CMRx), dead-times (also called dead-bands or non-overlapping times) are inserted between the edges of the two complementary outputs DTOHx and DTOLx. Note that enabling or disabling the dead-time generator is allowed only if the channel is disabled.

The dead-time is adjustable by the “[PWM Channel Dead Time Register](#)” (PWM\_DTx). Both outputs of the dead-time generator can be adjusted separately by DTH and DTL. The dead-time values can be updated synchronously to the PWM period by using the “[PWM Channel Dead Time Update Register](#)” (PWM\_DTUPDx).

The dead-time is based on a specific counter which uses the same selected clock that feeds the channel counter of the comparator. Depending on the edge and the configuration of the dead-time, DTOHx and DTOLx are delayed until the counter has reached the value defined by DTH or DTL. An inverted configuration bit (DTHI and DTLI bit in the PWM\_CMRx register) is provided for each output to invert the dead-time outputs. The following figure shows the waveform of the dead-time generator.

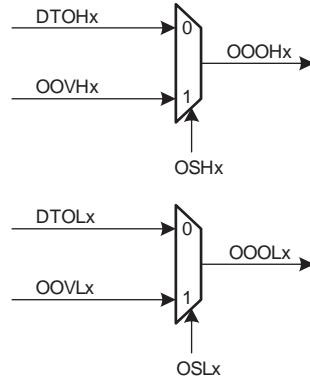
**Figure 40-7. Complementary Output Waveforms**



#### 40.6.2.5 Output Override

The two complementary outputs DTOHx and DTOLx of the dead-time generator can be forced to a value defined by the software.

**Figure 40-8. Override Output Selection**



The fields OSHx and OSLx in the “[PWM Output Selection Register](#)” (PWM\_OS) allow the outputs of the dead-time generator DTOHx and DTOLx to be overridden by the value defined in the fields OOVHx and OOVLx in the “[PWM Output Override Value Register](#)” (PWM\_OOV).

The set registers “[PWM Output Selection Set Register](#)” and “[PWM Output Selection Set Update Register](#)” (PWM\_OSS and PWM\_OSSUPD) enable the override of the outputs of a channel regardless of other channels. In the same way, the clear registers “[PWM Output Selection Clear Register](#)” and “[PWM Output Selection Clear Update Register](#)” (PWM\_OSC and PWM\_OSCUPD) disable the override of the outputs of a channel regardless of other channels.

By using buffer registers PWM\_OSSUPD and PWM\_OSCUPD, the output selection of PWM outputs is done synchronously to the channel counter, at the beginning of the next PWM period.

By using registers PWM\_OSS and PWM\_OSC, the output selection of PWM outputs is done asynchronously to the channel counter, as soon as the register is written.

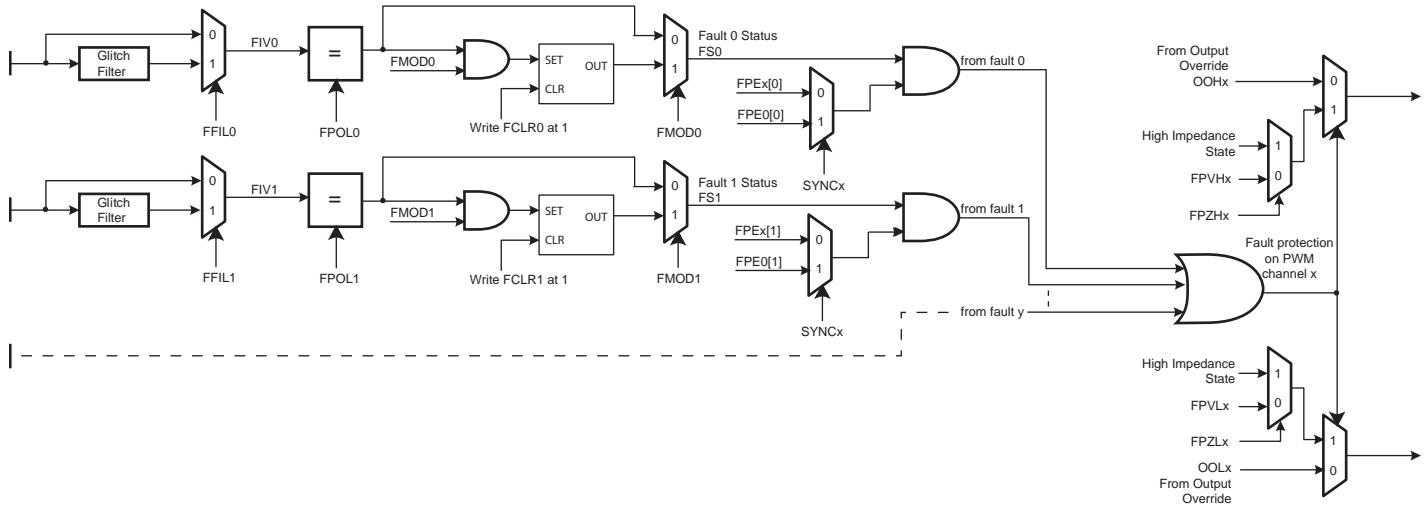
The value of the current output selection can be read in PWM\_OS.

While overriding PWM outputs, the channel counters continue to run, only the PWM outputs are forced to user defined values.

#### 40.6.2.6 Fault Protection

8 inputs provide fault protection which can force any of the PWM output pair to a programmable value. This mechanism has priority over output overriding.

**Figure 40-9. Fault Protection**



The polarity level of the fault inputs is configured by the FPOL field in the “[PWM Fault Mode Register](#)” (PWM\_FMR). For fault inputs coming from internal peripherals such as ADC, Timer Counter, to name but a few, the polarity level must be FPOL = 1. For fault inputs coming from external GPIO pins the polarity level depends on the user's implementation.

The configuration of the Fault Activation Mode (FMOD bit in PWMC\_FMR) depends on the peripheral generating the fault. If the corresponding peripheral does not have “Fault Clear” management, then the FMOD configuration to use must be FMOD = 1, to avoid spurious fault detection. Check the corresponding peripheral documentation for details on handling fault generation.

The fault inputs can be glitch filtered or not in function of the FFIL field in the PWM\_FMR register. When the filter is activated, glitches on fault inputs with a width inferior to the PWM master clock (MCK) period are rejected.

A fault becomes active as soon as its corresponding fault input has a transition to the programmed polarity level. If the corresponding bit FMOD is set to 0 in the PWM\_FMR register, the fault remains active as long as the fault input is at this polarity level. If the corresponding FMOD bit is set to 1, the fault remains active until the fault input is not at this polarity level anymore and until it is cleared by writing the corresponding bit FCLR in the “[PWM Fault Clear Register](#)” (PWM\_FSCR). By reading the “[PWM Fault Status Register](#)” (PWM\_FSR), the user can read the current level of the fault inputs by means of the field FIV, and can know which fault is currently active thanks to the FS field.

Each fault can be taken into account or not by the fault protection mechanism in each channel. To be taken into account in the channel x, the fault y must be enabled by the bit FPEx[y] in the “[PWM Fault Protection Enable Registers](#)” (PWM\_FPE1). However the synchronous channels (see [Section 40.6.2.9 “Synchronous Channels”](#)) do not use their own fault enable bits, but those of the channel 0 (bits FPE0[y]).

The fault protection on a channel is triggered when this channel is enabled and when any one of the faults that are enabled for this channel is active. It can be triggered even if the PWM master clock (MCK) is not running but only by a fault input that is not glitch filtered.

When the fault protection is triggered on a channel, the fault protection mechanism resets the counter of this channel and forces the channel outputs to the values defined by the fields FPVHx and FPVLx in the “[PWM Fault Protection Value Register 1](#)” (PWM\_FPV) and fields FPZHx/FPZLx in the “[PWM Fault Protection Value Register 2](#)”, according to [Table 40-5](#). The output forcing is made asynchronously to the channel counter.

**Table 40-5. Forcing Values of PWM Outputs by Fault Protection**

FPZH/Lx	FPVH/Lx	Forcing Value of PWMH/Lx
0	0	0
0	1	1
1	–	High impedance state (Hi-Z)

**CAUTION:**

- To prevent an unexpected activation of the status flag FSy in the PWM\_FSR register, the FMODy bit can be set to “1” only if the FPOLy bit has been previously configured to its final value.
- To prevent an unexpected activation of the Fault Protection on the channel x, the bit FPEx[y] can be set to “1” only if the FPOLy bit has been previously configured to its final value.

If a comparison unit is enabled (see [Section 40.6.3 “PWM Comparison Units”](#)) and if a fault is triggered in the channel 0, in this case the comparison cannot match.

As soon as the fault protection is triggered on a channel, an interrupt (different from the interrupt generated at the end of the PWM period) can be generated but only if it is enabled and not masked. The interrupt is reset by reading the interrupt status register, even if the fault which has caused the trigger of the fault protection is kept active.

#### 40.6.2.7 Spread Spectrum Counter

The PWM macrocell includes a spread spectrum counter allowing the generation of a constantly varying duty cycle on the output PWM waveform (only for the channel 0). This feature may be useful to minimize electromagnetic interference or to reduce the acoustic noise of a PWM driven motor.

This is achieved by varying the effective period in a range defined by a spread spectrum value which is programmed by the field SPRD in the “[PWM Spread Spectrum Register](#)” (PWM\_SSPPR). The effective period of the output waveform is the value of the spread spectrum counter added to the programmed waveform period CPRD in the “[PWM Channel Period Register](#)” (PWM\_CPRO).

It will cause the effective period to vary from CPRD-SPRD to CPRD+SPRD. This leads to a constantly varying duty cycle on the PWM output waveform because the duty cycle value programmed is unchanged.

The value of the spread spectrum counter can change in two ways depending on the bit SPRDM in the “[PWM Spread Spectrum Register](#)” (PWM\_SSPPR).

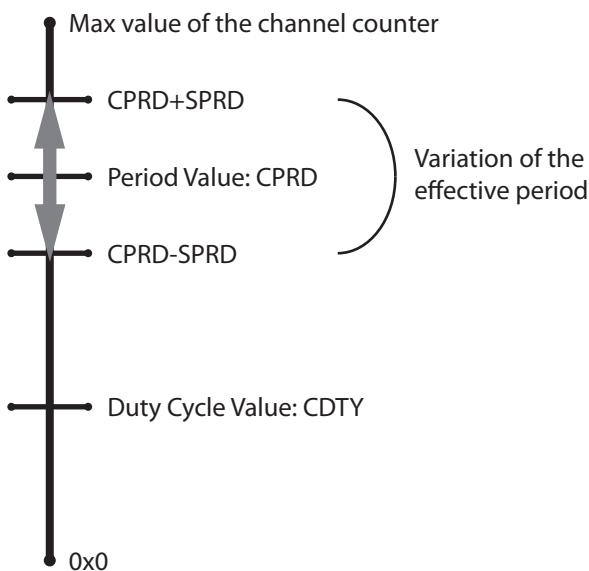
If SPRDM=0, the mode TRIANGULAR is selected:

The spread spectrum counter starts to count from -SPRD when the channel 0 is enabled or after reset and counts upwards at each period of the channel counter. When it reaches SPRD, it restarts to count from -SPRD again.

If SPRDM=1, the mode RANDOM is selected:

A new random value is assigned to the spread spectrum counter at each period of the channel counter. This random value is between -SPRD and +SPRD and is uniformly distributed.

**Figure 40-10. Spread Spectrum Counter**



#### 40.6.2.8 Additional Edges

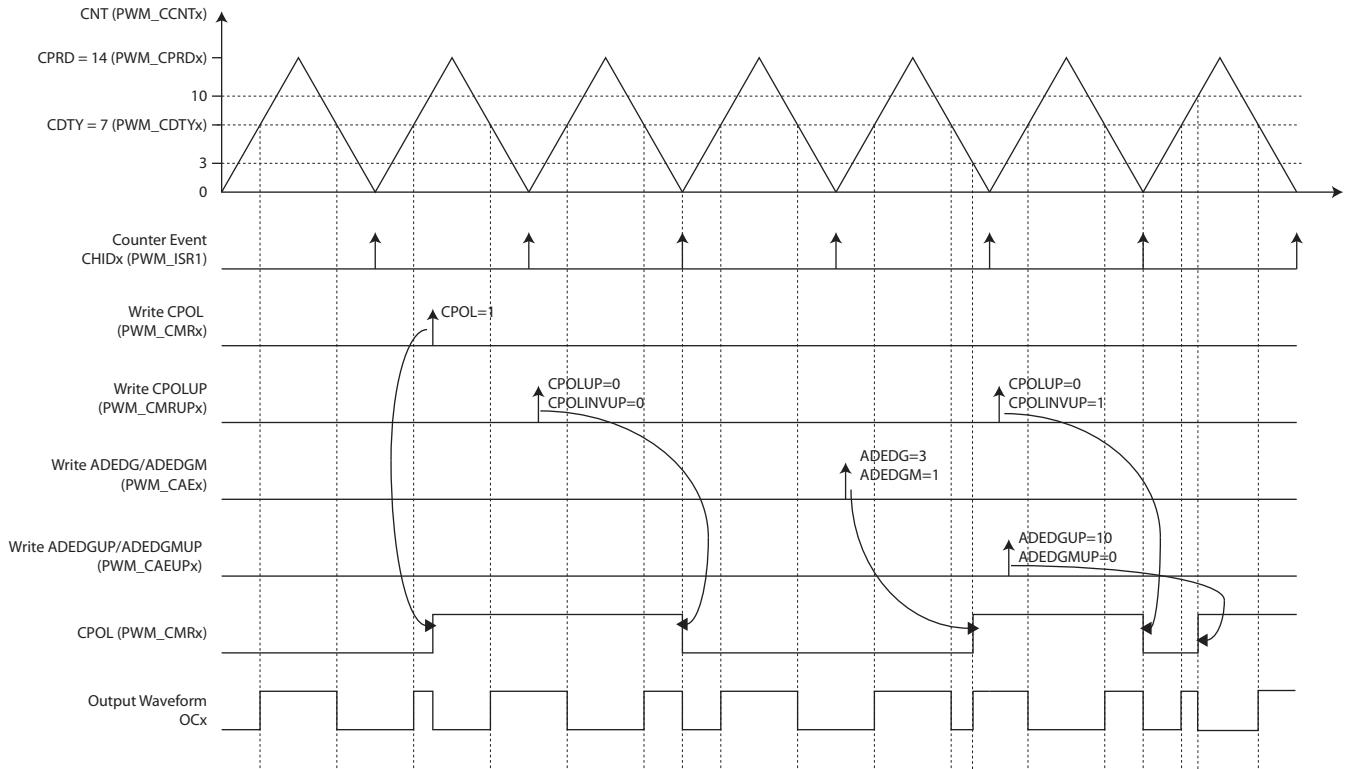
The PWM macrocell is able to generate additional edges of the channel output waveform by inverting the waveform polarity CPOL, in top of these generated by the duty cycle value.

There are several complementary ways in order to invert the polarity:

- By inverting the polarity at any time:  
write a new value for CPOL in [“PWM Channel Mode Register” on page 1075](#). The polarity of the output waveform changes immediately.
- By inverting the polarity at the next PWM period border:  
write the field CPOLUP in [“PWM Channel Mode Update Register” on page 1084](#), the polarity will be updated synchronously with the PWM period. If you want to invert the polarity whatever the current polarity is, write the bit CPOLINVUP at 1 in the same register. In this case the polarity will be inverted synchronously with the PWM period and the bit CPOLUP is not taken into account.
- By inverting the polarity at a precise moment of the PWM period:  
write the field ADEDGV and the field ADEDGM in [“PWM Channel Additional Edge Register” on page 1085](#). As soon as the channel counter reaches the value defined by ADEDGV, the polarity of the output waveform is inverted. The field ADEDGM is used when the channel is center-aligned (CALG=1 in [“PWM Channel Mode Register” on page 1075](#)), if ADEDGM=0 the additional edge occurs when the channel counter is incrementing, if ADEDGM=1 the additional edge occurs when it is decrementing, if ADEDGM=2 the additional edge occurs whether the counter is incrementing or not.
- By inverting the polarity at a precise moment of the next PWM period:  
write the field ADEDGVUP and the field ADEDGMUP in [“PWM Channel Additional Edge Update Register” on page 1086](#). As soon as the channel counter reaches the value defined by ADEDGVUP at the next PWM period, the polarity of the output waveform is inverted. The field ADEDGMUP is used when the channel is center-aligned (CALG=1 in [“PWM Channel Mode Register” on page 1075](#)), if ADEDGMUP=0 the additional edge occurs when the channel counter is incrementing, if ADEDGMUP=1 the additional edge occurs when it is decrementing, if ADEDGMUP=2 the additional edge occurs whether the counter is incrementing or not.

[Figure 40-11 on page 1017](#) illustrates various ways to insert additional edges on the channel waveform by inverting the waveform polarity.

**Figure 40-11. Inserting additional edges on the output waveform**



#### 40.6.2.9 Synchronous Channels

Some channels can be linked together as synchronous channels. They have the same source clock, the same period, the same alignment and are started together. In this way, their counters are synchronized together.

The synchronous channels are defined by the SYNCx bits in the “[PWM Sync Channels Mode Register](#)” (PWM\_SCM). Only one group of synchronous channels is allowed.

When a channel is defined as a synchronous channel, the channel 0 is automatically defined as a synchronous channel too, because the channel 0 counter configuration is used by all the synchronous channels.

If a channel x is defined as a synchronous channel, it uses the following configuration fields of the channel 0 instead of its own:

- CPRE0 field in PWM\_CMRO register instead of CPREx field in PWM\_CMRx register (same source clock)
- CPRD0 field in PWM\_CMRO register instead of CPRDx field in PWM\_CMRx register (same period)
- CALG0 field in PWM\_CMRO register instead of CALGx field in PWM\_CMRx register (same alignment)

Thus writing these fields of a synchronous channel has no effect on the output waveform of this channel (except channel 0 of course).

Because counters of synchronous channels must start at the same time, they are all enabled together by enabling the channel 0 (by the CHID0 bit in PWM\_ENA register). In the same way, they are all disabled together by disabling channel 0 (by the CHID0 bit in PWM\_DIS register). However, a synchronous channel x different from channel 0 can be enabled or disabled independently from others (by the CHIDx bit in PWM\_ENA and PWM\_DIS registers).

Defining a channel as a synchronous channel while it is an asynchronous channel (by writing the bit SYNCx to 1 while it was at 0) is allowed only if the channel is disabled at this time (CHIDx = 0 in PWM\_SR register). In the same way, defining a channel as an asynchronous channel while it is a synchronous channel (by writing the SYNCx bit to 0 while it was 1) is allowed only if the channel is disabled at this time.

The field UPDM (Update Mode) in the PWM\_SCM register allow to select one of the three methods to update the registers of the synchronous channels:

- Method 1 (UPDM = 0): the period value, the duty-cycle values and the dead-time values must be written by the CPU in their respective update registers (respectively PWM\_CPRDUPDx, PWM\_CDTYUPDx and PWM\_DTUPDx). The update is triggered at the next PWM period as soon as the bit UPDULOCK in the “[PWM Sync Channels Update Control Register](#)” (PWM\_SCUC) is set to 1 (see “[Method 1: Manual write of duty-cycle values and manual trigger of the update](#)” on page 1020).
- Method 2 (UPDM = 1): the period value, the duty-cycle values, the dead-time values and the update period value must be written by the CPU in their respective update registers (respectively PWM\_CPRDUPDx, PWM\_CDTYUPDx and PWM\_DTUPD). The update of the period value and of the dead-time values is triggered at the next PWM period as soon as the bit UPDULOCK in the “[PWM Sync Channels Update Control Register](#)” (PWM\_SCUC) is set to 1. The update of the duty-cycle values and the update period value is triggered automatically after an update period defined by the field UPR in the “[PWM Sync Channels Update Period Register](#)” (PWM\_SCUP) (see “[Method 2: Manual write of duty-cycle values and automatic trigger of the update](#)” on page 1021).
- Method 3 (UPDM = 2): same as Method 2 apart from the fact that the duty-cycle values of ALL synchronous channels are written by the Peripheral DMA Controller (PDC) (see “[Method 3: Automatic write of duty-cycle values and automatic trigger of the update](#)” on page 1023). The user can choose to synchronize the PDC transfer request with a comparison match (see [Section 40.6.3 “PWM Comparison Units”](#)), by the fields PTRM and PTRCS in the PWM\_SCM register.

**Table 40-6. Summary of the Update of Registers of Synchronous Channels**

	UPDM=0	UPDM=1	UPDM=2
Period Value (PWM_CPRDUPDx)	Write by the CPU		
	Update is triggered at the next PWM period as soon as the bit UPDULOCK is set to 1		
Dead-Time Values (PWM_DTUPDx)	Write by the CPU		
	Update is triggered at the next PWM period as soon as the bit UPDULOCK is set to 1		
Duty-Cycle Values (PWM_CDTYUPDx)	Write by the CPU	Write by the CPU	Write by the PDC
	Update is triggered at the next PWM period as soon as the bit UPDULOCK is set to 1	Update is triggered at the next PWM period as soon as the update period counter has reached the value UPR	
Update Period Value (PWM_SCUPUPD)	Not applicable	Write by the CPU	
	Not applicable	Update is triggered at the next PWM period as soon as the update period counter has reached the value UPR	

### Method 1: Manual write of duty-cycle values and manual trigger of the update

In this mode, the update of the period value, the duty-cycle values and the dead-time values must be done by writing in their respective update registers with the CPU (respectively PWM\_CPRDUPDx, PWM\_CDTYUPDx and PWM\_DTUPDx).

To trigger the update, the user must use the bit UPDULOCK of the “[PWM Sync Channels Update Control Register](#)” (PWM\_SCUC) which allows to update synchronously (at the same PWM period) the synchronous channels:

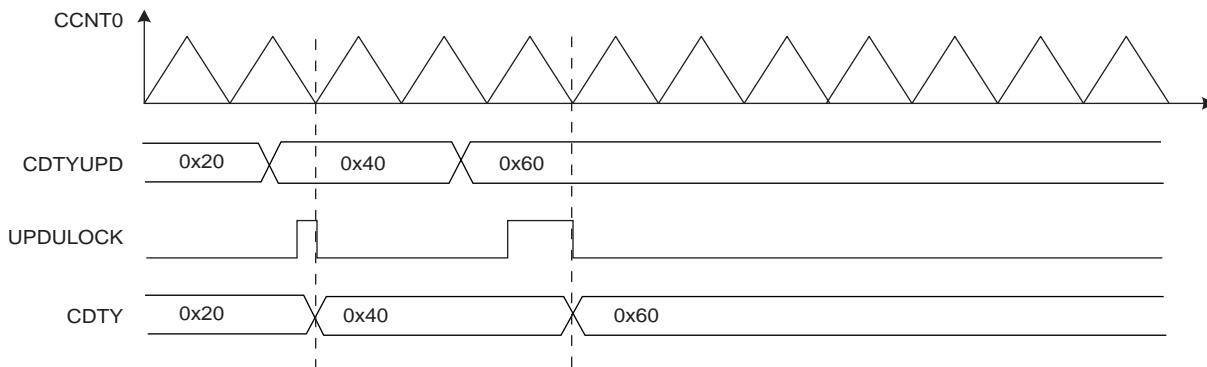
- If the bit UPDULOCK is set to 1, the update is done at the next PWM period of the synchronous channels.
- If the UPDULOCK bit is not set to 1, the update is locked and cannot be performed.

After writing the UPDULOCK bit to 1, it is held at this value until the update occurs, then it is read 0.

Sequence for Method 1:

1. Select the manual write of duty-cycle values and the manual update by setting the UPDM field to 0 in the PWM\_SCM register
2. Define the synchronous channels by the SYNCx bits in the PWM\_SCM register.
3. Enable the synchronous channels by writing CHID0 in the PWM\_ENA register.
4. If an update of the period value and/or the duty-cycle values and/or the dead-time values is required, write registers that need to be updated (PWM\_CPRDUPDx, PWM\_CDTYUPDx and PWM\_DTUPDx).
5. Set UPDULOCK to 1 in PWM\_SCUC.
6. The update of the registers will occur at the beginning of the next PWM period. At this moment the UPDULOCK bit is reset, go to Step 4.) for new values.

**Figure 40-12. Method 1 (UPDM = 0)**



## *Method 2: Manual write of duty-cycle values and automatic trigger of the update*

In this mode, the update of the period value, the duty-cycle values, the dead-time values and the update period value must be done by writing in their respective update registers with the CPU (respectively PWM\_CPRDUPDx, PWM\_CDTYUPDx, PWM\_DTUPDx and PWM\_SCUPUPD).

To trigger the update of the period value and the dead-time values, the user must use the bit UPDULOCK of the “[PWM Sync Channels Update Control Register](#)” (PWM\_SCUC) which allows to update synchronously (at the same PWM period) the synchronous channels:

- If the bit UPDULOCK is set to 1, the update is done at the next PWM period of the synchronous channels.
- If the UPDULOCK bit is not set to 1, the update is locked and cannot be performed.

After writing the UPDULOCK bit to 1, it is held at this value until the update occurs, then it is read 0.

The update of the duty-cycle values and the update period is triggered automatically after an update period.

To configure the automatic update, the user must define a value for the Update Period by the UPR field in the “[PWM Sync Channels Update Period Register](#)” (PWM\_SCUP). The PWM controller waits UPR+1 period of synchronous channels before updating automatically the duty values and the update period value.

The status of the duty-cycle value write is reported in the “[PWM Interrupt Status Register 2](#)” (PWM\_ISR2) by the following flags:

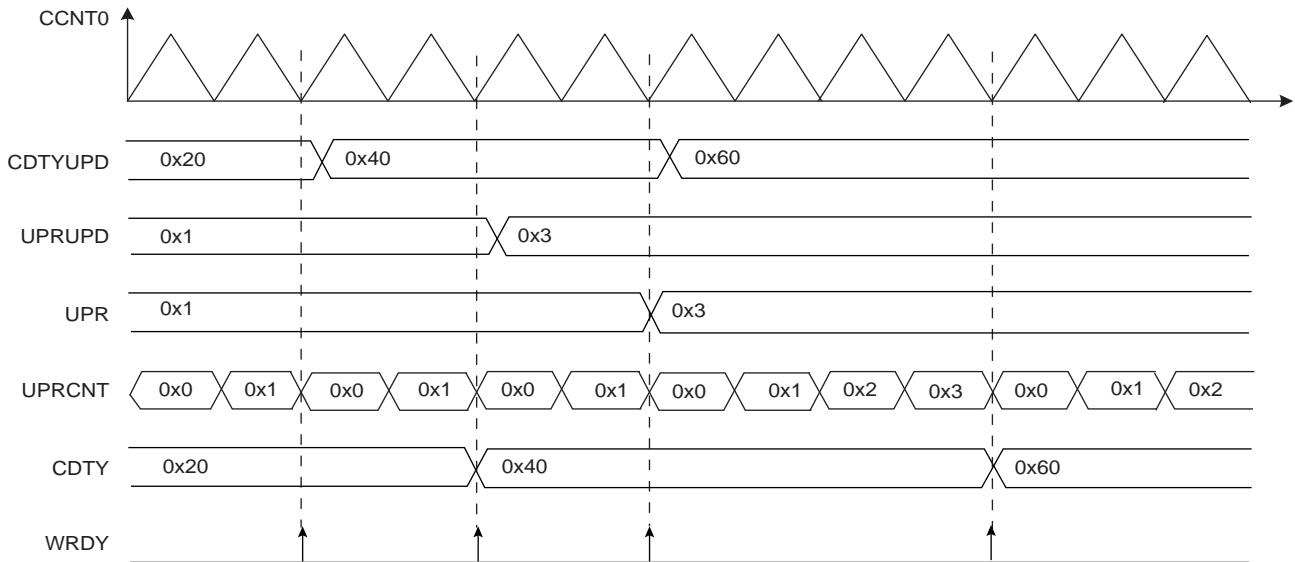
- WRDY: this flag is set to 1 when the PWM Controller is ready to receive new duty-cycle values and a new update period value. It is reset to 0 when the PWM\_ISR2 register is read.

Depending on the interrupt mask in the PWM\_IMR2 register, an interrupt can be generated by these flags.

Sequence for Method 2:

1. Select the manual write of duty-cycle values and the automatic update by setting the field UPDM to 1 in the PWM\_SCM register
2. Define the synchronous channels by the bits SYNCx in the PWM\_SCM register.
3. Define the update period by the field UPR in the PWM\_SCUP register.
4. Enable the synchronous channels by writing CHID0 in the PWM\_ENA register.
5. If an update of the period value and/or of the dead-time values is required, write registers that need to be updated (PWM\_CPRDUPDx, PWM\_DTUPDx), else go to [Step 8](#).
6. Set UPDULOCK to 1 in PWM\_SCUC.
7. The update of these registers will occur at the beginning of the next PWM period. At this moment the bit UPDULOCK is reset, go to [Step 5](#). for new values.
8. If an update of the duty-cycle values and/or the update period is required, check first that write of new update values is possible by polling the flag WRDY (or by waiting for the corresponding interrupt) in the PWM\_ISR2 register.
9. Write registers that need to be updated (PWM\_CDTYUPDx, PWM\_SCUPUPD).
10. The update of these registers will occur at the next PWM period of the synchronous channels when the Update Period is elapsed. Go to [Step 8](#). for new values.

**Figure 40-13. Method 2 (UPDM=1)**



### *Method 3: Automatic write of duty-cycle values and automatic trigger of the update*

In this mode, the update of the duty cycle values is made automatically by the Peripheral DMA Controller (PDC). The update of the period value, the dead-time values and the update period value must be done by writing in their respective update registers with the CPU (respectively PWM\_CPRDUPDx, PWM\_DTUPDx and PWM\_SCUPUPD).

To trigger the update of the period value and the dead-time values, the user must use the bit UPDULOCK which allows to update synchronously (at the same PWM period) the synchronous channels:

- If the bit UPDULOCK is set to 1, the update is done at the next PWM period of the synchronous channels.
- If the UPDULOCK bit is not set to 1, the update is locked and cannot be performed.

After writing the UPDULOCK bit to 1, it is held at this value until the update occurs, then it is read 0.

The update of the duty-cycle values and the update period value is triggered automatically after an update period.

To configure the automatic update, the user must define a value for the Update Period by the field UPR in the “[PWM Sync Channels Update Period Register](#)” (PWM\_SCUP). The PWM controller waits UPR+1 periods of synchronous channels before updating automatically the duty values and the update period value.

Using the PDC removes processor overhead by reducing its intervention during the transfer. This significantly reduces the number of clock cycles required for a data transfer, which improves microcontroller performance.

The PDC must write the duty-cycle values in the synchronous channels index order. For example if the channels 0, 1 and 3 are synchronous channels, the PDC must write the duty-cycle of the channel 0 first, then the duty-cycle of the channel 1, and finally the duty-cycle of the channel 3.

The status of the PDC transfer is reported in the “[PWM Interrupt Status Register 2](#)” (PWM\_ISR2) by the following flags:

- WRDY: this flag is set to 1 when the PWM Controller is ready to receive new duty-cycle values and a new update period value. It is reset to 0 when the PWM\_ISR2 register is read. The user can choose to synchronize the WRDY flag and the PDC transfer request with a comparison match (see [Section 40.6.3 “PWM Comparison Units”](#)), by the fields PTRM and PTRCS in the PWM\_SCM register.
- ENDTX: this flag is set to 1 when a PDC transfer is completed
- TXBUFE: this flag is set to 1 when the PDC buffer is empty (no pending PDC transfers)
- UNRE: this flag is set to 1 when the update period defined by the UPR field has elapsed while the whole data has not been written by the PDC. It is reset to 0 when the PWM\_ISR2 register is read.

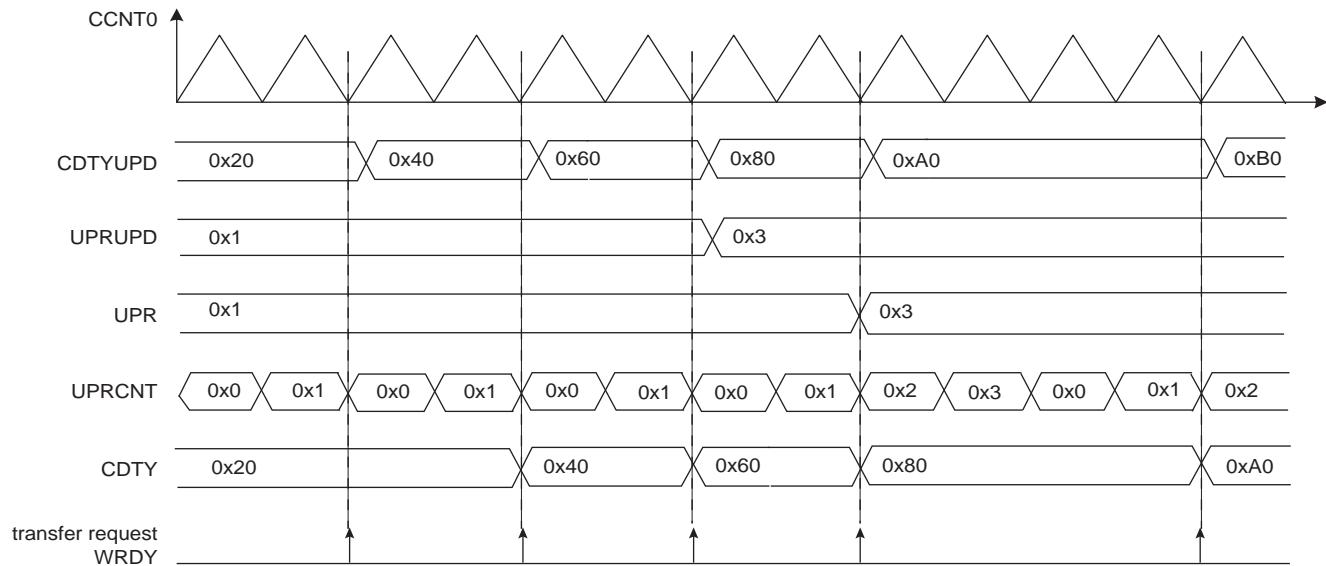
Depending on the interrupt mask in the PWM\_IMR2 register, an interrupt can be generated by these flags.

Sequence for Method 3:

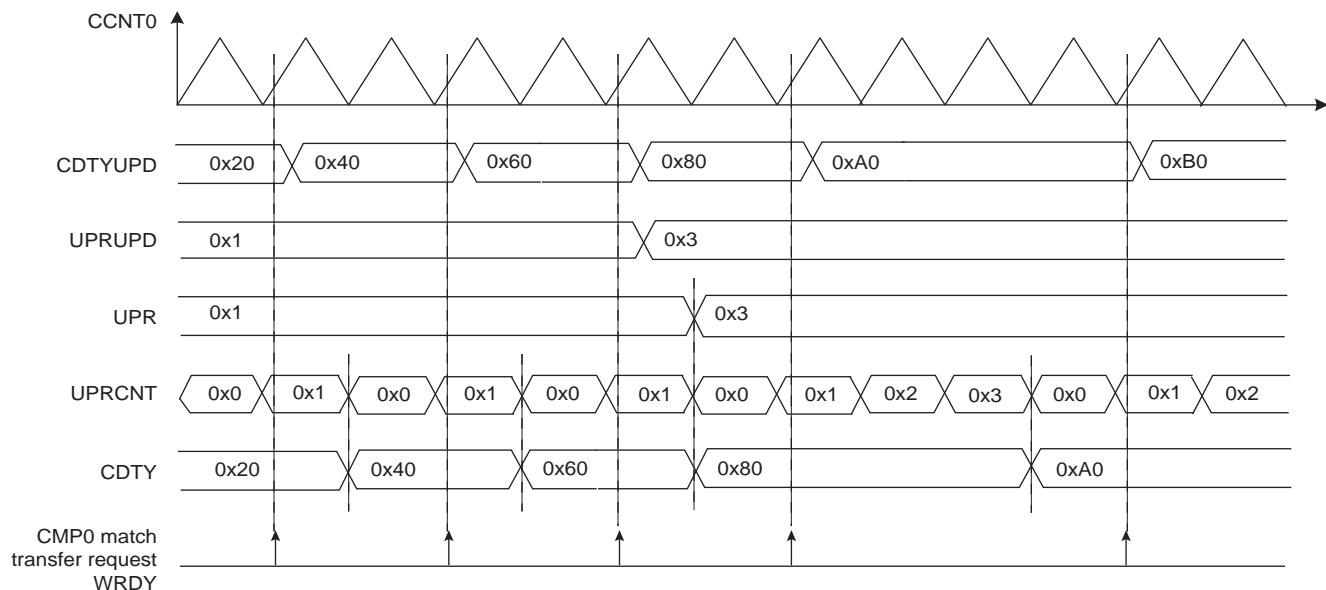
1. Select the automatic write of duty-cycle values and automatic update by setting the field UPDM to 2 in the PWM\_SCM register.
2. Define the synchronous channels by the bits SYNCx in the PWM\_SCM register.
3. Define the update period by the field UPR in the PWM\_SCUP register.
4. Define when the WRDY flag and the corresponding PDC transfer request must be set in the update period by the PTRM bit and the PTRCS field in the PWM\_SCM register (at the end of the update period or when a comparison matches).
5. Define the PDC transfer settings for the duty-cycle values and enable it in the PDC registers
6. Enable the synchronous channels by writing CHID0 in the PWM\_ENA register.
7. If an update of the period value and/or of the dead-time values is required, write registers that need to be updated (PWM\_CPRDUPDx, PWM\_DTUPDx), else go to [Step 10](#).
8. Set UPDULOCK to 1 in PWM\_SCUC.
9. The update of these registers will occur at the beginning of the next PWM period. At this moment the bit UPDULOCK is reset, go to [Step 7](#). for new values.
10. If an update of the update period value is required, check first that write of a new update value is possible by polling the flag WRDY (or by waiting for the corresponding interrupt) in the PWM\_ISR2 register, else go to [Step 13](#).
11. Write the register that needs to be updated (PWM\_SCUPUPD).

12. The update of this register will occur at the next PWM period of the synchronous channels when the Update Period is elapsed. Go to [Step 10.](#) for new values.
13. Check the end of the PDC transfer by the flag ENDTX. If the transfer has ended, define a new PDC transfer in the PDC registers for new duty-cycle values. Go to [Step 5.](#)

**Figure 40-14. Method 3 (UPDM=2 and PTRM=0)**



**Figure 40-15. Method 3 (UPDM=2 and PTRM=1 and PTRCS=0)**



#### 40.6.2.10 Update Time for Double-Buffering Registers

All channels integrate a double buffering system in order to prevent an unexpected output waveform while modifying the period, the spread spectrum value, the polarity, the duty-cycle, the additional edge value, the dead-times, the output override and the synchronous channels update period.

All these update registers are listed below:

- “[PWM Sync Channels Update Period Update Register](#)” on page 1047
- “[PWM Output Selection Set Update Register](#)” on page 1056
- “[PWM Output Selection Clear Update Register](#)” on page 1057
- “[PWM Spread Spectrum Update Register](#)” on page 1065
- “[PWM Channel Duty Cycle Update Register](#)” on page 1078
- “[PWM Channel Period Update Register](#)” on page 1080
- “[PWM Channel Dead Time Update Register](#)” on page 1083
- “[PWM Channel Mode Update Register](#)” on page 1084
- “[PWM Channel Additional Edge Update Register](#)” on page 1086

When one of these update registers are written to, the write is stored, but the values are updated only at the next PWM period border. In left-aligned mode (CALG=0) the update occurs when the channel counter reaches the period value CPRD. In center-aligned mode the update occurs when the channel counter value is decremented and reaches the 0 value.

In center-aligned mode, it is possible to trigger the update of the polarity, the duty-cycle, the additional edge value, at the next half period border. It concerns the following update registers:

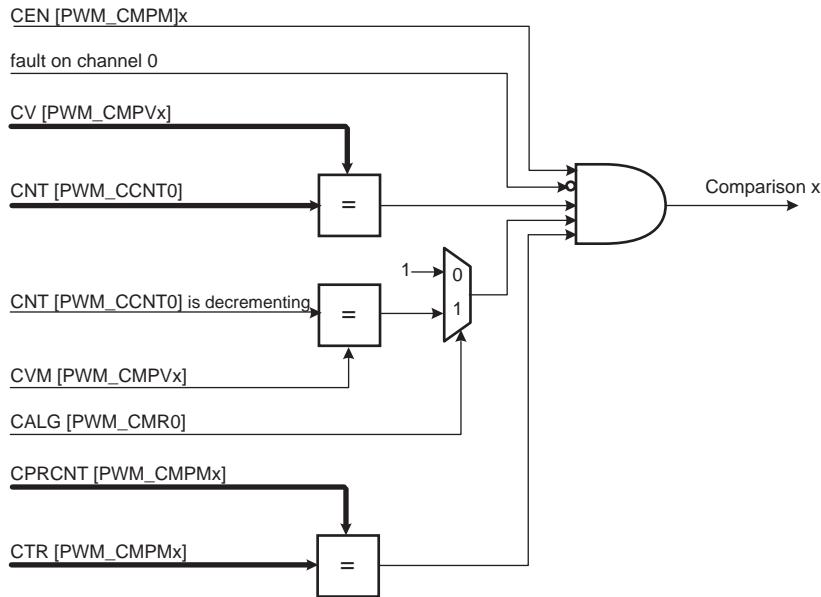
- “[PWM Channel Duty Cycle Update Register](#)” on page 1078
- “[PWM Channel Mode Update Register](#)” on page 1084
- “[PWM Channel Additional Edge Update Register](#)” on page 1086

That means the update occurs at the first half period following the write of the update register (either when the channel counter value is incrementing and reaches the period value CPRD, or when the channel counter value is decrementing and reaches the 0 value). To activate this mode, the user must write the bit UPDS at 1 in the “[PWM Channel Mode Register](#)” on page 1075.

### 40.6.3 PWM Comparison Units

The PWM provides 8 independent comparison units able to compare a programmed value with the current value of the channel 0 counter (which is the channel counter of all synchronous channels, [Section 40.6.2.9 “Synchronous Channels”](#)). These comparisons are intended to generate pulses on the event lines (used to synchronize ADC, see [Section 40.6.4 “PWM Event Lines”](#)), to generate software interrupts and to trigger PDC transfer requests for the synchronous channels (see “[Method 3: Automatic write of duty-cycle values and automatic trigger of the update](#)” on page 1023).

**Figure 40-16. Comparison Unit Block Diagram**



The comparison x matches when it is enabled by the bit CEN in the “[PWM Comparison x Mode Register](#)” (PWM\_CMPMx for the comparison x) and when the counter of the channel 0 reaches the comparison value defined by the field CV in “[PWM Comparison x Value Register](#)” (PWM\_CMPVx for the comparison x). If the counter of the channel 0 is center aligned (CALG = 1 in “[PWM Channel Mode Register](#)”), the bit CVM (in PWM\_CMPVx) defines if the comparison is made when the counter is counting up or counting down (in left alignment mode CALG=0, this bit is useless).

If a fault is active on the channel 0, the comparison is disabled and cannot match (see [Section 40.6.2.6 “Fault Protection”](#)).

The user can define the periodicity of the comparison x by the fields CTR and CPR (in PWM\_CMPVx). The comparison is performed periodically once every CPR+1 periods of the counter of the channel 0, when the value of the comparison period counter CPRCNT (in PWM\_CMPMx) reaches the value defined by CTR. CPR is the maximum value of the comparison period counter CPRCNT. If CPR=CTR=0, the comparison is performed at each period of the counter of the channel 0.

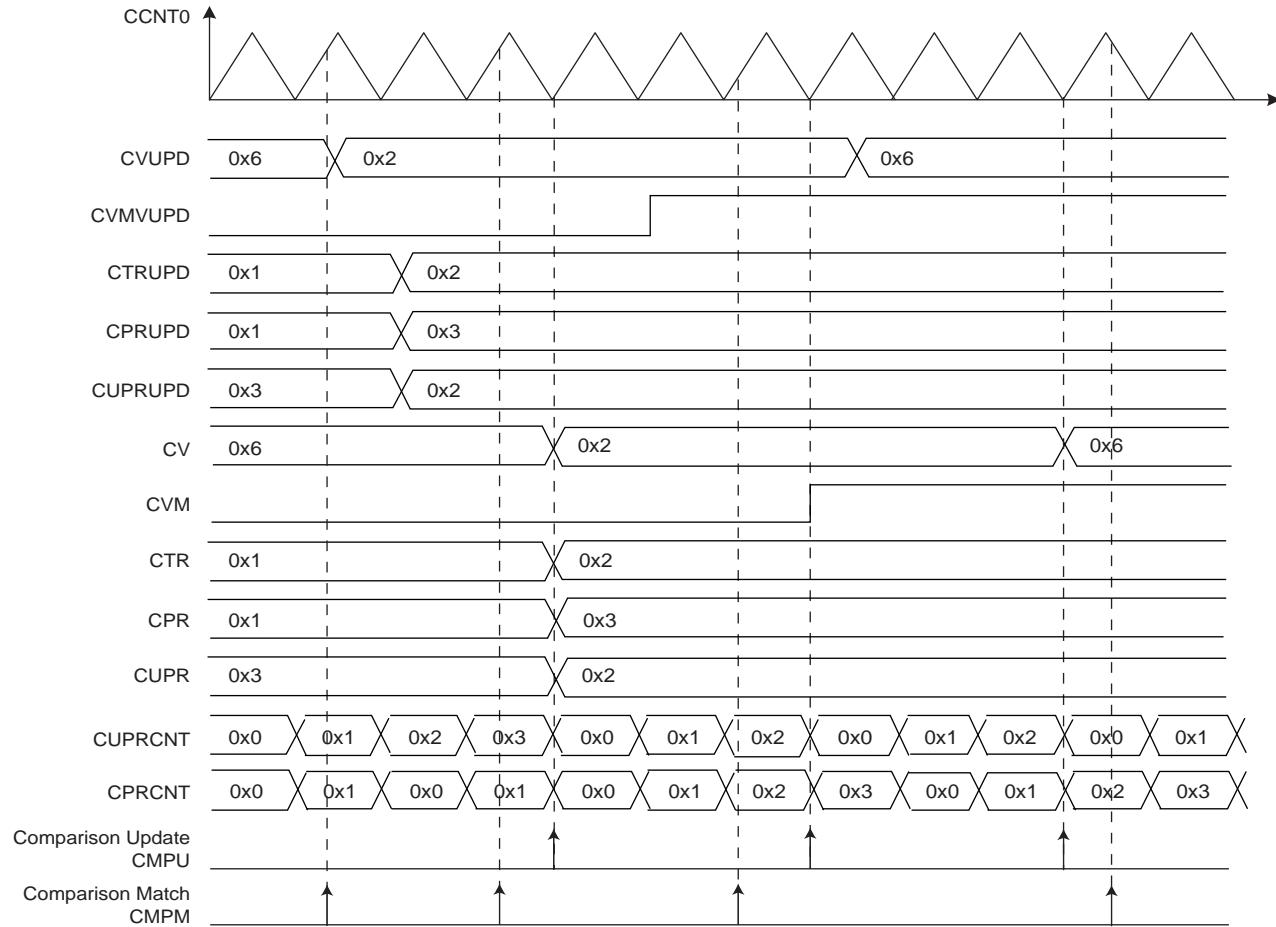
The comparison x configuration can be modified while the channel 0 is enabled by using the “[PWM Comparison x Mode Update Register](#)” (PWM\_CMPMUPDx registers for the comparison x). In the same way, the comparison x value can be modified while the channel 0 is enabled by using the “[PWM Comparison x Value Update Register](#)” (PWM\_CMPVUPDx registers for the comparison x).

The update of the comparison x configuration and the comparison x value is triggered periodically after the comparison x update period. It is defined by the field CUPR in the PWM\_CMPMx. The comparison unit has an update period counter independent from the period counter to trigger this update. When the value of the comparison update period counter CUPRCNT (in PWM\_CMPMx) reaches the value defined by CUPR, the update is triggered. The comparison x update period CUPR itself can be updated while the channel 0 is enabled by using the PWM\_CMPMUPDx register.

**CAUTION:** To be taken into account, the write of the register PWM\_CMPVUPDx must be followed by a write of the register PWM\_CMPMUPDx.

The comparison match and the comparison update can be source of an interrupt, but only if it is enabled and not masked. These interrupts can be enabled by the “[PWM Interrupt Enable Register 2](#)” and disabled by the “[PWM Interrupt Disable Register 2](#)”. The comparison match interrupt and the comparison update interrupt are reset by reading the “[PWM Interrupt Status Register 2](#)”.

**Figure 40-17. Comparison Waveform**

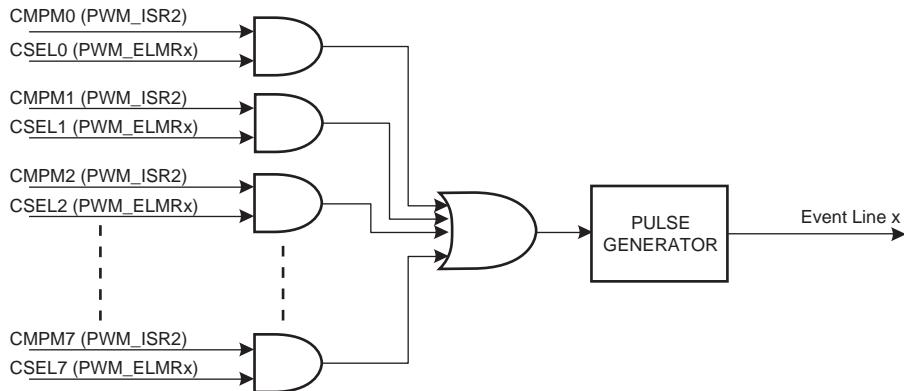


#### 40.6.4 PWM Event Lines

The PWM provides 2 independent event lines intended to trigger actions in other peripherals (in particular for ADC (Analog-to-Digital Converter)).

A pulse (one cycle of the master clock (MCK)) is generated on an event line, when at least one of the selected comparisons is matching. The comparisons can be selected or unselected independently by the CSEL bits in the “[PWM Event Line x Register](#)” (PWM\_ELMRx for the Event Line x).

**Figure 40-18. Event Line Block Diagram**



#### 40.6.5 PWM Controller Operations

##### 40.6.5.1 Initialization

Before enabling the channels, they must have been configured by the software application:

- Unlock User Interface by writing the WPCMD field in the PWM\_WPCR Register.
- Configuration of the clock generator (DIVA, PREA, DIVB, PREB in the PWM\_CLK register if required).
- Selection of the clock for each channel (CPRE field in the PWM\_CMRx register)
- Configuration of the waveform alignment for each channel (CALG field in the PWM\_CMRx register)
- Selection of the counter event selection (if CALG = 1) for each channel (CES field in the PWM\_CMRx register)
- Configuration of the output waveform polarity for each channel (CPOL in the PWM\_CMRx register)
- Configuration of the period for each channel (CPRD in the PWM\_CPRDx register). Writing in PWM\_CPRDx register is possible while the channel is disabled. After validation of the channel, the user must use PWM\_CPRDUPDx register to update PWM\_CPRDx as explained below.
- Configuration of the duty-cycle for each channel (CDTY in the PWM\_CDTYx register). Writing in PWM\_CDTYx register is possible while the channel is disabled. After validation of the channel, the user must use PWM\_CDTYUPDx register to update PWM\_CDTYx as explained below.
- Configuration of the dead-time generator for each channel (DTH and DTL in PWM\_DTx) if enabled (DTE bit in the PWM\_CMRx register). Writing in the PWM\_DTx register is possible while the channel is disabled. After validation of the channel, the user must use PWM\_DTUPDx register to update PWM\_DTx
- Selection of the synchronous channels (SYNCx in the PWM\_SCM register)
- Selection of the moment when the WRDY flag and the corresponding PDC transfer request are set (PTRM and PTRCS in the PWM\_SCM register)
- Configuration of the update mode (UPDM in the PWM\_SCM register)
- Configuration of the update period (UPR in the PWM\_SCUP register) if needed.
- Configuration of the comparisons (PWM\_CMPVx and PWM\_CMPMx).
- Configuration of the event lines (PWM\_ELMRx).
- Configuration of the fault inputs polarity (FPOL in PWM\_FMR)
- Configuration of the fault protection (FMOD and FFIL in PWM\_FMR, PWM\_FPV and PWM\_FPE1)

- Enable of the Interrupts (writing CHIDx and FCHIDx in PWM\_IER1 register, and writing WRDYE, ENDTXE, TXBUFE, UNRE, CMPMx and CMPUx in PWM\_IER2 register)
- Enable of the PWM channels (writing CHIDx in the PWM\_ENA register)

#### 40.6.5.2 Source Clock Selection Criteria

The large number of source clocks can make selection difficult. The relationship between the value in the “[PWM Channel Period Register](#)” (PWM\_CPRDx) and the “[PWM Channel Duty Cycle Register](#)” (PWM\_CDTYx) can help the user in choosing. The event number written in the Period Register gives the PWM accuracy. The Duty-Cycle quantum cannot be lower than  $1/CPRDx$  value. The higher the value of PWM\_CPRDx, the greater the PWM accuracy.

For example, if the user sets 15 (in decimal) in PWM\_CPRDx, the user is able to set a value from between 1 up to 14 in PWM\_CDTYx Register. The resulting duty-cycle quantum cannot be lower than 1/15 of the PWM period.

#### 40.6.5.3 Changing the Duty-Cycle, the Period and the Dead-Times

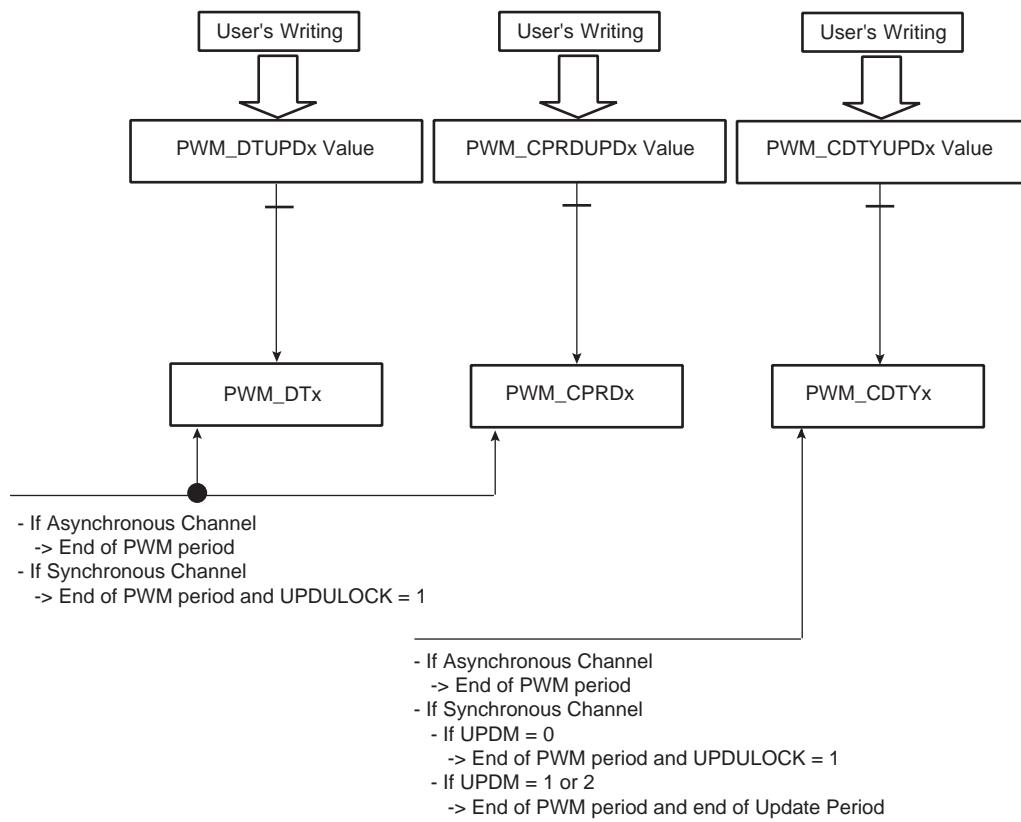
It is possible to modulate the output waveform duty-cycle, period and dead-times.

To prevent unexpected output waveform, the user must use the “[PWM Channel Duty Cycle Update Register](#)”, the “[PWM Channel Period Update Register](#)” and the “[PWM Channel Dead Time Update Register](#)” (PWM\_CDTYUPDx, PWM\_CPRDUPDx and PWM\_DTUPDx) to change waveform parameters while the channel is still enabled.

- If the channel is an asynchronous channel (SYNCx = 0 in “[PWM Sync Channels Mode Register](#)” (PWM\_SCM)), these registers hold the new period, duty-cycle and dead-times values until the end of the current PWM period and update the values for the next period.
- If the channel is a synchronous channel and update method 0 is selected (SYNCx = 1 and UPDM = 0 in PWM\_SCM register), these registers hold the new period, duty-cycle and dead-times values until the bit UPDULOCK is written at “1” (in “[PWM Sync Channels Update Control Register](#)” (PWM\_SCUC)) and the end of the current PWM period, then update the values for the next period.
- If the channel is a synchronous channel and update method 1 or 2 is selected (SYNCx=1 and UPDM=1 or 2 in PWM\_SCM register):
  - registers PWM\_CPRDUPDx and PWM\_DTUPDx hold the new period and dead-times values until the bit UPDULOCK is written at “1” (in PWM\_SCUC register) and the end of the current PWM period, then update the values for the next period.
  - register PWM\_CDTYUPDx holds the new duty-cycle value until the end of the update period of synchronous channels (when UPRCNT is equal to UPR in “[PWM Sync Channels Update Period Register](#)” (PWM\_SCUP)) and the end of the current PWM period, then updates the value for the next period.

Note: If the update registers PWM\_CDTYUPDx, PWM\_CPRDUPDx and PWM\_DTUPDx are written several times between two updates, only the last written value is taken into account.

**Figure 40-19. Synchronized Period, Duty-Cycle and Dead-Times Update**



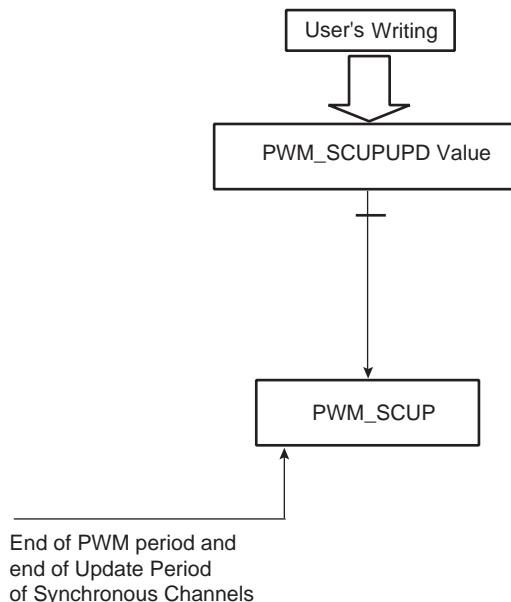
#### 40.6.5.4 Changing the Synchronous Channels Update Period

It is possible to change the update period of synchronous channels while they are enabled. (See “[Method 2: Manual write of duty-cycle values and automatic trigger of the update](#)” on page 1021 and “[Method 3: Automatic write of duty-cycle values and automatic trigger of the update](#)” on page 1023.)

To prevent an unexpected update of the synchronous channels registers, the user must use the “[PWM Sync Channels Update Period Update Register](#)” (PWM\_SCUPUPD) to change the update period of synchronous channels while they are still enabled. This register holds the new value until the end of the update period of synchronous channels (when UPRCNT is equal to UPR in “[PWM Sync Channels Update Period Register](#)” (PWM\_SCUP)) and the end of the current PWM period, then updates the value for the next period.

- Notes:
1. If the update register PWM\_SCUPUPD is written several times between two updates, only the last written value is taken into account.
  2. Changing the update period does make sense only if there is one or more synchronous channels and if the update method 1 or 2 is selected (UPDM = 1 or 2 in “[PWM Sync Channels Mode Register](#)”).

**Figure 40-20. Synchronized Update of Update Period Value of Synchronous Channels**



#### 40.6.5.5 Changing the Comparison Value and the Comparison Configuration

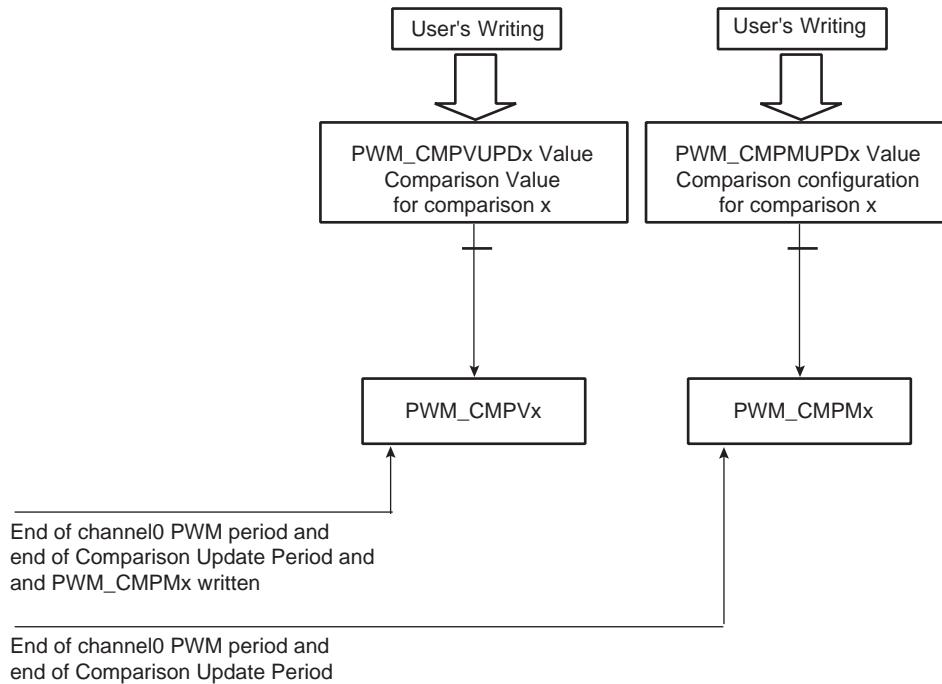
It is possible to change the comparison values and the comparison configurations while the channel 0 is enabled (see Section 40.6.3 “[PWM Comparison Units](#)”).

To prevent unexpected comparison match, the user must use the “[PWM Comparison x Value Update Register](#)” and the “[PWM Comparison x Mode Update Register](#)” (PWM\_CMPVUPDx and PWM\_CMPMUPDx) to change respectively the comparison values and the comparison configurations while the channel 0 is still enabled. These registers hold the new values until the end of the comparison update period (when CUPRCNT is equal to CUPR in “[PWM Comparison x Mode Register](#)” (PWM\_CMPMx) and the end of the current PWM period, then update the values for the next period.

**CAUTION:** To be taken into account, the write of the register PWM\_CMPVUPDx must be followed by a write of the register PWM\_CMPMUPDx.

Note: If the update registers PWM\_CMPVUPDx and PWM\_CMPMUPDx are written several times between two updates, only the last written value are taken into account.

**Figure 40-21. Synchronized Update of Comparison Values and Configurations**



#### 40.6.5.6 Interrupts

Depending on the interrupt mask in the PWM\_IMR1 and PWM\_IMR2 registers, an interrupt can be generated at the end of the corresponding channel period (CHIDx in the PWM\_ISR1 register), after a fault event (FCHIDx in the PWM\_ISR1 register), after a comparison match (CMPMx in the PWM\_ISR2 register), after a comparison update (CMPUx in the PWM\_ISR2 register) or according to the transfer mode of the synchronous channels (WRDY, ENDTX, TXBUFE and UNRE in the PWM\_ISR2 register).

If the interrupt is generated by the flags CHIDx or FCHIDx, the interrupt remains active until a read operation in the PWM\_ISR1 register occurs.

If the interrupt is generated by the flags WRDY or UNRE or CMPMx or CMPUx, the interrupt remains active until a read operation in the PWM\_ISR2 register occurs.

A channel interrupt is enabled by setting the corresponding bit in the PWM\_IER1 and PWM\_IER2 registers. A channel interrupt is disabled by setting the corresponding bit in the PWM\_IDR1 and PWM\_IDR2 registers.

#### 40.6.5.7 Write Protect Registers

To prevent any single software error that may corrupt PWM behavior, the registers listed below can be write-protected by writing the field WPCMD in the “[PWM Write Protect Control Register](#)” on page 1068 (PWM\_WPCR). They are divided into 6 groups:

- Register group 0:
  - “[PWM Clock Register](#)” on page 1037
- Register group 1:
  - “[PWM Disable Register](#)” on page 1039
- Register group 2:
  - “[PWM Sync Channels Mode Register](#)” on page 1044
  - “[PWM Channel Mode Register](#)” on page 1075
  - “[PWM Stepper Motor Mode Register](#)” on page 1066
  - “[PWM Channel Mode Update Register](#)” on page 1084
- Register group 3:
  - “[PWM Spread Spectrum Register](#)” on page 1064
  - “[PWM Spread Spectrum Update Register](#)” on page 1065
  - “[PWM Channel Period Register](#)” on page 1079
  - “[PWM Channel Period Update Register](#)” on page 1080
- Register group 4:
  - “[PWM Channel Dead Time Register](#)” on page 1082
  - “[PWM Channel Dead Time Update Register](#)” on page 1083
- Register group 5:
  - “[PWM Fault Mode Register](#)” on page 1058
  - “[PWM Fault Protection Value Register 1](#)” on page 1061

There are two types of Write Protect:

- Write Protect SW, which can be enabled or disabled.
- Write Protect HW, which can just be enabled, only a hardware reset of the PWM controller can disable it.

Both types of Write Protect can be applied independently to a particular register group by means of the WPCMD and WPRG fields in PWM\_WPCR register. If at least one Write Protect is active, the register group is write-protected. The field WPCMD allows to perform the following actions depending on its value:

- 0 = Disabling the Write Protect SW of the register groups of which the bit WPRG is at 1.
- 1 = Enabling the Write Protect SW of the register groups of which the bit WPRG is at 1.
- 2 = Enabling the Write Protect HW of the register groups of which the bit WPRG is at 1.

At any time, the user can determine which Write Protect is active in which register group by the fields WPSWS and WPHWS in the “[PWM Write Protect Status Register](#)” on page 1070 (PWM\_WPSR).

If a write access in a write-protected register is detected, then the WPVS flag in the PWM\_WPSR register is set and the field WPVSR indicates in which register the write access has been attempted, through its address offset without the two LSBs.

The WPVS and PWM\_WPSR fields are automatically reset after reading the PWM\_WPSR register.

## 40.7 Pulse Width Modulation Controller (PWM) Controller User Interface

Table 40-7. Register Mapping

Offset	Register	Name	Access	Reset
0x00	PWM Clock Register	PWM_CLK	Read-write	0x0
0x04	PWM Enable Register	PWM_ENA	Write-only	-
0x08	PWM Disable Register	PWM_DIS	Write-only	-
0x0C	PWM Status Register	PWM_SR	Read-only	0x0
0x10	PWM Interrupt Enable Register 1	PWM_IER1	Write-only	-
0x14	PWM Interrupt Disable Register 1	PWM_IDR1	Write-only	-
0x18	PWM Interrupt Mask Register 1	PWM_IMR1	Read-only	0x0
0x1C	PWM Interrupt Status Register 1	PWM_ISR1	Read-only	0x0
0x20	PWM Sync Channels Mode Register	PWM_SCM	Read-write	0x0
0x24	Reserved	-	-	-
0x28	PWM Sync Channels Update Control Register	PWM_SCUC	Read-write	0x0
0x2C	PWM Sync Channels Update Period Register	PWM_SCUP	Read-write	0x0
0x30	PWM Sync Channels Update Period Update Register	PWM_SCUPUPD	Write-only	0x0
0x34	PWM Interrupt Enable Register 2	PWM_IER2	Write-only	-
0x38	PWM Interrupt Disable Register 2	PWM_IDR2	Write-only	-
0x3C	PWM Interrupt Mask Register 2	PWM_IMR2	Read-only	0x0
0x40	PWM Interrupt Status Register 2	PWM_ISR2	Read-only	0x0
0x44	PWM Output Override Value Register	PWM_OOV	Read-write	0x0
0x48	PWM Output Selection Register	PWM_OS	Read-write	0x0
0x4C	PWM Output Selection Set Register	PWM_OSS	Write-only	-
0x50	PWM Output Selection Clear Register	PWM_OSC	Write-only	-
0x54	PWM Output Selection Set Update Register	PWM_OSSUPD	Write-only	-
0x58	PWM Output Selection Clear Update Register	PWM_OSCUPD	Write-only	-
0x5C	PWM Fault Mode Register	PWM_FMR	Read-write	0x0
0x60	PWM Fault Status Register	PWM_FSR	Read-only	0x0
0x64	PWM Fault Clear Register	PWM_FCR	Write-only	-
0x68	PWM Fault Protection Value Register 1	PWM_FPV1	Read-write	0x0
0x6C	PWM Fault Protection Enable Register	PWM_FPE	Read-write	0x0
0x70-0x78	Reserved	-	-	-
0x7C	PWM Event Line 0 Mode Register	PWM_ELMR0	Read-write	0x0
0x80	PWM Event Line 1 Mode Register	PWM_ELMR1	Read-write	0x0
0x84-0x9C	Reserved	-	-	-
0xA0	PWM Spread Spectrum Register	PWM_SSPr	Read-write	0x0
0xA4	PWM Spread Spectrum Update Register	PWM_SSPrUPD	Write-only	-
0xA8-AC	Reserved	-	-	-

**Table 40-7. Register Mapping (Continued)**

<b>Offset</b>	<b>Register</b>	<b>Name</b>	<b>Access</b>	<b>Reset</b>
0xB0	PWM Stepper Motor Mode Register	PWM_SMMR	Read-write	0x0
0xB4-0xBC	Reserved	—	—	—
0xC0	PWM Fault Protection Value 2 Register	PWM_FPV2	Read-write	0x0
0xC4-E0	Reserved	—	—	—
0xE4	PWM Write Protect Control Register	PWM_WPCR	Write-only	—
0xE8	PWM Write Protect Status Register	PWM_WPSR	Read-only	0x0
0xEC - 0xFC	Reserved	—	—	—
0x100 - 0x128	Reserved for PDC registers	—	—	—
0x12C	Reserved	—	—	—
0x130	PWM Comparison 0 Value Register	PWM_CMPV0	Read-write	0x0
0x134	PWM Comparison 0 Value Update Register	PWM_CMPVUPD0	Write-only	—
0x138	PWM Comparison 0 Mode Register	PWM_CMPM0	Read-write	0x0
0x13C	PWM Comparison 0 Mode Update Register	PWM_CMPMUPD0	Write-only	—
0x140	PWM Comparison 1 Value Register	PWM_CMPV1	Read-write	0x0
0x144	PWM Comparison 1 Value Update Register	PWM_CMPVUPD1	Write-only	—
0x148	PWM Comparison 1 Mode Register	PWM_CMPM1	Read-write	0x0
0x14C	PWM Comparison 1 Mode Update Register	PWM_CMPMUPD1	Write-only	—
0x150	PWM Comparison 2 Value Register	PWM_CMPV2	Read-write	0x0
0x154	PWM Comparison 2 Value Update Register	PWM_CMPVUPD2	Write-only	—
0x158	PWM Comparison 2 Mode Register	PWM_CMPM2	Read-write	0x0
0x15C	PWM Comparison 2 Mode Update Register	PWM_CMPMUPD2	Write-only	—
0x160	PWM Comparison 3 Value Register	PWM_CMPV3	Read-write	0x0
0x164	PWM Comparison 3 Value Update Register	PWM_CMPVUPD3	Write-only	—
0x168	PWM Comparison 3 Mode Register	PWM_CMPM3	Read-write	0x0
0x16C	PWM Comparison 3 Mode Update Register	PWM_CMPMUPD3	Write-only	—
0x170	PWM Comparison 4 Value Register	PWM_CMPV4	Read-write	0x0
0x174	PWM Comparison 4 Value Update Register	PWM_CMPVUPD4	Write-only	—
0x178	PWM Comparison 4 Mode Register	PWM_CMPM4	Read-write	0x0
0x17C	PWM Comparison 4 Mode Update Register	PWM_CMPMUPD4	Write-only	—
0x180	PWM Comparison 5 Value Register	PWM_CMPV5	Read-write	0x0
0x184	PWM Comparison 5 Value Update Register	PWM_CMPVUPD5	Write-only	—
0x188	PWM Comparison 5 Mode Register	PWM_CMPM5	Read-write	0x0
0x18C	PWM Comparison 5 Mode Update Register	PWM_CMPMUPD5	Write-only	—
0x190	PWM Comparison 6 Value Register	PWM_CMPV6	Read-write	0x0
0x194	PWM Comparison 6 Value Update Register	PWM_CMPVUPD6	Write-only	—
0x198	PWM Comparison 6 Mode Register	PWM_CMPM6	Read-write	0x0
0x19C	PWM Comparison 6 Mode Update Register	PWM_CMPMUPD6	Write-only	—

**Table 40-7. Register Mapping (Continued)**

<b>Offset</b>	<b>Register</b>	<b>Name</b>	<b>Access</b>	<b>Reset</b>
0x1A0	PWM Comparison 7 Value Register	PWM_CMPV7	Read-write	0x0
0x1A4	PWM Comparison 7 Value Update Register	PWM_CMPVUPD7	Write-only	–
0x1A8	PWM Comparison 7 Mode Register	PWM_CMPM7	Read-write	0x0
0x1AC	PWM Comparison 7 Mode Update Register	PWM_CMMPMUPD7	Write-only	–
0x1B0 - 0x1FC	Reserved	–	–	–
0x200 + ch_num * 0x20 + 0x00	PWM Channel Mode Register <sup>(1)</sup>	PWM_CMR	Read-write	0x0
0x200 + ch_num * 0x20 + 0x04	PWM Channel Duty Cycle Register <sup>(1)</sup>	PWM_CDTY	Read-write	0x0
0x200 + ch_num * 0x20 + 0x08	PWM Channel Duty Cycle Update Register <sup>(1)</sup>	PWM_CDTYUPD	Write-only	–
0x200 + ch_num * 0x20 + 0x0C	PWM Channel Period Register <sup>(1)</sup>	PWM_CPRD	Read-write	0x0
0x200 + ch_num * 0x20 + 0x10	PWM Channel Period Update Register <sup>(1)</sup>	PWM_CPRDUPD	Write-only	–
0x200 + ch_num * 0x20 + 0x14	PWM Channel Counter Register <sup>(1)</sup>	PWM_CCNT	Read-only	0x0
0x200 + ch_num * 0x20 + 0x18	PWM Channel Dead Time Register <sup>(1)</sup>	PWM_DT	Read-write	0x0
0x200 + ch_num * 0x20 + 0x1C	PWM Channel Dead Time Update Register <sup>(1)</sup>	PWM_DTUPD	Write-only	–
0x400 + ch_num * 0x20 + 0x00	PWM Channel Mode Update Register <sup>(1)</sup>	PWM_CMUPD	Write-only	–
0x400 + ch_num * 0x20 + 0x04	PWM Channel Additional Edge Register <sup>(1)</sup>	PWM_CAE	Read-write	0x0
0x400 + ch_num * 0x20 + 0x08	PWM Channel Additional Edge Update Register <sup>(1)</sup>	PWM_CAEUPD	Write-only	–

Note: 1. Some registers are indexed with “ch\_num” index ranging from 0 to 3.

#### 40.7.1 PWM Clock Register

**Name:** PWM\_CLK

**Address:** 0x40000000

**Access:** Read-write

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

This register can only be written if the bits WPSWS0 and WPHWS0 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

- DIVA, DIVB: CLKA, CLKb Divide Factor

DIVA, DIVB	CLKA, CLKb
0	CLKA, CLKb clock is turned off
1	CLKA, CLKb clock is clock selected by PREA, PREB
2-255	CLKA, CLKb clock is clock selected by PREA, PREB divided by DIVA, DIVB factor.

- PREA, PREB: CLKA, CLKb Source Clock Selection

PREA, PREB				Divider Input Clock
0	0	0	0	MCK
0	0	0	1	MCK/2
0	0	1	0	MCK/4
0	0	1	1	MCK/8
0	1	0	0	MCK/16
0	1	0	1	MCK/32
0	1	1	0	MCK/64
0	1	1	1	MCK/128
1	0	0	0	MCK/256
1	0	0	1	MCK/512
1	0	1	0	MCK/1024
Other				Reserved

#### 40.7.2 PWM Enable Register

**Name:** PWM\_ENA

**Address:** 0x40000004

**Access:** Write-only

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
-	-	-	-	CHID3	CHID2	CHID1	CHID0

- **CHIDx:** Channel ID

0 = No effect.

1 = Enable PWM output for channel x.

#### 40.7.3 PWM Disable Register

**Name:** PWM\_DIS

**Address:** 0x40000008

**Access:** Write-only

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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

This register can only be written if the bits WPSWS1 and WPHWS1 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

- **CHID<sub>x</sub>:** Channel ID

0 = No effect.

1 = Disable PWM output for channel x.

#### 40.7.4 PWM Status Register

**Name:** PWM\_SR

**Address:** 0x4000000C

**Access:** Read-only

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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHID<sub>x</sub>:** Channel ID

0 = PWM output for channel x is disabled.

1 = PWM output for channel x is enabled.

#### 40.7.5 PWM Interrupt Enable Register 1

**Name:** PWM\_IER1

**Address:** 0x40000010

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHID<sub>x</sub>:** Counter Event on Channel x Interrupt Enable
- **FCHID<sub>x</sub>:** Fault Protection Trigger on Channel x Interrupt Enable

#### 40.7.6 PWM Interrupt Disable Register 1

**Name:** PWM\_IDR1

**Address:** 0x40000014

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHID<sub>x</sub>:** Counter Event on Channel x Interrupt Disable
- **FCHID<sub>x</sub>:** Fault Protection Trigger on Channel x Interrupt Disable

#### 40.7.7 PWM Interrupt Mask Register 1

**Name:** PWM\_IMR1

**Address:** 0x40000018

**Access:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	CHID3	CHID2	CHID1	CHID0

- **CHIDx:** Counter Event on Channel x Interrupt Mask
- **FCHIDx:** Fault Protection Trigger on Channel x Interrupt Mask

#### 40.7.8 PWM Interrupt Status Register 1

**Name:** PWM\_ISR1

**Address:** 0x4000001C

**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHID<sub>x</sub>: Counter Event on Channel x**

0 = No new counter event has occurred since the last read of the PWM\_ISR1 register.

1 = At least one counter event has occurred since the last read of the PWM\_ISR1 register.

- **FCHID<sub>x</sub>: Fault Protection Trigger on Channel x**

0 = No new trigger of the fault protection since the last read of the PWM\_ISR1 register.

1 = At least one trigger of the fault protection since the last read of the PWM\_ISR1 register.

Note: Reading PWM\_ISR1 automatically clears CHID<sub>x</sub> and FCHID<sub>x</sub> flags.

#### 40.7.9 PWM Sync Channels Mode Register

**Name:** PWM\_SCM

**Address:** 0x40000020

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
PTRCS			PTRM	—	—	UPDM	
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	—
7	6	5	4	3	2	1	0
—	—	—	—	SYNC3	SYNC2	SYNC1	SYNC0

This register can only be written if the bits WPSWS2 and WPHWS2 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

- **SYNCx: Synchronous Channel x**

0 = Channel x is not a synchronous channel.

1 = Channel x is a synchronous channel.

- **UPDM: Synchronous Channels Update Mode**

Value	Name	Description
0	MODE0	Manual write of double buffer registers and manual update of synchronous channels <sup>(1)</sup>
1	MODE1	Manual write of double buffer registers and automatic update of synchronous channels <sup>(2)</sup>
2	MODE2	Automatic write of duty-cycle update registers by the PDC and automatic update of synchronous channels <sup>(2)</sup>
3	—	Reserved

Notes: 1. The update occurs at the beginning of the next PWM period, when the UPDULOCK bit in “[PWM Sync Channels Update Control Register](#)” is set.

2. The update occurs when the Update Period is elapsed.

- **PTRM: PDC Transfer Request Mode**

UPDM	PTRM	WRDY Flag and PDC Transfer Request
0	x	The WRDY flag in “ <a href="#">PWM Interrupt Status Register 2</a> ” on page <a href="#">1051</a> and the PDC transfer request are never set to 1.
1	x	The WRDY flag in “ <a href="#">PWM Interrupt Status Register 2</a> ” on page <a href="#">1051</a> is set to 1 as soon as the update period is elapsed, the PDC transfer request is never set to 1.
2	0	The WRDY flag in “ <a href="#">PWM Interrupt Status Register 2</a> ” on page <a href="#">1051</a> and the PDC transfer request are set to 1 as soon as the update period is elapsed.
	1	The WRDY flag in “ <a href="#">PWM Interrupt Status Register 2</a> ” on page <a href="#">1051</a> and the PDC transfer request are set to 1 as soon as the selected comparison matches.

- **PTRCS: PDC Transfer Request Comparison Selection**

Selection of the comparison used to set the flag WRDY and the corresponding PDC transfer request.

#### 40.7.10 PWM Sync Channels Update Control Register

**Name:** PWM\_SCUC

**Address:** 0x40000028

**Access:** Read-write

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	
–	–	–	–	–	–	–	–	UPDULOCK

- **UPDULOCK: Synchronous Channels Update Unlock**

0 = No effect

1 = If the UPDM field is set to “0” in [“PWM Sync Channels Mode Register” on page 1044](#), writing the UPDULOCK bit to “1” triggers the update of the period value, the duty-cycle and the dead-time values of synchronous channels at the beginning of the next PWM period. If the field UPDM is set to “1” or “2”, writing the UPDULOCK bit to “1” triggers only the update of the period value and of the dead-time values of synchronous channels.

This bit is automatically reset when the update is done.

#### 40.7.11 PWM Sync Channels Update Period Register

**Name:** PWM\_SCUP

**Address:** 0x4000002C

**Access:** Read-write

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
UPRCNT				UPR			

- **UPR: Update Period**

Defines the time between each update of the synchronous channels if automatic trigger of the update is activated (UPDM = 1 or UPDM = 2 in “[PWM Sync Channels Mode Register](#)” on page 1044). This time is equal to UPR+1 periods of the synchronous channels.

- **UPRCNT: Update Period Counter**

Reports the value of the Update Period Counter.

#### 40.7.12 PWM Sync Channels Update Period Update Register

**Name:** PWM\_SCUPUPD

**Address:** 0x40000030

**Access:** Write-only

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
–	–	–	–	–	–	–	UPRUPD

This register acts as a double buffer for the UPR value. This prevents an unexpected automatic trigger of the update of synchronous channels.

- **UPRUPD: Update Period Update**

Defines the wanted time between each update of the synchronous channels if automatic trigger of the update is activated (UPDM = 1 or UPDM = 2 in “[PWM Sync Channels Mode Register](#)” on page 1044). This time is equal to UPR+1 periods of the synchronous channels.

#### 40.7.13 PWM Interrupt Enable Register 2

**Name:** PWM\_IER2

**Address:** 0x40000034

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
CMPU7	CMPU6	CMPU5	CMPU4	CMPU3	CMPU2	CMPU1	CMPU0
15	14	13	12	11	10	9	8
CMPM7	CMPM6	CMPM5	CMPM4	CMPM3	CMPM2	CMPM1	CMPM0
7	6	5	4	3	2	1	0
–	–	–	–	UNRE	TXBUFE	ENDTX	WRDY

- **WRDY:** Write Ready for Synchronous Channels Update Interrupt Enable
- **ENDTX:** PDC End of TX Buffer Interrupt Enable
- **TXBUFE:** PDC TX Buffer Empty Interrupt Enable
- **UNRE:** Synchronous Channels Update Underrun Error Interrupt Enable
- **CMPMx:** Comparison x Match Interrupt Enable
- **CMPUx:** Comparison x Update Interrupt Enable

#### 40.7.14 PWM Interrupt Disable Register 2

**Name:** PWM\_IDR2

**Address:** 0x40000038

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
CMPU7	CMPU6	CMPU5	CMPU4	CMPU3	CMPU2	CMPU1	CMPU0
15	14	13	12	11	10	9	8
CMPM7	CMPM6	CMPM5	CMPM4	CMPM3	CMPM2	CMPM1	CMPM0
7	6	5	4	3	2	1	0
–	–	–	–	UNRE	TXBUFE	ENDTX	WRDY

- **WRDY:** Write Ready for Synchronous Channels Update Interrupt Disable
- **ENDTX:** PDC End of TX Buffer Interrupt Disable
- **TXBUFE:** PDC TX Buffer Empty Interrupt Disable
- **UNRE:** Synchronous Channels Update Underrun Error Interrupt Disable
- **CMPMx:** Comparison x Match Interrupt Disable
- **CMPUx:** Comparison x Update Interrupt Disable

#### 40.7.15 PWM Interrupt Mask Register 2

**Name:** PWM\_IMR2

**Address:** 0x4000003C

**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
CMPU7	CMPU6	CMPU5	CMPU4	CMPU3	CMPU2	CMPU1	CMPU0
15	14	13	12	11	10	9	8
CMPM7	CMPM6	CMPM5	CMPM4	CMPM3	CMPM2	CMPM1	CMPM0
7	6	5	4	3	2	1	0
–	–	–	–	UNRE	TXBUFE	ENDTX	WRDY

- **WRDY:** Write Ready for Synchronous Channels Update Interrupt Mask
- **ENDTX:** PDC End of TX Buffer Interrupt Mask
- **TXBUFE:** PDC TX Buffer Empty Interrupt Mask
- **UNRE:** Synchronous Channels Update Underrun Error Interrupt Mask
- **CMPMx:** Comparison x Match Interrupt Mask
- **CMPUx:** Comparison x Update Interrupt Mask

#### 40.7.16 PWM Interrupt Status Register 2

**Name:** PWM\_ISR2

**Address:** 0x40000040

**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
CMPU7	CMPU6	CMPU5	CMPU4	CMPU3	CMPU2	CMPU1	CMPU0
15	14	13	12	11	10	9	8
CMPM7	CMPM6	CMPM5	CMPM4	CMPM3	CMPM2	CMPM1	CMPM0
7	6	5	4	3	2	1	0
–	–	–	–	UNRE	TXBUFE	ENDTX	WRDY

- **WRDY: Write Ready for Synchronous Channels Update**

0 = New duty-cycle and dead-time values for the synchronous channels cannot be written.

1 = New duty-cycle and dead-time values for the synchronous channels can be written.

- **ENDTX: PDC End of TX Buffer**

0 = The Transmit Counter register has not reached 0 since the last write of the PDC.

1 = The Transmit Counter register has reached 0 since the last write of the PDC.

- **TXBUFE: PDC TX Buffer Empty**

0 = PWM\_TCR or PWM\_TCNR has a value other than 0.

1 = Both PWM\_TCR and PWM\_TCNR have a value other than 0.

- **UNRE: Synchronous Channels Update Underrun Error**

0 = No Synchronous Channels Update Underrun has occurred since the last read of the PWM\_ISR2 register.

1 = At least one Synchronous Channels Update Underrun has occurred since the last read of the PWM\_ISR2 register.

- **CMPMx: Comparison x Match**

0 = The comparison x has not matched since the last read of the PWM\_ISR2 register.

1 = The comparison x has matched at least one time since the last read of the PWM\_ISR2 register.

- **CMPUx: Comparison x Update**

0 = The comparison x has not been updated since the last read of the PWM\_ISR2 register.

1 = The comparison x has been updated at least one time since the last read of the PWM\_ISR2 register.

Note: Reading PWM\_ISR2 automatically clears flags WRDY, UNRE and CMPSx.

#### 40.7.17 PWM Output Override Value Register

**Name:** PWM\_OOV

**Address:** 0x40000044

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OOVL3	OOVL2	OOVL1	OOVL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OOVH3	OOVH2	OOVH1	OOVH0

- **OOVHx: Output Override Value for PWMH output of the channel x**

0 = Override value is 0 for PWMH output of channel x.

1 = Override value is 1 for PWMH output of channel x.

- **OOVLx: Output Override Value for PWML output of the channel x**

0 = Override value is 0 for PWML output of channel x.

1 = Override value is 1 for PWML output of channel x.

#### 40.7.18 PWM Output Selection Register

**Name:** PWM\_OS

**Address:** 0x40000048

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSL3	OSL2	OSL1	OSL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSH3	OSH2	OSH1	OSH0

- **OSHx: Output Selection for PWMH output of the channel x**

0 = Dead-time generator output DTOHx selected as PWMH output of channel x.

1 = Output override value OOVHx selected as PWMH output of channel x.

- **OSLx: Output Selection for PWML output of the channel x**

0 = Dead-time generator output DTOLx selected as PWML output of channel x.

1 = Output override value OOVLx selected as PWML output of channel x.

#### 40.7.19 PWM Output Selection Set Register

**Name:** PWM\_OSS

**Address:** 0x4000004C

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSSL3	OSSL2	OSSL1	OSSL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSSH3	OSSH2	OSSH1	OSSH0

- **OSSHx: Output Selection Set for PWMH output of the channel x**

0 = No effect.

1 = Output override value OOVHx selected as PWMH output of channel x.

- **OSSLx: Output Selection Set for PWML output of the channel x**

0 = No effect.

1 = Output override value OOVLx selected as PWML output of channel x.

#### 40.7.20 PWM Output Selection Clear Register

**Name:** PWM\_OSC

**Address:** 0x40000050

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSCL3	OSCL2	OSCL1	OSCL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSCH3	OSCH2	OSCH1	OSCH0

- **OSCHx: Output Selection Clear for PWMH output of the channel x**

0 = No effect.

1 = Dead-time generator output DTOHx selected as PWMH output of channel x.

- **OSCLx: Output Selection Clear for PWML output of the channel x**

0 = No effect.

1 = Dead-time generator output DTOLx selected as PWML output of channel x.

#### 40.7.21 PWM Output Selection Set Update Register

**Name:** PWM\_OSSUPD

**Address:** 0x40000054

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSSUPL3	OSSUPL2	OSSUPL1	OSSUPL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSSUPH3	OSSUPH2	OSSUPH1	OSSUPH0

- **OSSUPHx: Output Selection Set for PWMH output of the channel x**

0 = No effect.

1 = Output override value OOVHx selected as PWMH output of channel x at the beginning of the next channel x PWM period.

- **OSSUPLx: Output Selection Set for PWML output of the channel x**

0 = No effect.

1 = Output override value OOVLx selected as PWML output of channel x at the beginning of the next channel x PWM period.

#### 40.7.22 PWM Output Selection Clear Update Register

**Name:** PWM\_OSCUPD

**Address:** 0x40000058

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSCUPL3	OSCUPL2	OSCUPL1	OSCUPL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSCUPH3	OSCUPH2	OSCUPH1	OSCUPH0

- **OSCUPHx: Output Selection Clear for PWMH output of the channel x**

0 = No effect.

1 = Dead-time generator output DTOHx selected as PWMH output of channel x at the beginning of the next channel x PWM period.

- **OSCUPLx: Output Selection Clear for PWML output of the channel x**

0 = No effect.

1 = Dead-time generator output DTOLx selected as PWML output of channel x at the beginning of the next channel x PWM period.

#### 40.7.23 PWM Fault Mode Register

**Name:** PWM\_FMR

**Address:** 0x4000005C

**Access:** Read-write

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

This register can only be written if the bits WPSWS5 and WPHWS5 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

- **FPOL: Fault Polarity (fault input bit varies from 0 to 7)**

For each field bit y (fault input number):

0 = The fault y becomes active when the fault input y is at 0.

1 = The fault y becomes active when the fault input y is at 1.

- **FMOD: Fault Activation Mode (fault input bit varies from 0 to 7)**

For each field bit y (fault input number):

0 = The fault y is active until the Fault condition is removed at the peripheral<sup>(1)</sup> level.

1 = The fault y stays active until the Fault condition is removed at the peripheral<sup>(1)</sup> level AND until it is cleared in the “[PWM Fault Clear Register](#)” .

Note: 1. The Peripheral generating the fault.

- **FFIL: Fault Filtering (fault input bit varies from 0 to 7)**

For each field bit y (fault input number):

0 = The fault input y is not filtered.

1 = The fault input y is filtered.

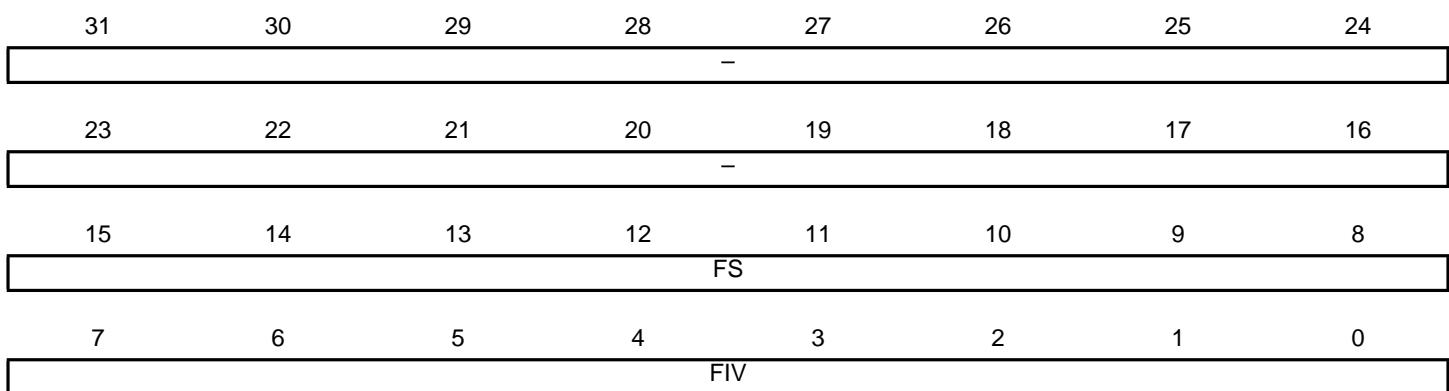
**CAUTION:** To prevent an unexpected activation of the status flag FSy in the “[PWM Fault Status Register](#)” on page [1059](#), the bit FMODY can be set to “1” only if the FPOLy bit has been previously configured to its final value.

#### 40.7.24 PWM Fault Status Register

**Name:** PWM\_FSR

**Address:** 0x40000060

**Access:** Read-only



- **FIV: Fault Input Value (fault input bit varies from 0 to 7)**

For each field bit y (fault input number):

0 = The current sampled value of the fault input y is 0 (after filtering if enabled).

1 = The current sampled value of the fault input y is 1 (after filtering if enabled).

- **FS: Fault Status (fault input bit varies from 0 to 7)**

For each field bit y (fault input number):

0 = The fault y is not currently active.

1 = The fault y is currently active.

#### 40.7.25 PWM Fault Clear Register

**Name:** PWM\_FCR

**Address:** 0x40000064

**Access:** Write-only

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
				FCLR			

- **FCLR: Fault Clear (fault input bit varies from 0 to 7)**

For each field bit y (fault input number):

0 = No effect.

1 = If bit y of FMOD field is set to 1 and if the fault input y is not at the level defined by the bit y of FPOL field, the fault y is cleared and becomes inactive (FMOD and FPOL fields belong to “[PWM Fault Mode Register](#)” on page 1058), else writing this bit to 1 has no effect.

#### 40.7.26 PWM Fault Protection Value Register 1

**Name:** PWM\_FPV1

**Address:** 0x40000068

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FPVL3	FPVL2	FPVL1	FPVL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	FPVH3	FPVH2	FPVH1	FPVH0

This register can only be written if the bits WPSWS5 and WPHWS5 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

- **FPVHx: Fault Protection Value for PWMH output on channel x**

This bit is taken into account only if the bit FPZHx is set to 0 in “[PWM Fault Protection Value Register 2](#)” on page [1067](#).

0 = PWMH output of channel x is forced to 0 when fault occurs.

1 = PWMH output of channel x is forced to 1 when fault occurs.

- **FPVLx: Fault Protection Value for PWML output on channel x**

This bit is taken into account only if the bit FPZLx is set to 0 in “[PWM Fault Protection Value Register 2](#)” on page [1067](#).

0 = PWML output of channel x is forced to 0 when fault occurs.

1 = PWML output of channel x is forced to 1 when fault occurs.

#### 40.7.27 PWM Fault Protection Enable Register

**Name:** PWM\_FPE  
**Address:** 0x4000006C  
**Access:** Read-write

31	30	29	28	27	26	25	24
FPE3							
23	22	21	20	19	18	17	16
FPE2							
15	14	13	12	11	10	9	8
FPE1							
7	6	5	4	3	2	1	0
FPE0							

This register can only be written if the bits WPSWS5 and WPHWS5 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

Only the first 8 bits (number of fault input pins) of fields FPE0, FPE1, FPE2 and FPE3 are significant.

- **FPE<sub>x</sub>: Fault Protection Enable for channel x (fault input bit varies from 0 to 7)**

For each field bit y (fault input number):

0 = Fault y is not used for the Fault Protection of channel x.

1 = Fault y is used for the Fault Protection of channel x.

**CAUTION:** To prevent an unexpected activation of the Fault Protection, the bit y of FPE<sub>x</sub> field can be set to “1” only if the corresponding FPOL bit has been previously configured to its final value in “[PWM Fault Mode Register](#)” on page [1058](#).

#### 40.7.28 PWM Event Line x Register

**Name:** PWM\_ELMRx

**Address:** 0x4000007C

**Access:** Read-write

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
CSEL7	CSEL6	CSEL5	CSEL4	CSEL3	CSEL2	CSEL1	CSEL0

- **CSELy: Comparison y Selection**

0 = A pulse is not generated on the event line x when the comparison y matches.

1 = A pulse is generated on the event line x when the comparison y match.

#### 40.7.29 PWM Spread Spectrum Register

**Name:** PWM\_SSPR

**Address:** 0x400000A0

**Access:** Read-write

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

This register can only be written if the bits WPSWS3 and WPHWS3 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

Only the first 16 bits (channel counter size) are significant.

- **SPRD: Spread Spectrum Limit Value**

The spread spectrum limit value defines the range for the spread spectrum counter. It is introduced in order to achieve constant varying PWM period for the output waveform.

- **SPRDM: Spread Spectrum Counter Mode**

0 = TRIANGULAR mode. The spread spectrum counter starts to count from -SPRD when the channel 0 is enabled and counts upwards at each PWM period. When it reaches +SPRD, it restarts to count from -SPRD again.

1 = RANDOM mode. The spread spectrum counter is loaded with a new random value at each PWM period. This random value is uniformly distributed and is between -SPRD and +SPRD.

#### 40.7.30 PWM Spread Spectrum Update Register

**Name:** PWM\_SSPUP

**Address:** 0x400000A4

**Access:** Write-only

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

This register can only be written if the bits WPSWS3 and WPHWS3 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

This register acts as a double buffer for the SPRD value. This prevents an unexpected waveform when modifying the spread spectrum limit value.

Only the first 16 bits (channel counter size) are significant.

- **SPRDUP: Spread Spectrum Limit Value Update**

The spread spectrum limit value defines the range for the spread spectrum counter. It is introduced in order to achieve constant varying period for the output waveform.

#### 40.7.31 PWM Stepper Motor Mode Register

**Name:** PWM\_SMMR

**Address:** 0x400000B0

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	DOWN1	DOWN0
15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	—
7	6	5	4	3	2	1	0
—	—	—	—	—	—	GCEN1	GCEN0

- **GCEN<sub>x</sub>: Gray Count ENable**

0 = Disable gray count generation on PWML[2<sup>x</sup>], PWMH[2<sup>x</sup>], PWML[2<sup>x</sup>+1], PWMH[2<sup>x</sup>+1]

1 = enable gray count generation on PWML[2<sup>x</sup>], PWMH[2<sup>x</sup>], PWML[2<sup>x</sup>+1], PWMH[2<sup>x</sup>+1].

- **DOWN<sub>x</sub>: DOWN Count**

0 = Up counter.

1 = Down counter.

#### 40.7.32 PWM Fault Protection Value Register 2

**Name:** PWM\_FPV2

**Address:** 0x400000C0

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FPZL3	FPZL2	FPZL1	FPZL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	FPZH3	FPZH2	FPZH1	FPZH0

This register can only be written if the bits WPSWS5 and WPHWS5 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

- **FPZHx: Fault Protection to Hi-Z for PWMH output on channel x**

0 = When fault occurs, PWMH output of channel x is forced to value defined by the bit FPVHx in “[PWM Fault Protection Value Register 1](#)” on page [1061](#).

1 = When fault occurs, PWMH output of channel x is forced to high-impedance state.

- **FPZLx: Fault Protection to Hi-Z for PWML output on channel x**

0 = When fault occurs, PWML output of channel x is forced to value defined by the bit FPVLx in “[PWM Fault Protection Value Register 1](#)” on page [1061](#).

1 = When fault occurs, PWML output of channel x is forced to high-impedance state.

#### 40.7.33 PWM Write Protect Control Register

**Name:** PWM\_WPCR

**Address:** 0x400000E4

**Access:** Write-only

31	30	29	28	27	26	25	24
WPKEY							
23	22	21	20	19	18	17	16
WPKEY							
15	14	13	12	11	10	9	8
WPKEY							
7	6	5	4	3	2	1	0
WPRG5	WPRG4	WPRG3	WPRG2	WPRG1	WPRG0	WPCMD	

- **WPCMD: Write Protect Command**

This command is performed only if the WPKEY value is correct.

0 = Disable the Write Protect SW of the register groups of which the bit WPRGx is at 1.

1 = Enable the Write Protect SW of the register groups of which the bit WPRGx is at 1.

2 = Enable the Write Protect HW of the register groups of which the bit WPRGx is at 1.

Moreover, to meet security requirements, in this mode of operation, the PIO lines associated with PWM can not be configured through the PIO interface, not even by the PIO controller.

3 = No effect.

Note: Only a hardware reset of the PWM controller can disable the Write Protect HW.

- **WPRGx: Write Protect Register Group x**

0 = The WPCMD command has no effect on the register group x.

1 = The WPCMD command is applied to the register group x.

- **WPKEY: Write Protect Key**

Should be written at value 0x50574D ("PWM" in ASCII). Writing any other value in this field aborts the write operation of the WPCMD field. Always reads as 0.

List of register groups:

- Register group 0:
  - ["PWM Clock Register" on page 1037](#)
- Register group 1:
  - ["PWM Disable Register" on page 1039](#)
- Register group 2:
  - ["PWM Sync Channels Mode Register" on page 1044](#)
  - ["PWM Channel Mode Register" on page 1075](#)
  - ["PWM Stepper Motor Mode Register" on page 1066](#)
  - ["PWM Channel Mode Update Register" on page 1084](#)
- Register group 3:

- “[PWM Spread Spectrum Register](#)” on page 1064
  - “[PWM Spread Spectrum Update Register](#)” on page 1065
  - “[PWM Channel Period Register](#)” on page 1079
  - “[PWM Channel Period Update Register](#)” on page 1080
- Register group 4:
    - “[PWM Channel Dead Time Register](#)” on page 1082
    - “[PWM Channel Dead Time Update Register](#)” on page 1083
  - Register group 5:
    - “[PWM Fault Mode Register](#)” on page 1058
    - “[PWM Fault Protection Value Register 1](#)” on page 1061

#### 40.7.34 PWM Write Protect Status Register

**Name:** PWM\_WPSR

**Address:** 0x400000E8

**Access:** Read-only

31	30	29	28	27	26	25	24
WPVSRC							
23	22	21	20	19	18	17	16
WPVSRC							
15	14	13	12	11	10	9	8
—	—	WPHWS5	WPHWS4	WPHWS3	WPHWS2	WPHWS1	WPHWS0
7	6	5	4	3	2	1	0
WPVS	—	WPSWS5	WPSWS4	WPSWS3	WPSWS2	WPSWS1	WPSWS0

- **WPSWS<sub>x</sub>: Write Protect SW Status**

0 = The Write Protect SW  $x$  of the register group  $x$  is disabled.

1 = The Write Protect SW  $x$  of the register group  $x$  is enabled.

- **WPHWS<sub>x</sub>: Write Protect HW Status**

0 = The Write Protect HW  $x$  of the register group  $x$  is disabled.

1 = The Write Protect HW  $x$  of the register group  $x$  is enabled.

- **WPVS: Write Protect Violation Status**

0 = No Write Protect violation has occurred since the last read of the PWM\_WPSR register.

1 = At least one Write Protect violation has occurred since the last read of the PWM\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset) in which a write access has been attempted.

Notes:

1. The two LSBs of the address offset of the write-protected register are not reported
2. Reading PWM\_WPSR automatically clears WPVS and WPVSRC fields.

#### 40.7.35 PWM Comparison x Value Register

**Name:** PWM\_CMPVx

**Address:** 0x40000130 [0], 0x40000140 [1], 0x40000150 [2], 0x40000160 [3], 0x40000170 [4], 0x40000180 [5],  
0x40000190 [6], 0x400001A0 [7]

**Access:** Read-write

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

Only the first 16 bits (channel counter size) of field CV are significant.

- **CV: Comparison x Value**

Define the comparison x value to be compared with the counter of the channel 0.

- **CVM: Comparison x Value Mode**

0 = The comparison x between the counter of the channel 0 and the comparison x value is performed when this counter is incrementing.

1 = The comparison x between the counter of the channel 0 and the comparison x value is performed when this counter is decrementing.

Note: This bit is useless if the counter of the channel 0 is left aligned (CALG = 0 in “[PWM Channel Mode Register](#)” on page [1075](#))

#### 40.7.36 PWM Comparison x Value Update Register

**Name:** PWM\_CMPVUPDx

**Address:** 0x40000134 [0], 0x40000144 [1], 0x40000154 [2], 0x40000164 [3], 0x40000174 [4], 0x40000184 [5],  
0x40000194 [6], 0x400001A4 [7]

**Access:** Write-only

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

This register acts as a double buffer for the CV and CVM values. This prevents an unexpected comparison x match.

Only the first 16 bits (channel counter size) of field CVUPD are significant.

- **CVUPD: Comparison x Value Update**

Define the comparison x value to be compared with the counter of the channel 0.

- **CVMUPD: Comparison x Value Mode Update**

0 = The comparison x between the counter of the channel 0 and the comparison x value is performed when this counter is incrementing.

1 = The comparison x between the counter of the channel 0 and the comparison x value is performed when this counter is decrementing.

Note: This bit is useless if the counter of the channel 0 is left aligned (CALG = 0 in “[PWM Channel Mode Register](#)” on page [1075](#))

**CAUTION:** To be taken into account, the write of the register PWM\_CMPVUPDx must be followed by a write of the register PWM\_CMPMUPDx.

#### 40.7.37 PWM Comparison x Mode Register

**Name:** PWM\_CMPMx

**Address:** 0x40000138 [0], 0x40000148 [1], 0x40000158 [2], 0x40000168 [3], 0x40000178 [4], 0x40000188 [5],  
0x40000198 [6], 0x400001A8 [7]

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
CUPRCNT				CUPR			
15	14	13	12	11	10	9	8
CPRCNT				CPR			
7	6	5	4	3	2	1	0
CTR				-	-	-	CEN

- **CEN: Comparison x Enable**

0 = The comparison x is disabled and can not match.

1 = The comparison x is enabled and can match.

- **CTR: Comparison x Trigger**

The comparison x is performed when the value of the comparison x period counter (CPRCNT) reaches the value defined by CTR.

- **CPR: Comparison x Period**

CPR defines the maximum value of the comparison x period counter (CPRCNT). The comparison x value is performed periodically once every CPR+1 periods of the channel 0 counter.

- **CPRCNT: Comparison x Period Counter**

Reports the value of the comparison x period counter.

Note: The field CPRCNT is read-only

- **CUPR: Comparison x Update Period**

Defines the time between each update of the comparison x mode and the comparison x value. This time is equal to CUPR+1 periods of the channel 0 counter.

- **CUPRCNT: Comparison x Update Period Counter**

Reports the value of the comparison x update period counter.

Note: The field CUPRCNT is read-only

#### 40.7.38 PWM Comparison x Mode Update Register

Name:	PWM_CMPMUPDx							
Address:	0x4000013C [0], 0x4000014C [1], 0x4000015C [2], 0x4000016C [3], 0x4000017C [4], 0x4000018C [5], 0x4000019C [6], 0x400001AC [7]							
Access:	Write-only							
31	30	29	28	27	26	25	24	
-	-	-	-	-	-	-	-	
23	22	21	20	19	18	17	16	
-	-	-	-		CUPRUPD			
15	14	13	12	11	10	9	8	
-	-	-	-		CPRUPD			
7	6	5	4	3	2	1	0	
	CTRUPD			-	-	-	-	CENUPD

This register acts as a double buffer for the CEN, CTR, CPR and CUPR values. This prevents an unexpected comparison x match.

- **CENUPD: Comparison x Enable Update**

0 = The comparison x is disabled and can not match.

1 = The comparison x is enabled and can match.

- **CTRUPD: Comparison x Trigger Update**

The comparison x is performed when the value of the comparison x period counter (CPRCNT) reaches the value defined by CTR.

- **CPRUPD: Comparison x Period Update**

CPR defines the maximum value of the comparison x period counter (CPRCNT). The comparison x value is performed periodically once every CPR+1 periods of the channel 0 counter.

- **CUPRUPD: Comparison x Update Period Update**

Defines the time between each update of the comparison x mode and the comparison x value. This time is equal to CUPR+1 periods of the channel 0 counter.

#### 40.7.39 PWM Channel Mode Register

**Name:** PWM\_CMRx [x=0..3]

**Address:** 0x40000200 [0], 0x40000220 [1], 0x40000240 [2], 0x40000260 [3]

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	DTLI	DTHI	DTE
15	14	13	12	11	10	9	8
—	—	—	—	UPDS	CES	CPOL	CALG
7	6	5	4	3	2	1	0
—	—	—	—	CPRE			

This register can only be written if the bits WPSWS2 and WPHWS2 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

- **CPRE: Channel Pre-scaler**

Value	Name	Description
0b0000	MCK	Master clock
0b0001	MCK_DIV_2	Master clock/2
0b0010	MCK_DIV_4	Master clock/4
0b0011	MCK_DIV_8	Master clock/8
0b0100	MCK_DIV_16	Master clock/16
0b0101	MCK_DIV_32	Master clock/32
0b0110	MCK_DIV_64	Master clock/64
0b0111	MCK_DIV_128	Master clock/128
0b1000	MCK_DIV_256	Master clock/256
0b1001	MCK_DIV_512	Master clock/512
0b1010	MCK_DIV_1024	Master clock/1024
0b1011	CLKA	Clock A
0b1100	CLKB	Clock B

- **CALG: Channel Alignment**

0 = The period is left aligned.

1 = The period is center aligned.

- **CPOL: Channel Polarity**

0 = The OCx output waveform (output from the comparator) starts at a low level.

1 = The OCx output waveform (output from the comparator) starts at a high level.

- **CES: Counter Event Selection**

The bit CES defines when the channel counter event occurs when the period is center aligned (flag CHIDx in the “[PWM Interrupt Status Register 1](#)” on page 1043).

CALG = 0 (Left Alignment):

0/1 = The channel counter event occurs at the end of the PWM period.

CALG = 1 (Center Alignment):

0 = The channel counter event occurs at the end of the PWM period.

1 = The channel counter event occurs at the end of the PWM period and at half the PWM period.

- **UPDS: Update Selection**

When the period is center aligned, the bit UPDS defines when the update of the duty cycle, the polarity and the additional edge value/mode occurs after writing the corresponding update registers.

CALG = 0 (Left Alignment):

0/1 = The update always occurs at the end of the PWM period after writing the update register(s).

CALG = 1 (Center Alignment):

0 = The update occurs at the next end of the PWM period after writing the update register(s).

1 = The update occurs at the next end of the PWM half period after writing the update register(s).

- **DTE: Dead-Time Generator Enable**

0 = The dead-time generator is disabled.

1 = The dead-time generator is enabled.

- **DTHI: Dead-Time PWMHx Output Inverted**

0 = The dead-time PWMHx output is not inverted.

1 = The dead-time PWMHx output is inverted.

- **DTLI: Dead-Time PWMLx Output Inverted**

0 = The dead-time PWMLx output is not inverted.

1 = The dead-time PWMLx output is inverted.

#### 40.7.40 PWM Channel Duty Cycle Register

**Name:** PWM\_CDTYx [x=0..3]

**Address:** 0x40000204 [0], 0x40000224 [1], 0x40000244 [2], 0x40000264 [3]

**Access:** Read-write

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

Only the first 16 bits (channel counter size) are significant.

- **CDTY: Channel Duty-Cycle**

Defines the waveform duty-cycle. This value must be defined between 0 and CPRD (PWM\_CPRx).

#### 40.7.41 PWM Channel Duty Cycle Update Register

**Name:** PWM\_CDTYUPDx [x=0..3]

**Address:** 0x40000208 [0], 0x40000228 [1], 0x40000248 [2], 0x40000268 [3]

**Access:** Write-only.

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

This register acts as a double buffer for the CDTY value. This prevents an unexpected waveform when modifying the waveform duty-cycle.

Only the first 16 bits (channel counter size) are significant.

- **CDTYUPD: Channel Duty-Cycle Update**

Defines the waveform duty-cycle. This value must be defined between 0 and CPRD (PWM\_CPRx).

#### 40.7.42 PWM Channel Period Register

**Name:** PWM\_CPRDx [x=0..3]

**Address:** 0x4000020C [0], 0x4000022C [1], 0x4000024C [2], 0x4000026C [3]

**Access:** Read-write

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

This register can only be written if the bits WPSWS3 and WPHWS3 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

Only the first 16 bits (channel counter size) are significant.

- **CPRD: Channel Period**

If the waveform is left-aligned, then the output waveform period depends on the channel counter source clock and can be calculated:

- By using the PWM master clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(X \times CPRD)}{MCK}$$

- By using the PWM master clock (MCK) divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(CRPD \times DIVA)}{MCK} \text{ or } \frac{(CRPD \times DIVB)}{MCK}$$

If the waveform is center-aligned, then the output waveform period depends on the channel counter source clock and can be calculated:

- By using the PWM master clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(2 \times X \times CPRD)}{MCK}$$

- By using the PWM master clock (MCK) divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(2 \times CPRD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRD \times DIVB)}{MCK}$$

#### 40.7.43 PWM Channel Period Update Register

**Name:** PWM\_CPRDUPD $x$  [ $x=0..3$ ]

**Address:** 0x40000210 [0], 0x40000230 [1], 0x40000250 [2], 0x40000270 [3]

**Access:** Write-only

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

This register can only be written if the bits WPSWS3 and WPHWS3 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

This register acts as a double buffer for the CPRD value. This prevents an unexpected waveform when modifying the waveform period.

Only the first 16 bits (channel counter size) are significant.

- **CPRDUPD: Channel Period Update**

If the waveform is left-aligned, then the output waveform period depends on the channel counter source clock and can be calculated:

- By using the PWM master clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(X \times CPRDUPD)}{MCK}$$

- By using the PWM master clock (MCK) divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(CRPDUPD \times DIVA)}{MCK} \text{ or } \frac{(CRPDUPD \times DIVB)}{MCK}$$

If the waveform is center-aligned, then the output waveform period depends on the channel counter source clock and can be calculated:

- By using the PWM master clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(2 \times X \times CPRDUPD)}{MCK}$$

- By using the PWM master clock (MCK) divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(2 \times CPRDUPD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRDUPD \times DIVB)}{MCK}$$

#### 40.7.44 PWM Channel Counter Register

**Name:** PWM\_CCNTx [x=0..3]

**Address:** 0x40000214 [0], 0x40000234 [1], 0x40000254 [2], 0x40000274 [3]

**Access:** Read-only

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

Only the first 16 bits (channel counter size) are significant.

- **CNT: Channel Counter Register**

Channel counter value. This register is reset when:

- The channel is enabled (writing CHIDx in the PWM\_ENA register).
- The channel counter reaches CPRD value defined in the PWM\_CPRDx register if the waveform is left aligned.

#### 40.7.45 PWM Channel Dead Time Register

**Name:** PWM\_DTx [x=0..3]

**Address:** 0x40000218 [0], 0x40000238 [1], 0x40000258 [2], 0x40000278 [3]

**Access:** Read-write

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

This register can only be written if the bits WPSWS4 and WPHWS4 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

Only the first 12 bits (dead-time counter size) of fields DTH and DTL are significant.

- **DTH: Dead-Time Value for PWMHx Output**

Defines the dead-time value for PWMHx output. This value must be defined between 0 and CPRD-CDTY (PWM\_CPRx and PWM\_CDTYx).

- **DTL: Dead-Time Value for PWMLx Output**

Defines the dead-time value for PWMLx output. This value must be defined between 0 and CDTY (PWM\_CDTYx).

#### 40.7.46 PWM Channel Dead Time Update Register

**Name:** PWM\_DTUPD<sub>x</sub> [x=0..3]

**Address:** 0x4000021C [0], 0x4000023C [1], 0x4000025C [2], 0x4000027C [3]

**Access:** Write-only

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

This register can only be written if the bits WPSWS4 and WPHWS4 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

This register acts as a double buffer for the DTH and DTL values. This prevents an unexpected waveform when modifying the dead-time values.

Only the first 12 bits (dead-time counter size) of fields DTHUPD and DTLUPD are significant.

- **DTHUPD: Dead-Time Value Update for PWMH<sub>x</sub> Output**

Defines the dead-time value for PWMH<sub>x</sub> output. This value must be defined between 0 and CPRD-CDTY (PWM\_CPR<sub>x</sub> and PWM\_CDTY<sub>x</sub>). This value is applied only at the beginning of the next channel x PWM period.

- **DTLUPD: Dead-Time Value Update for PWML<sub>x</sub> Output**

Defines the dead-time value for PWML<sub>x</sub> output. This value must be defined between 0 and CDTY (PWM\_CDTY<sub>x</sub>). This value is applied only at the beginning of the next channel x PWM period.

#### 40.7.47 PWM Channel Mode Update Register

**Name:** PWM\_CMUPDx [x=0..3]

**Address:** 0x40000400 [0], 0x40000420 [1], 0x40000440 [2], 0x40000460 [3]

**Access:** Read-write

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

This register can only be written if the bits WPSWS2 and WPHWS2 are cleared in “[PWM Write Protect Status Register](#)” on page [1070](#).

This register acts as a double buffer for the CPOL value. This prevents an unexpected waveform when modifying the polarity value.

- **CPOLUP: Channel Polarity Update**

The write of this bit is taken into account only if the bit CPOLINVUP is written at 0 at the same time.

0 = The OCx output waveform (output from the comparator) starts at a low level.

1 = The OCx output waveform (output from the comparator) starts at a high level.

- **CPOLINVUP: Channel Polarity Inversion Update**

If this bit is written at 1, the write of the bit CPOLUP is not taken into account.

0 = No effect.

1 = The OCx output waveform (output from the comparator) is inverted.

#### 40.7.48 PWM Channel Additional Edge Register

**Name:** PWM\_CAE<sub>x</sub> [x=0..3]

**Address:** 0x40000404 [0], 0x40000424 [1], 0x40000444 [2], 0x40000464 [3]

**Access:** Read-write

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

Only the first 16 bits (channel counter size) are significant.

- **ADEDGV: Channel Additional Edge Value**

Defines the timing of the additional edge. After writing this register, the channel polarity is inverted when the channel counter reaches the value defined by this field, that leads to an additional edge of the channel output waveform. This value must be defined between 0 and CPRD (PWM\_CPR<sub>x</sub>). The additional edge occurs only one time after writing this register.

- **ADEDGM: Channel Additional Edge Mode**

Value	Name	Description
0	INC	The additional edge of the channel x output waveform occurs when CCNT <sub>x</sub> reaches ADEDGV and the counter of the channel x is incrementing.
1	DEC	The additional edge of the channel x output waveform occurs when CCNT <sub>x</sub> reaches ADEDGV and the counter of the channel x is incrementing.
2	BOTH	The additional edge of the channel x output waveform occurs when CCNT <sub>x</sub> reaches ADEDGV, whether the counter is incrementing or not.
3	–	Reserved

Note: This field is useless if the counter of the channel x is left aligned (CALG = 0 in “[PWM Channel Mode Register](#)” on [page 1075](#))

#### 40.7.49 PWM Channel Additional Edge Update Register

**Name:** PWM\_CAEUPDx [x=0..3]

**Address:** 0x40000408 [0], 0x40000428 [1], 0x40000448 [2], 0x40000468 [3]

**Access:** Write-only

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

This register acts as a double buffer for the ADEDGV and ADEDGM values. This prevents an unexpected waveform when modifying the additional edge value and the additional edge mode.

Only the first 16 bits (channel counter size) are significant.

- **ADEDGVUP: Channel Additional Edge Value Update**

Defines the timing of the additional edge. After writing this register, the channel polarity is inverted when the channel counter reaches the value defined by this field, that leads to an additional edge of the channel output waveform. This value must be defined between 0 and CPRD (PWM\_CPRx). The additional edge occurs only one time after writing this register.

- **ADEDGMUP: Channel Additional Edge Mode Update**

Value	Name	Description
0	INC	The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP and the counter of the channel x is incrementing.
1	DEC	The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP and the counter of the channel x is incrementing.
2	BOTH	The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP, whether the counter is incrementing or not.
3	–	Reserved

Note: This field is useless if the counter of the channel x is left aligned (CALG = 0 in [“PWM Channel Mode Register” on page 1075](#))

## 41. USB Device Port (UDP)

### 41.1 Description

The USB Device Port (UDP) is compliant with the Universal Serial Bus (USB) V2.0 full-speed device specification.

Each endpoint can be configured in one of several USB transfer types. It can be associated with one or two banks of a dual-port RAM used to store the current data payload. If two banks are used, one DPR bank is read or written by the processor, while the other is read or written by the USB device peripheral. This feature is mandatory for isochronous endpoints. Thus the device maintains the maximum bandwidth (1M bytes/s) by working with endpoints with two banks of DPR.

Table 41-1. USB Endpoint Description

Endpoint Number	Mnemonic	Dual-Bank <sup>(1)</sup>	Max. Endpoint Size	Endpoint Type
0	EP0	No	64	Control/Bulk/Interrupt
1	EP1	Yes	64	Bulk/Iso/Interrupt
2	EP2	Yes	64	Bulk/Iso/Interrupt
3	EP3	No	64	Control/Bulk/Interrupt
4	EP4	Yes	512	Bulk/Iso/Interrupt
5	EP5	Yes	512	Bulk/Iso/Interrupt
6	EP6	Yes	64	Bulk/Iso/Interrupt
7	EP7	Yes	64	Bulk/Iso/Interrupt

Note: 1. The Dual-Bank function provides two banks for an endpoint. This feature is used for ping-pong mode.

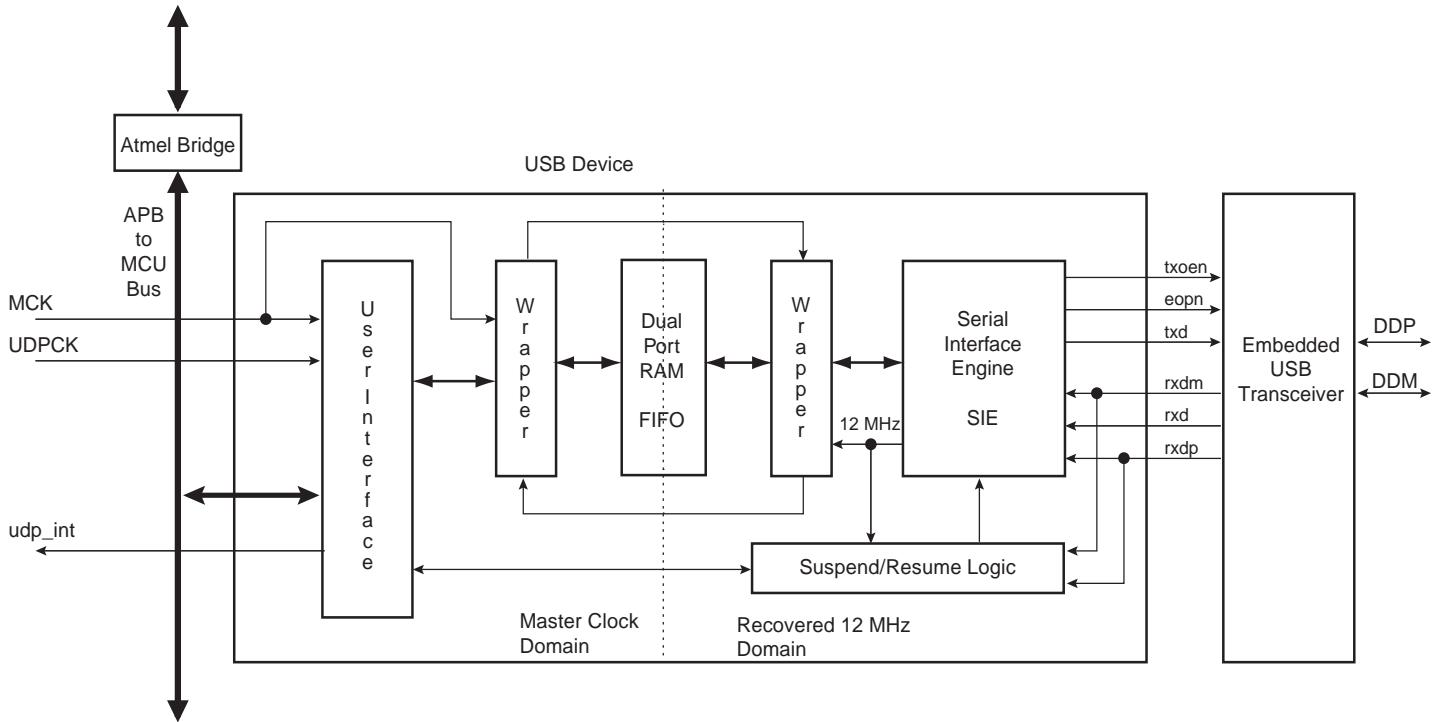
Suspend and resume are automatically detected by the USB device, which notifies the processor by raising an interrupt. Depending on the product, an external signal can be used to send a wake up to the USB host controller.

### 41.2 Embedded Characteristics

- USB V2.0 Full-speed Compliant, 12 Mbps
- Embedded USB V2.0 Full-speed Transceiver
- Integrated Pull-up on DP
- 8 Endpoints
- Embedded Dual-port RAM for Endpoints
- Suspend/Resume Logic
- Ping-pong Mode (2 Memory Banks) for Isochronous and Bulk Endpoints

## 41.3 Block Diagram

Figure 41-1. Block Diagram



Access to the UDP is via the APB bus interface. Read and write to the data FIFO are done by reading and writing 8-bit values to APB registers.

The UDP peripheral requires two clocks: one peripheral clock used by the Master Clock domain (MCK) and a 48 MHz clock (UDPCK) used by the 12 MHz domain.

A USB 2.0 full-speed pad is embedded and controlled by the Serial Interface Engine (SIE).

The signal **external\_resume** is optional. It allows the UDP peripheral to wake up once in system mode. The host is then notified that the device asks for a resume. This optional feature must also be negotiated with the host during the enumeration.

### 41.3.1 Signal Description

Table 41-2. Signal Names

Signal Name	Description	Type
UDPCK	48 MHz clock	Input
MCK	Master clock	Input
udp_int	Interrupt line connected to the Interrupt Controller	Input
DDP	USB D+ line	I/O
DDM	USB D- line	I/O

## 41.4 Product Dependencies

For further details on the USB Device hardware implementation, see the specific Product Properties document.

The USB physical transceiver is integrated into the product. The bidirectional differential signals DDP and DDM are available from the product boundary.

One I/O line may be used by the application to check that VBUS is still available from the host. Self-powered devices may use this entry to be notified that the host has been powered off. In this case, the pull-up on DP must be disabled in order to prevent feeding current to the host. The application should disconnect the transceiver, then remove the pull-up.

#### 41.4.1 I/O Lines

The USB pins are shared with PIO lines. By default, the USB function is activated, and pins DDP and DDM are used for USB. To configure DDP or DDM as PIOs, the user needs to configure the system I/O configuration register (CCFG\_SYSIO) in the MATRIX.

#### 41.4.2 Power Management

The USB device peripheral requires a 48 MHz clock. This clock must be generated by a PLL with an accuracy of  $\pm 0.25\%$ .

Thus, the USB device receives two clocks from the Power Management Controller (PMC): the master clock, MCK, used to drive the peripheral user interface, and the UDPCK, used to interface with the bus USB signals (recovered 12 MHz domain).

**WARNING:** The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP\_TXVC register.

#### 41.4.3 Interrupt

The USB device interface has an interrupt line connected to the Interrupt Controller.

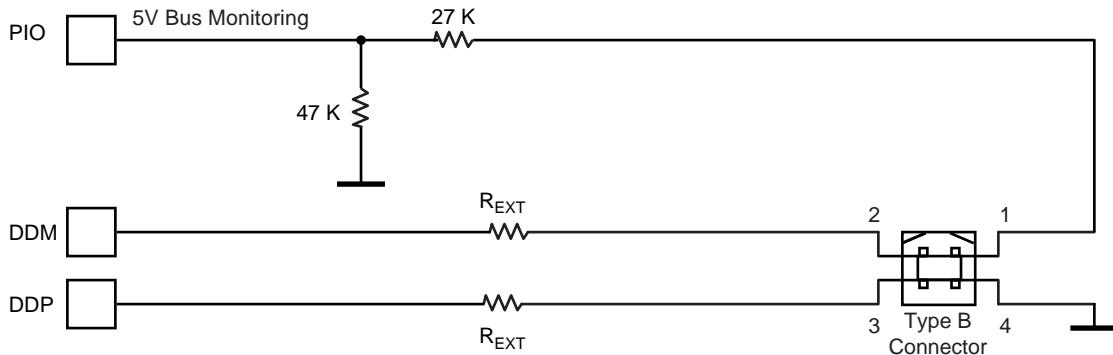
Handling the USB device interrupt requires programming the Interrupt Controller before configuring the UDP.

**Table 41-3. Peripheral IDs**

Instance	ID
UDP	35

## 41.5 Typical Connection

Figure 41-2. Board Schematic to Interface Device Peripheral



### 41.5.1 USB Device Transceiver

The USB device transceiver is embedded in the product. A few discrete components are required as follows:

- The application detects all device states as defined in chapter 9 of the USB specification;
  - VBUS monitoring
- To reduce power consumption the host is disconnected
- For line termination.

### 41.5.2 VBUS Monitoring

VBUS monitoring is required to detect host connection. VBUS monitoring is done using a standard PIO with internal pull-up disabled. When the host is switched off, it should be considered as a disconnect, the pull-up must be disabled in order to prevent powering the host through the pull-up resistor.

When the host is disconnected and the transceiver is enabled, then DDP and DDM are floating. This may lead to over consumption. A solution is to enable the integrated pull-down by disabling the transceiver (TXVDIS = 1) and then remove the pull-up (PUON = 0).

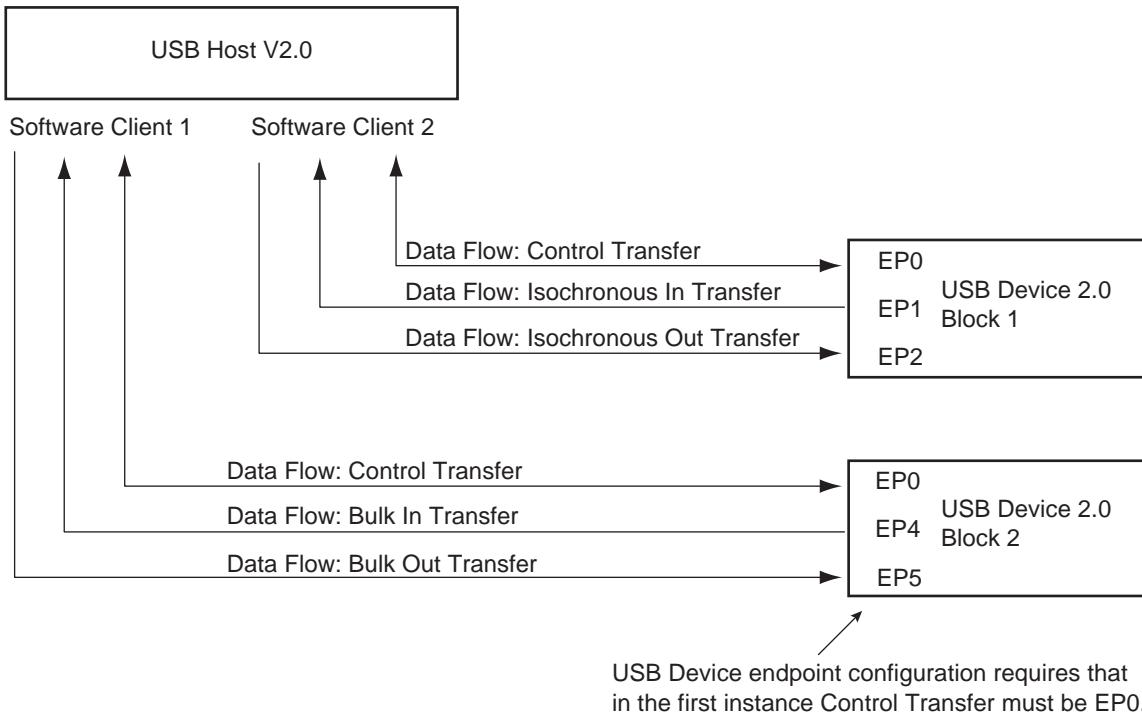
A termination serial resistor must be connected to DDP and DDM. The resistor value is defined in the electrical specification of the product ( $R_{EXT}$ ).

## 41.6 Functional Description

### 41.6.1 USB V2.0 Full-speed Introduction

The USB V2.0 full-speed provides communication services between host and attached USB devices. Each device is offered with a collection of communication flows (pipes) associated with each endpoint. Software on the host communicates with a USB device through a set of communication flows.

**Figure 41-3. Example of USB V2.0 Full-speed Communication Control**



The Control Transfer endpoint EP0 is always used when a USB device is first configured (USB v. 2.0 specifications).

#### 41.6.1.1 USB V2.0 Full-speed Transfer Types

A communication flow is carried over one of four transfer types defined by the USB device.

**Table 41-4. USB Communication Flow**

Transfer	Direction	Bandwidth	Supported Endpoint Size	Error Detection	Retrying
Control	Bidirectional	Not guaranteed	8, 16, 32, 64	Yes	Automatic
Isochronous	Unidirectional	Guaranteed	512	Yes	No
Interrupt	Unidirectional	Not guaranteed	$\leq 64$	Yes	Yes
Bulk	Unidirectional	Not guaranteed	8, 16, 32, 64	Yes	Yes

#### 41.6.1.2 USB Bus Transactions

Each transfer results in one or more transactions over the USB bus. There are three kinds of transactions flowing across the bus in packets:

1. Setup Transaction
2. Data IN Transaction
3. Data OUT Transaction

#### 41.6.1.3 USB Transfer Event Definitions

As indicated below, transfers are sequential events carried out on the USB bus.

**Table 41-5. USB Transfer Events**

Control Transfers <sup>(1) (3)</sup>	<ul style="list-style-type: none"> <li>● Setup transaction &gt; Data IN transactions &gt; Status OUT transaction</li> <li>● Setup transaction &gt; Data OUT transactions &gt; Status IN transaction</li> <li>● Setup transaction &gt; Status IN transaction</li> </ul>
Interrupt IN Transfer (device toward host)	<ul style="list-style-type: none"> <li>● Data IN transaction &gt; Data IN transaction</li> </ul>
Interrupt OUT Transfer (host toward device)	<ul style="list-style-type: none"> <li>● Data OUT transaction &gt; Data OUT transaction</li> </ul>
Isochronous IN Transfer <sup>(2)</sup> (device toward host)	<ul style="list-style-type: none"> <li>● Data IN transaction &gt; Data IN transaction</li> </ul>
Isochronous OUT Transfer <sup>(2)</sup> (host toward device)	<ul style="list-style-type: none"> <li>● Data OUT transaction &gt; Data OUT transaction</li> </ul>
Bulk IN Transfer (device toward host)	<ul style="list-style-type: none"> <li>● Data IN transaction &gt; Data IN transaction</li> </ul>
Bulk OUT Transfer (host toward device)	<ul style="list-style-type: none"> <li>● Data OUT transaction &gt; Data OUT transaction</li> </ul>

Notes:

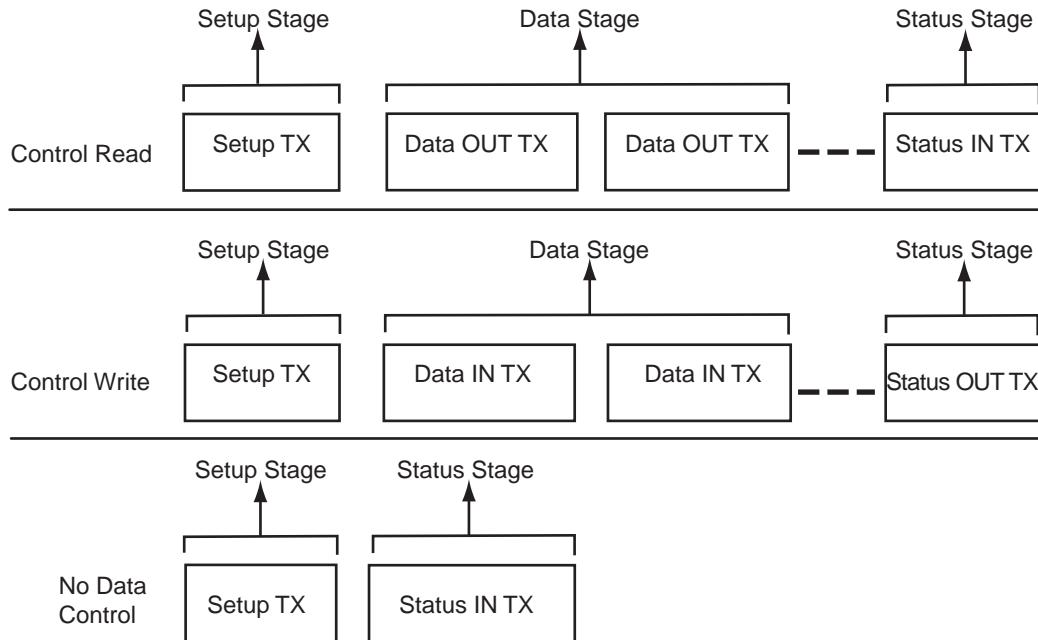
1. Control transfer must use endpoints with no ping-pong attributes.

2. Isochronous transfers must use endpoints with ping-pong attributes.

3. Control transfers can be aborted using a stall handshake.

A status transaction is a special type of host-to-device transaction used only in a control transfer. The control transfer must be performed using endpoints with no ping-pong attributes. According to the control sequence (read or write), the USB device sends or receives a status transaction.

**Figure 41-4. Control Read and Write Sequences**



Notes:

1. During the Status IN stage, the host waits for a zero length packet (Data IN transaction with no data) from the device using DATA1 PID. Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0*, for more information on the protocol layer.
2. During the Status OUT stage, the host emits a zero length packet to the device (Data OUT transaction with no data).

## 41.6.2 Handling Transactions with USB V2.0 Device Peripheral

### 41.6.2.1 Setup Transaction

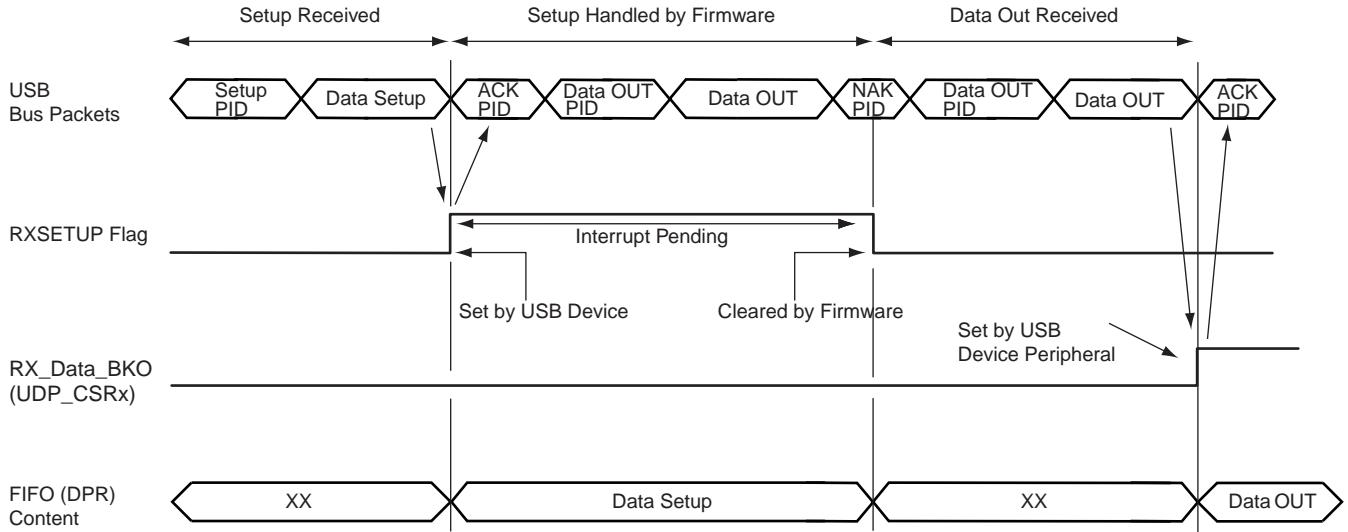
Setup is a special type of host-to-device transaction used during control transfers. Control transfers must be performed using endpoints with no ping-pong attributes. A setup transaction needs to be handled as soon as possible by the firmware. It is used to transmit requests from the host to the device. These requests are then handled by the USB device and may require more arguments. The arguments are sent to the device by a Data OUT transaction which follows the setup transaction. These requests may also return data. The data is carried out to the host by the next Data IN transaction which follows the setup transaction. A status transaction ends the control transfer.

When a setup transfer is received by the USB endpoint:

- The USB device automatically acknowledges the setup packet
- RXSETUP is set in the UDP\_CSRx register
- An endpoint interrupt is generated while the RXSETUP is not cleared. This interrupt is carried out to the microcontroller if interrupts are enabled for this endpoint.

Thus, firmware must detect the RXSETUP polling the UDP\_CSRx or catching an interrupt, read the setup packet in the FIFO, then clear the RXSETUP. RXSETUP cannot be cleared before the setup packet has been read in the FIFO. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the setup packet in the FIFO.

**Figure 41-5. Setup Transaction Followed by a Data OUT Transaction**



#### 41.6.2.2 Data IN Transaction

Data IN transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the device to the host. Data IN transactions in isochronous transfer must be done using endpoints with ping-pong attributes.

##### Using Endpoints Without Ping-pong Attributes

To perform a Data IN transaction using a non ping-pong endpoint:

1. The application checks if it is possible to write in the FIFO by polling TXPKTRDY in the endpoint's UDP\_CSRx register (TXPKTRDY must be cleared).
2. The application writes the first packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP\_FDRx register,
3. The application notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP\_CSRx register.
4. The application is notified that the endpoint's FIFO has been released by the USB device when TXCOMP in the endpoint's UDP\_CSRx register has been set. Then an interrupt for the corresponding endpoint is pending while TXCOMP is set.
5. The microcontroller writes the second packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP\_FDRx register,
6. The microcontroller notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP\_CSRx register.
7. The application clears the TXCOMP in the endpoint's UDP\_CSRx.

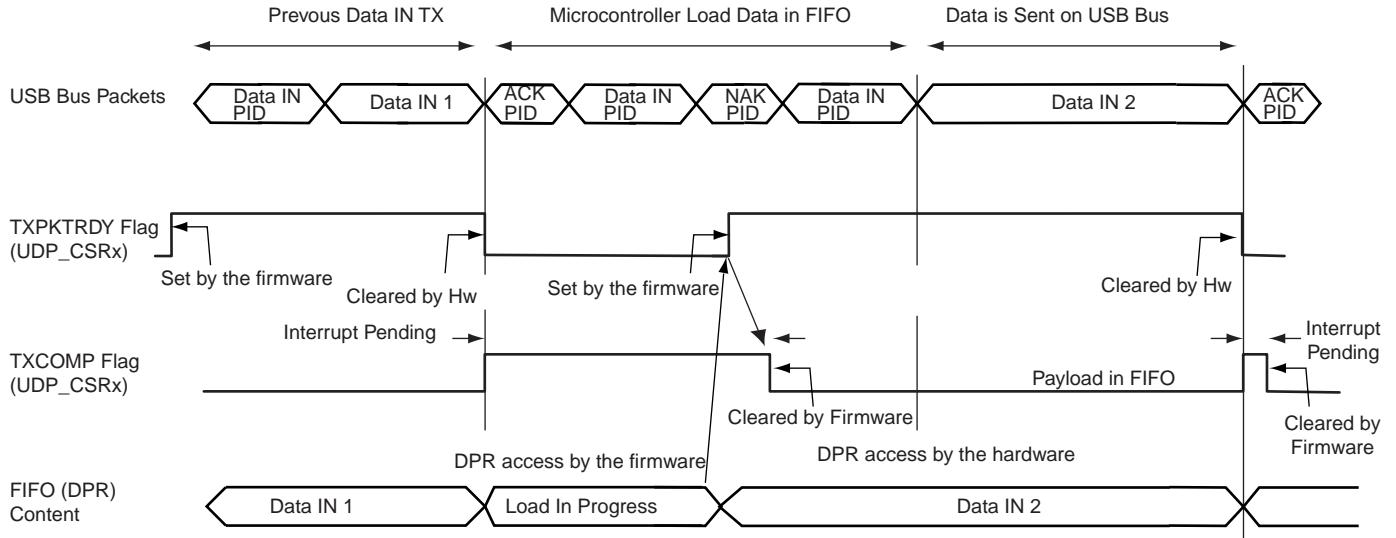
After the last packet has been sent, the application must clear TXCOMP once this has been set.

TXCOMP is set by the USB device when it has received an ACK PID signal for the Data IN packet. An interrupt is pending while TXCOMP is set.

**Warning:** TX\_COMP must be cleared after TX\_PKTRDY has been set.

Note: Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0*, for more information on the Data IN protocol layer.

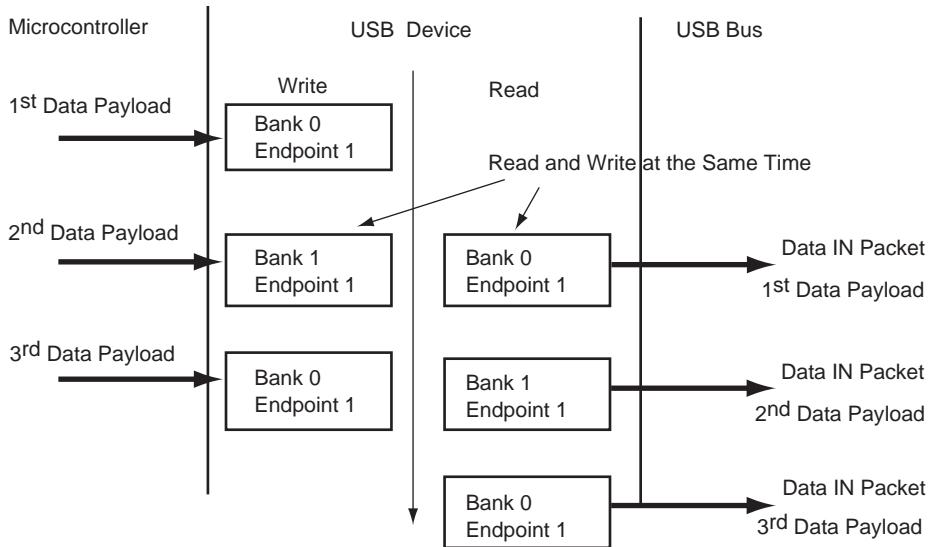
**Figure 41-6. Data IN Transfer for Non Ping-pong Endpoint**



### Using Endpoints With Ping-pong Attribute

The use of an endpoint with ping-pong attributes is necessary during isochronous transfer. This also allows handling the maximum bandwidth defined in the USB specification during bulk transfer. To be able to guarantee a constant or the maximum bandwidth, the microcontroller must prepare the next data payload to be sent while the current one is being sent by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

**Figure 41-7. Bank Swapping Data IN Transfer for Ping-pong Endpoints**

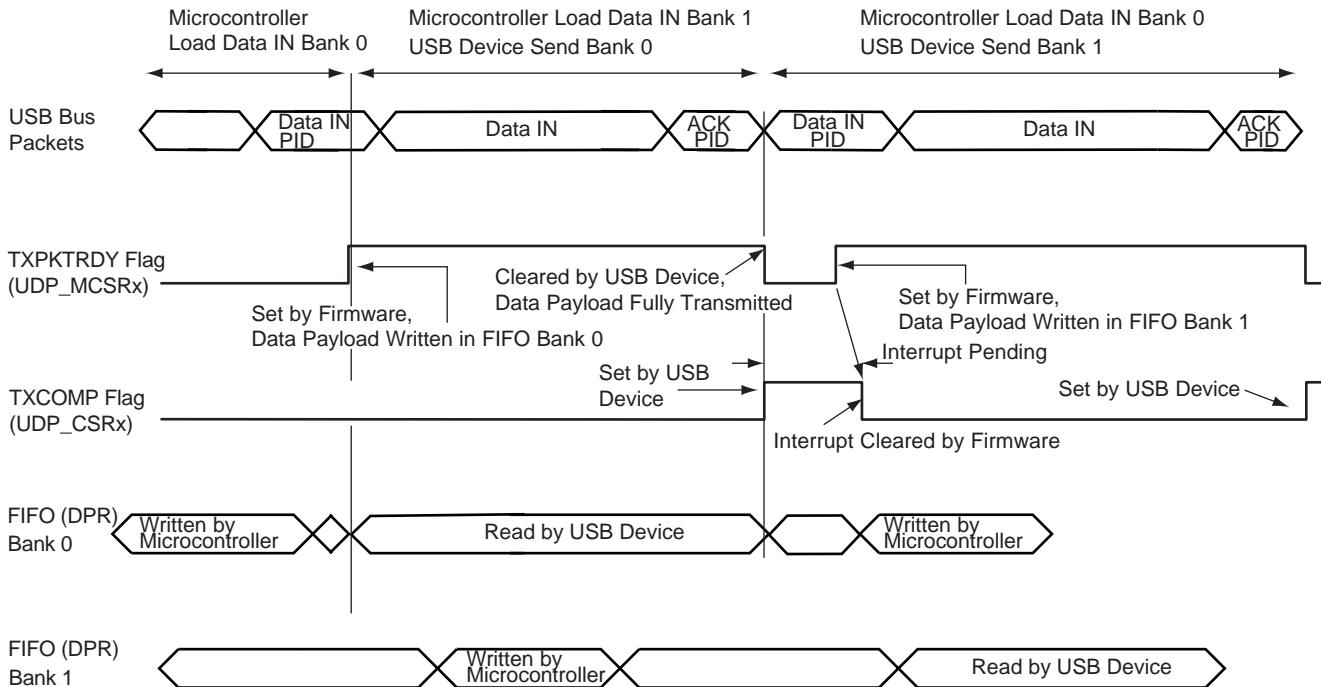


When using a ping-pong endpoint, the following procedures are required to perform Data IN transactions:

1. The microcontroller checks if it is possible to write in the FIFO by polling TXPKTRDY to be cleared in the endpoint's UDP\_CSRx register.
2. The microcontroller writes the first data payload to be sent in the FIFO (Bank 0), writing zero or more byte values in the endpoint's UDP\_FDRx register.

3. The microcontroller notifies the USB peripheral it has finished writing in Bank 0 of the FIFO by setting the TXPKTRDY in the endpoint's UDP\_CSRx register.
4. Without waiting for TXPKTRDY to be cleared, the microcontroller writes the second data payload to be sent in the FIFO (Bank 1), writing zero or more byte values in the endpoint's UDP\_FDRx register.
5. The microcontroller is notified that the first Bank has been released by the USB device when TXCOMP in the endpoint's UDP\_CSRx register is set. An interrupt is pending while TXCOMP is being set.
6. Once the microcontroller has received TXCOMP for the first Bank, it notifies the USB device that it has prepared the second Bank to be sent, raising TXPKTRDY in the endpoint's UDP\_CSRx register.
7. At this step, Bank 0 is available and the microcontroller can prepare a third data payload to be sent.

**Figure 41-8. Data IN Transfer for Ping-pong Endpoint**



**Warning:** There is software critical path due to the fact that once the second bank is filled, the driver has to wait for TX\_COMP to set TX\_PKTRDY. If the delay between receiving TX\_COMP is set and TX\_PKTRDY is set too long, some Data IN packets may be NACKed, reducing the bandwidth.

**Warning:** TX\_COMP must be cleared after TX\_PKTRDY has been set.

#### 41.6.2.3 Data OUT Transaction

Data OUT transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the host to the device. Data OUT transactions in isochronous transfers must be done using endpoints with ping-pong attributes.

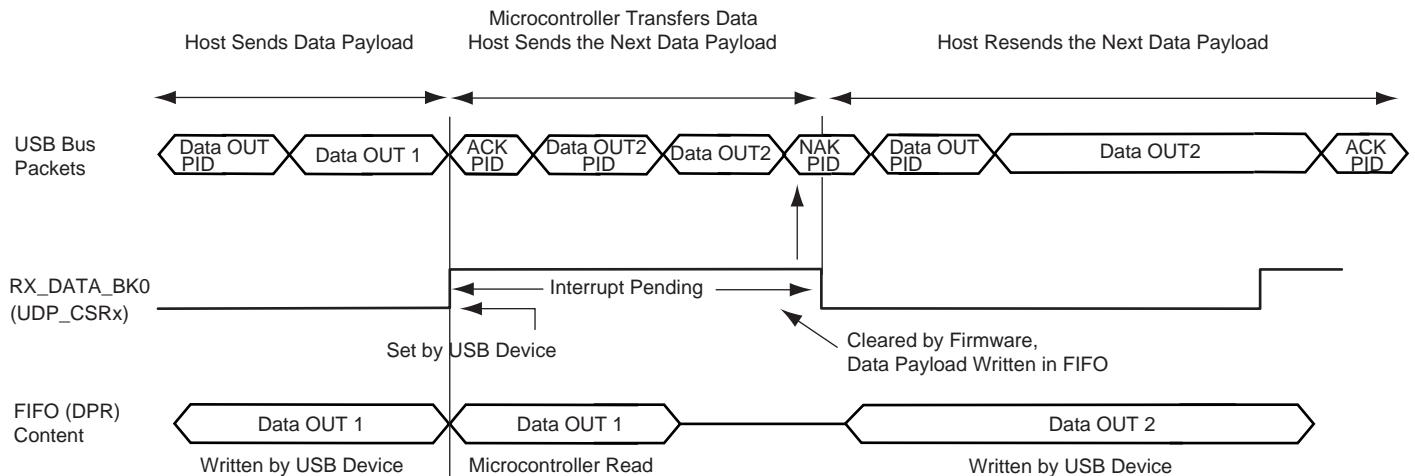
##### Data OUT Transaction Without Ping-pong Attributes

To perform a Data OUT transaction, using a non ping-pong endpoint:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. While the FIFO associated to this endpoint is being used by the microcontroller, a NAK PID is returned to the host. Once the FIFO is available, data are written to the FIFO by the USB device and an ACK is automatically carried out to the host.
3. The microcontroller is notified that the USB device has received a data payload polling RX\_DATA\_BK0 in the endpoint's UDP\_CSRx register. An interrupt is pending for this endpoint while RX\_DATA\_BK0 is set.

4. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP\_CSRx register.
5. The microcontroller carries out data received from the endpoint's memory to its memory. Data received is available by reading the endpoint's UDP\_FDRx register.
6. The microcontroller notifies the USB device that it has finished the transfer by clearing RX\_DATA\_BK0 in the endpoint's UDP\_CSRx register.
7. A new Data OUT packet can be accepted by the USB device.

**Figure 41-9. Data OUT Transfer for Non Ping-pong Endpoints**

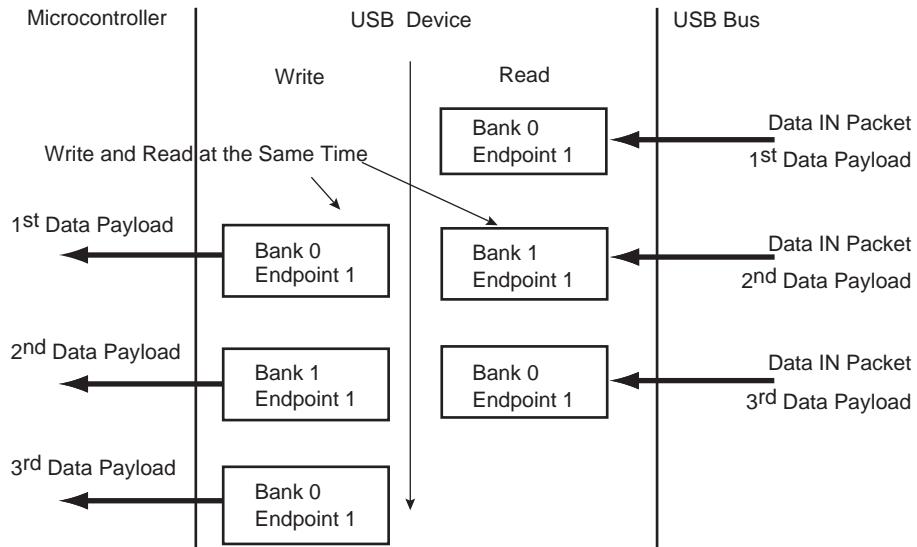


An interrupt is pending while the flag RX\_DATA\_BK0 is set. Memory transfer between the USB device, the FIFO and microcontroller memory can not be done after RX\_DATA\_BK0 has been cleared. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the current Data OUT packet in the FIFO.

#### Using Endpoints With Ping-pong Attributes

During isochronous transfer, using an endpoint with ping-pong attributes is obligatory. To be able to guarantee a constant bandwidth, the microcontroller must read the previous data payload sent by the host, while the current data payload is received by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

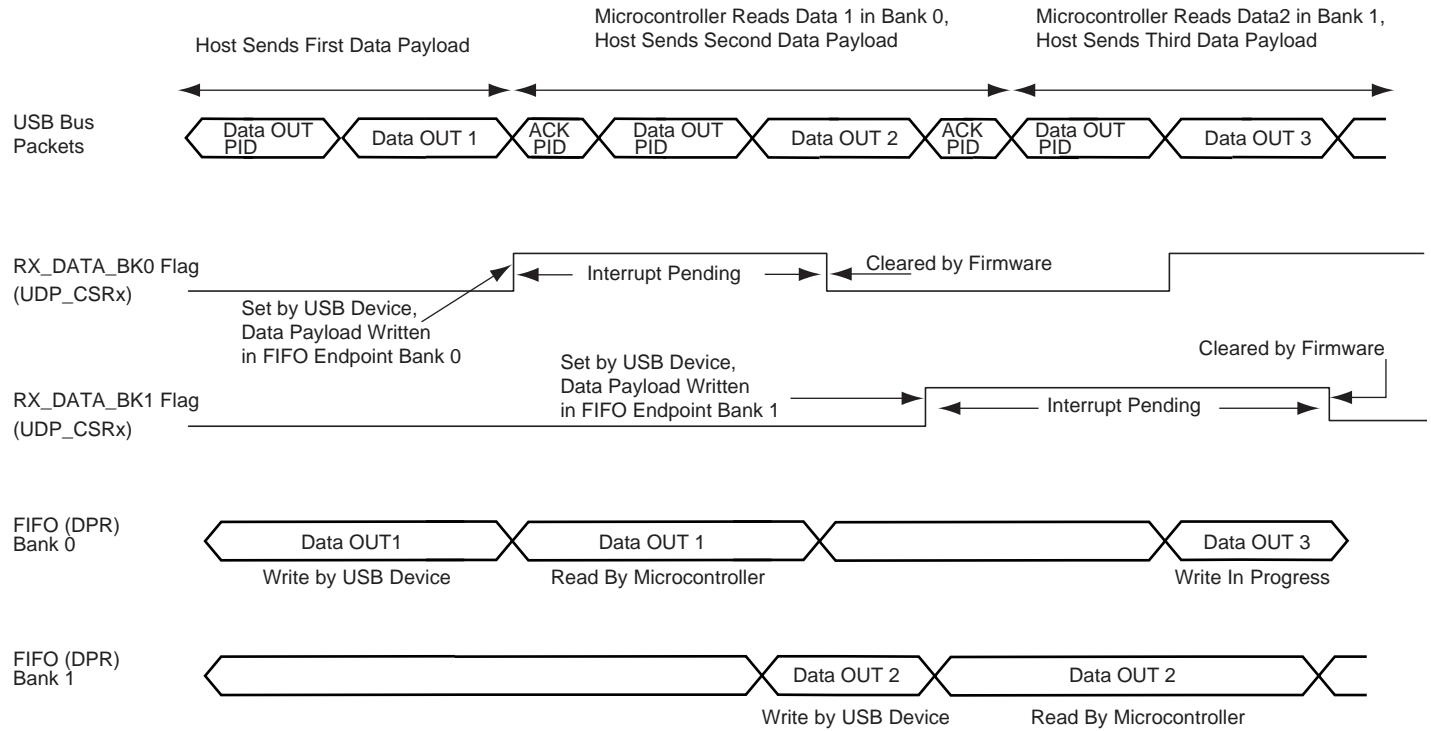
**Figure 41-10. Bank Swapping in Data OUT Transfers for Ping-pong Endpoints**



When using a ping-pong endpoint, the following procedures are required to perform Data OUT transactions:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. It is written in the endpoint's FIFO Bank 0.
3. The USB device sends an ACK PID packet to the host. The host can immediately send a second Data OUT packet. It is accepted by the device and copied to FIFO Bank 1.
4. The microcontroller is notified that the USB device has received a data payload, polling RX\_DATA\_BK0 in the endpoint's UDP\_CSRx register. An interrupt is pending for this endpoint while RX\_DATA\_BK0 is set.
5. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP\_CSRx register.
6. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is made available by reading the endpoint's UDP\_FDRx register.
7. The microcontroller notifies the USB peripheral device that it has finished the transfer by clearing RX\_DATA\_BK0 in the endpoint's UDP\_CSRx register.
8. A third Data OUT packet can be accepted by the USB peripheral device and copied in the FIFO Bank 0.
9. If a second Data OUT packet has been received, the microcontroller is notified by the flag RX\_DATA\_BK1 set in the endpoint's UDP\_CSRx register. An interrupt is pending for this endpoint while RX\_DATA\_BK1 is set.
10. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is available by reading the endpoint's UDP\_FDRx register.
11. The microcontroller notifies the USB device it has finished the transfer by clearing RX\_DATA\_BK1 in the endpoint's UDP\_CSRx register.
12. A fourth Data OUT packet can be accepted by the USB device and copied in the FIFO Bank 0.

**Figure 41-11. Data OUT Transfer for Ping-pong Endpoint**



Note: An interrupt is pending while the RX\_DATA\_BK0 or RX\_DATA\_BK1 flag is set.

**Warning:** When RX\_DATA\_BK0 and RX\_DATA\_BK1 are both set, there is no way to determine which one to clear first. Thus the software must keep an internal counter to be sure to clear alternatively RX\_DATA\_BK0 then RX\_DATA\_BK1. This situation may occur when the software application is busy elsewhere and the two banks are filled by the USB host. Once the application comes back to the USB driver, the two flags are set.

#### 41.6.2.4 Stall Handshake

A stall handshake can be used in one of two distinct occasions. (For more information on the stall handshake, refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0*.)

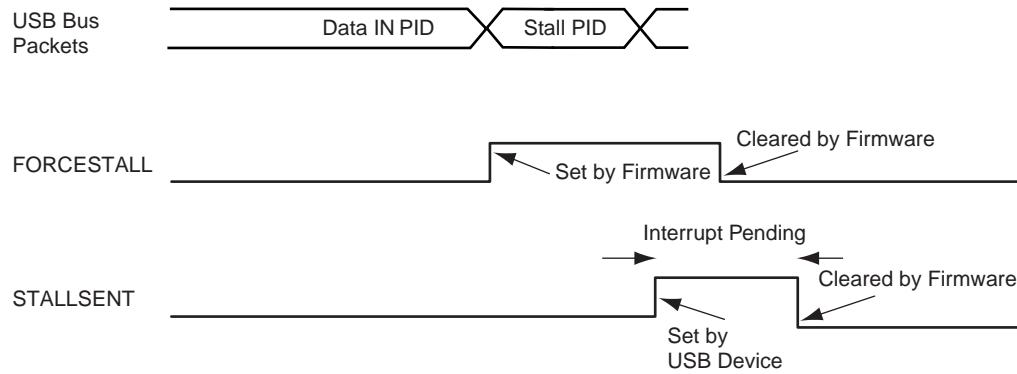
- A functional stall is used when the halt feature associated with the endpoint is set. (Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0*, for more information on the halt feature.)
- To abort the current request, a protocol stall is used, but uniquely with control transfer.

The following procedure generates a stall packet:

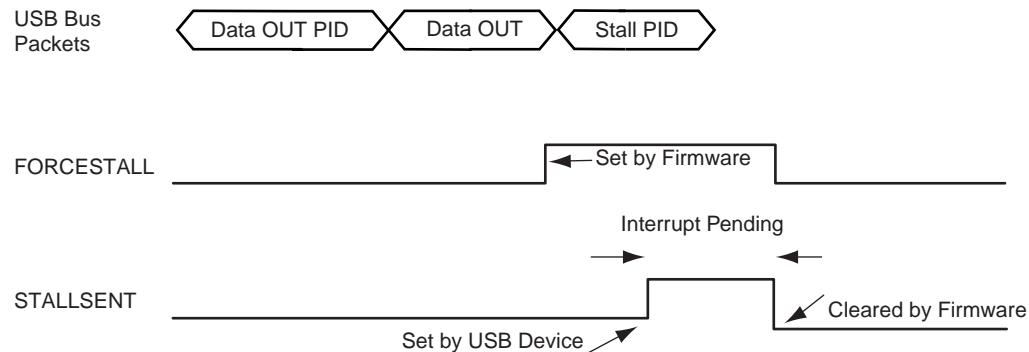
1. The microcontroller sets the FORCESTALL flag in the UDP\_CSRx endpoint's register.
2. The host receives the stall packet.
3. The microcontroller is notified that the device has sent the stall by polling the STALLSENT to be set. An endpoint interrupt is pending while STALLSENT is set. The microcontroller must clear STALLSENT to clear the interrupt.

When a setup transaction is received after a stall handshake, STALLSENT must be cleared in order to prevent interrupts due to STALLSENT being set.

**Figure 41-12. Stall Handshake (Data IN Transfer)**



**Figure 41-13. Stall Handshake (Data OUT Transfer)**



#### 41.6.2.5 Transmit Data Cancellation

Some endpoints have dual-banks whereas some endpoints have only one bank. The procedure to cancel transmission data held in these banks is described below.

To see the organization of dual-bank availability refer to [Table 41-1 "USB Endpoint Description"](#).

##### Endpoints **Without** Dual-Banks

There are two possibilities: In one case, TXPKTRDY field in UDP\_CSR has already been set. In the other instance, TXPKTRDY is not set.

- TXPKTRDY is not set:
  - Reset the endpoint to clear the FIFO (pointers). (See, [Section 41.7.9 "UDP Reset Endpoint Register"](#).)
- TXPKTRDY has already been set:
  - Clear TXPKTRDY so that no packet is ready to be sent
  - Reset the endpoint to clear the FIFO (pointers). (See, [Section 41.7.9 "UDP Reset Endpoint Register"](#).)

##### Endpoints **With** Dual-Banks

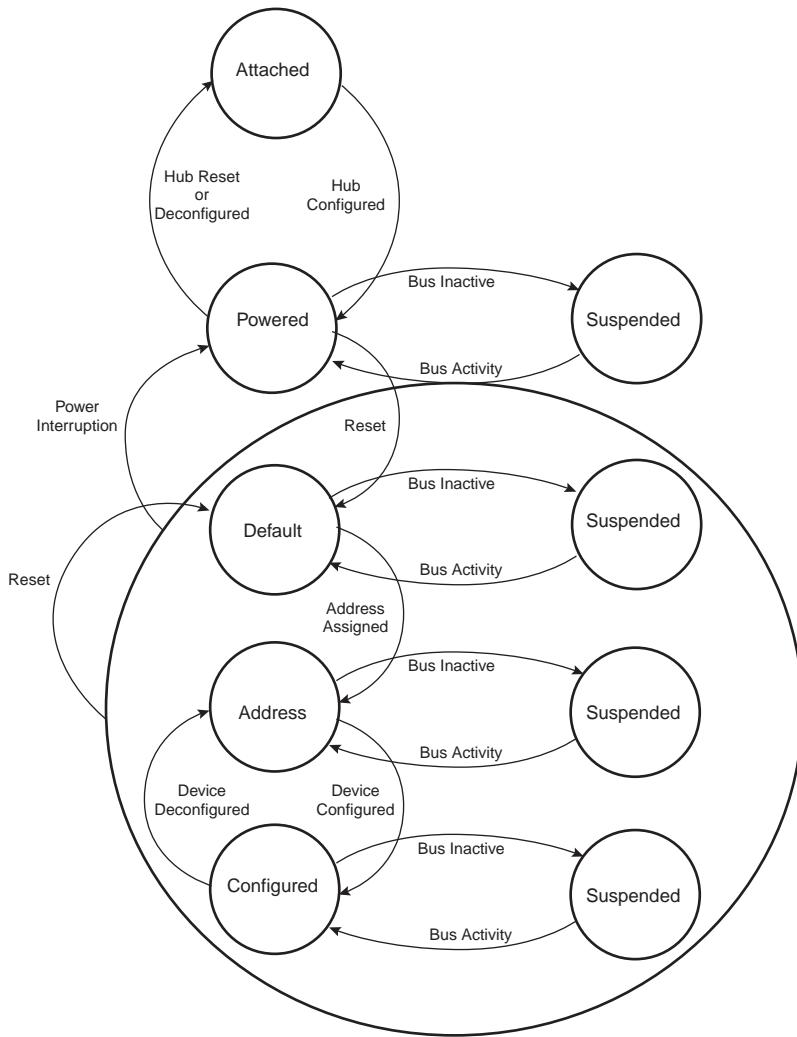
There are two possibilities: In one case, TXPKTRDY field in UDP\_CSR has already been set. In the other instance, TXPKTRDY is not set.

- TXPKTRDY is not set:
  - Reset the endpoint to clear the FIFO (pointers). (See, [Section 41.7.9 "UDP Reset Endpoint Register"](#).)
- TXPKTRDY has already been set:
  - Clear TXPKTRDY and read it back until actually read at 0.
  - Set TXPKTRDY and read it back until actually read at 1.
  - Clear TXPKTRDY so that no packet is ready to be sent.
  - Reset the endpoint to clear the FIFO (pointers). (See, [Section 41.7.9 "UDP Reset Endpoint Register"](#).)

#### 41.6.3 Controlling Device States

A USB device has several possible states. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0*.

Figure 41-14. USB Device State Diagram



Movement from one state to another depends on the USB bus state or on standard requests sent through control transactions via the default endpoint (endpoint 0).

After a period of bus inactivity, the USB device enters Suspend Mode. Accepting Suspend/Resume requests from the USB host is mandatory. Constraints in Suspend Mode are very strict for bus-powered applications; devices may not consume more than 500  $\mu$ A on the USB bus.

While in Suspend Mode, the host may wake up a device by sending a resume signal (bus activity) or a USB device may send a wake up request to the host, e.g., waking up a PC by moving a USB mouse.

The wake up feature is not mandatory for all devices and must be negotiated with the host.

##### 41.6.3.1 Not Powered State

Self powered devices can detect 5V VBUS using a PIO as described in the typical connection section. When the device is not connected to a host, device power consumption can be reduced by disabling MCK for the UDP, disabling UDPCK and disabling the transceiver. DDP and DDM lines are pulled down by 330 K $\Omega$  resistors.

#### 41.6.3.2 Entering Attached State

To enable integrated pull-up, the PUON bit in the UDP\_TXVC register must be set.

**Warning:** To write to the UDP\_TXVC register, MCK clock must be enabled on the UDP. This is done in the Power Management Controller.

After pull-up connection, the device enters the powered state. In this state, the UDPCK and MCK must be enabled in the Power Management Controller. The transceiver can remain disabled.

#### 41.6.3.3 From Powered State to Default State

After its connection to a USB host, the USB device waits for an end-of-bus reset. The unmaskable flag ENDBUSRES is set in the register UDP\_ISR and an interrupt is triggered.

Once the ENDBUSRES interrupt has been triggered, the device enters Default State. In this state, the UDP software must:

- Enable the default endpoint, setting the EPEDS flag in the UDP\_CSR[0] register and, optionally, enabling the interrupt for endpoint 0 by writing 1 to the UDP\_IER register. The enumeration then begins by a control transfer.
- Configure the interrupt mask register which has been reset by the USB reset detection
- Enable the transceiver clearing the TXVDIS flag in the UDP\_TXVC register.

In this state UDPCK and MCK must be enabled.

**Warning:** Each time an ENDBUSRES interrupt is triggered, the Interrupt Mask Register and UDP\_CSR registers have been reset.

#### 41.6.3.4 From Default State to Address State

After a set address standard device request, the USB host peripheral enters the address state.

**Warning:** Before the device enters in address state, it must achieve the Status IN transaction of the control transfer, i.e., the UDP device sets its new address once the TXCOMP flag in the UDP\_CSR[0] register has been received and cleared.

To move to address state, the driver software sets the FADDEN flag in the UDP\_GLB\_STAT register, sets its new address, and sets the FEN bit in the UDP\_FADDR register.

#### 41.6.3.5 From Address State to Configured State

Once a valid Set Configuration standard request has been received and acknowledged, the device enables endpoints corresponding to the current configuration. This is done by setting the EPEDS and EPTYPE fields in the UDP\_CSRx registers and, optionally, enabling corresponding interrupts in the UDP\_IER register.

#### 41.6.3.6 Entering in Suspend State

When a Suspend (no bus activity on the USB bus) is detected, the RXSUSP signal in the UDP\_ISR register is set. This triggers an interrupt if the corresponding bit is set in the UDP\_IMR register. This flag is cleared by writing to the UDP\_ICR register. Then the device enters Suspend Mode.

In this state bus powered devices must drain less than 500uA from the 5V VBUS. As an example, the microcontroller switches to slow clock, disables the PLL and main oscillator, and goes into Idle Mode. It may also switch off other devices on the board.

The USB device peripheral clocks can be switched off. Resume event is asynchronously detected. MCK and UDPCK can be switched off in the Power Management controller and the USB transceiver can be disabled by setting the TXVDIS field in the UDP\_TXVC register.

**Warning:** Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. Switching off MCK for the UDP peripheral must be one of the last operations after writing to the UDP\_TXVC and acknowledging the RXSUSP.

#### 41.6.3.7 Receiving a Host Resume

In suspend mode, a resume event on the USB bus line is detected asynchronously, transceiver and clocks are disabled (however the pull-up shall not be removed).

Once the resume is detected on the bus, the WAKEUP signal in the UDP\_ISR is set. It may generate an interrupt if the corresponding bit in the UDP\_IMR register is set. This interrupt may be used to wake up the core, enable PLL and main oscillators and configure clocks.

**Warning:** Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. MCK for the UDP must be enabled before clearing the WAKEUP bit in the UDP\_ICR register and clearing TXVDIS in the UDP\_TXVC register.

#### 41.6.3.8 Sending a Device Remote Wake-up

In Suspend state it is possible to wake up the host sending an external resume.

- The device must wait at least 5 ms after being entered in suspend before sending an external resume.
- The device has 10 ms from the moment it starts to drain current and it forces a K state to resume the host.
- The device must force a K state from 1 to 15 ms to resume the host

Before sending a K state to the host, MCK, UDPCK and the transceiver must be enabled. Then to enable the remote wake-up feature, the RMWUPE bit in the UDP\_GLB\_STAT register must be enabled. To force the K state on the line, a transition of the ESR bit from 0 to 1 has to be done in the UDP\_GLB\_STAT register. This transition must be accomplished by first writing a 0 in the ESR bit and then writing a 1.

The K state is automatically generated and released according to the USB 2.0 specification.

## 41.7 USB Device Port (UDP) User Interface

**WARNING:** The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers, including the UDP\_TXVC register.

**Table 41-6. Register Mapping**

Offset	Register	Name	Access	Reset
0x000	Frame Number Register	UDP_FRM_NUM	Read-only	0x0000_0000
0x004	Global State Register	UDP_GLB_STAT	Read-write	0x0000_0010
0x008	Function Address Register	UDP_FADDR	Read-write	0x0000_0100
0x00C	Reserved	—	—	—
0x010	Interrupt Enable Register	UDP_IER	Write-only	
0x014	Interrupt Disable Register	UDP_IDR	Write-only	
0x018	Interrupt Mask Register	UDP_IMR	Read-only	0x0000_1200
0x01C	Interrupt Status Register	UDP_ISR	Read-only	— <sup>(1)</sup>
0x020	Interrupt Clear Register	UDP_ICR	Write-only	
0x024	Reserved	—	—	—
0x028	Reset Endpoint Register	UDP_RST_EP	Read-write	0x0000_0000
0x02C	Reserved	—	—	—
0x030	Endpoint Control and Status Register 0	UDP_CSR0	Read-write	0x0000_0000
...	...	...	...	...
0x030 + 0x4 * 7	Endpoint Control and Status Register 7	UDP_CSR7	Read-write	0x0000_0000
0x050	Endpoint FIFO Data Register 0	UDP_FDR0	Read-write	0x0000_0000
...	...	...	...	...
0x050 + 0x4 * 7	Endpoint FIFO Data Register 7	UDP_FDR7	Read-write	0x0000_0000
0x070	Reserved	—	—	—
0x074	Transceiver Control Register	UDP_TXVC <sup>(2)</sup>	Read-write	0x0000_0100
0x078 - 0xFC	Reserved	—	—	—

Notes: 1. Reset values are not defined for UDP\_ISR.

2. See Warning above the "Register Mapping" on this page.

#### 41.7.1 UDP Frame Number Register

**Name:** UDP\_FRM\_NUM

**Address:** 0x40084000

**Access:** Read-only

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

- **FRM\_NUM[10:0]: Frame Number as Defined in the Packet Field Formats**

This 11-bit value is incremented by the host on a per frame basis. This value is updated at each start of frame.

Value Updated at the SOF\_EOP (Start of Frame End of Packet).

- **FRM\_ERR: Frame Error**

This bit is set at SOF\_EOP when the SOF packet is received containing an error.

This bit is reset upon receipt of SOF\_PID.

- **FRM\_OK: Frame OK**

This bit is set at SOF\_EOP when the SOF packet is received without any error.

This bit is reset upon receipt of SOF\_PID (Packet Identification).

In the Interrupt Status Register, the SOF interrupt is updated upon receiving SOF\_PID. This bit is set without waiting for EOP.

Note: In the 8-bit Register Interface, FRM\_OK is bit 4 of FRM\_NUM\_H and FRM\_ERR is bit 3 of FRM\_NUM\_L.

#### 41.7.2 UDP Global State Register

**Name:** UDP\_GLB\_STAT

**Address:** 0x40084004

**Access:** Read-write

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
—	—	—	RMWUPE	RSMINPR	ESR	CONFG	FADDEN

This register is used to get and set the device state as specified in Chapter 9 of the *USB Serial Bus Specification, Rev.2.0*.

- **FADDEN: Function Address Enable**

Read:

0 = Device is not in address state.

1 = Device is in address state.

Write:

0 = No effect, only a reset can bring back a device to the default state.

1 = Sets device in address state. This occurs after a successful Set Address request. Beforehand, the UDP\_FADDR register must have been initialized with Set Address parameters. Set Address must complete the Status Stage before setting FADDEN. Refer to chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **CONFG: Configured**

Read:

0 = Device is not in configured state.

1 = Device is in configured state.

Write:

0 = Sets device in a non configured state

1 = Sets device in configured state.

The device is set in configured state when it is in address state and receives a successful Set Configuration request. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **ESR: Enable Send Resume**

0 = Mandatory value prior to starting any Remote Wake Up procedure.

1 = Starts the Remote Wake Up procedure if this bit value was 0 and if RMWUPE is enabled.

- **RMWUPE: Remote Wake-up Enable**

0 = The Remote Wake Up feature of the device is disabled.

1 = The Remote Wake Up feature of the device is enabled.

#### 41.7.3 UDP Function Address Register

**Name:** UDP\_FADDR

**Address:** 0x40084008

**Access:** Read-write

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

- **FADD[6:0]: Function Address Value**

The Function Address Value must be programmed by firmware once the device receives a set address request from the host, and has achieved the status stage of the no-data control sequence. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information. After power up or reset, the function address value is set to 0.

- **FEN: Function Enable**

Read:

0 = Function endpoint disabled.

1 = Function endpoint enabled.

Write:

0 = Disables function endpoint.

1 = Default value.

The Function Enable bit (FEN) allows the microcontroller to enable or disable the function endpoints. The microcontroller sets this bit after receipt of a reset from the host. Once this bit is set, the USB device is able to accept and transfer data packets from and to the host.

#### 41.7.4 UDP Interrupt Enable Register

**Name:** UDP\_IER

**Address:** 0x40084010

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Enable Endpoint 0 Interrupt**

- **EP1INT: Enable Endpoint 1 Interrupt**

- **EP2INT: Enable Endpoint 2 Interrupt**

- **EP3INT: Enable Endpoint 3 Interrupt**

- **EP4INT: Enable Endpoint 4 Interrupt**

- **EP5INT: Enable Endpoint 5 Interrupt**

- **EP6INT: Enable Endpoint 6 Interrupt**

- **EP7INT: Enable Endpoint 7 Interrupt**

0 = No effect.

1 = Enables corresponding Endpoint Interrupt.

- **RXSUSP: Enable UDP Suspend Interrupt**

0 = No effect.

1 = Enables UDP Suspend Interrupt.

- **RXRSM: Enable UDP Resume Interrupt**

0 = No effect.

1 = Enables UDP Resume Interrupt.

- **SOFINT: Enable Start Of Frame Interrupt**

0 = No effect.

1 = Enables Start Of Frame Interrupt.

- **WAKEUP: Enable UDP bus Wake-up Interrupt**

0 = No effect.

1 = Enables USB bus Interrupt.

#### 41.7.5 UDP Interrupt Disable Register

**Name:** UDP\_IDR

**Address:** 0x40084014

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Disable Endpoint 0 Interrupt**

- **EP1INT: Disable Endpoint 1 Interrupt**

- **EP2INT: Disable Endpoint 2 Interrupt**

- **EP3INT: Disable Endpoint 3 Interrupt**

- **EP4INT: Disable Endpoint 4 Interrupt**

- **EP5INT: Disable Endpoint 5 Interrupt**

- **EP6INT: Disable Endpoint 6 Interrupt**

- **EP7INT: Disable Endpoint 7 Interrupt**

0 = No effect.

1 = Disables corresponding Endpoint Interrupt.

- **RXSUSP: Disable UDP Suspend Interrupt**

0 = No effect.

1 = Disables UDP Suspend Interrupt.

- **RXRSM: Disable UDP Resume Interrupt**

0 = No effect.

1 = Disables UDP Resume Interrupt.

- **SOFINT: Disable Start Of Frame Interrupt**

0 = No effect.

1 = Disables Start Of Frame Interrupt

- **WAKEUP: Disable USB Bus Interrupt**

0 = No effect.

1 = Disables USB Bus Wake-up Interrupt.

#### 41.7.6 UDP Interrupt Mask Register

**Name:** UDP\_IMR

**Address:** 0x40084018

**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	BIT12	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Mask Endpoint 0 Interrupt**

- **EP1INT: Mask Endpoint 1 Interrupt**

- **EP2INT: Mask Endpoint 2 Interrupt**

- **EP3INT: Mask Endpoint 3 Interrupt**

- **EP4INT: Mask Endpoint 4 Interrupt**

- **EP5INT: Mask Endpoint 5 Interrupt**

- **EP6INT: Mask Endpoint 6 Interrupt**

- **EP7INT: Mask Endpoint 7 Interrupt**

0 = Corresponding Endpoint Interrupt is disabled.

1 = Corresponding Endpoint Interrupt is enabled.

- **RXSUSP: Mask UDP Suspend Interrupt**

0 = UDP Suspend Interrupt is disabled.

1 = UDP Suspend Interrupt is enabled.

- **RXRSM: Mask UDP Resume Interrupt.**

0 = UDP Resume Interrupt is disabled.

1 = UDP Resume Interrupt is enabled.

- **SOFINT: Mask Start Of Frame Interrupt**

0 = Start of Frame Interrupt is disabled.

1 = Start of Frame Interrupt is enabled.

- **BIT12: UDP\_IMR Bit 12**

Bit 12 of UDP\_IMR cannot be masked and is always read at 1.

- **WAKEUP: USB Bus WAKEUP Interrupt**

0 = USB Bus Wake-up Interrupt is disabled.

1 = USB Bus Wake-up Interrupt is enabled.

Note: When the USB block is in suspend mode, the application may power down the USB logic. In this case, any USB HOST resume request that is made must be taken into account and, thus, the reset value of the RXRSM bit of the register UDP\_IMR is enabled.

#### 41.7.7 UDP Interrupt Status Register

**Name:** UDP\_ISR

**Address:** 0x4008401C

**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	ENDBUSRES	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Endpoint 0 Interrupt Status**

- **EP1INT: Endpoint 1 Interrupt Status**

- **EP2INT: Endpoint 2 Interrupt Status**

- **EP3INT: Endpoint 3 Interrupt Status**

- **EP4INT: Endpoint 4 Interrupt Status**

- **EP5INT: Endpoint 5 Interrupt Status**

- **EP6INT: Endpoint 6 Interrupt Status**

- **EP7INT: Endpoint 7 Interrupt Status**

0 = No Endpoint0 Interrupt pending.

1 = Endpoint0 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading UDP\_CSR0:

RXSETUP set to 1

RX\_DATA\_BK0 set to 1

RX\_DATA\_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP0INT is a sticky bit. Interrupt remains valid until EP0INT is cleared by writing in the corresponding UDP\_CSR0 bit.

- **RXSUSP: UDP Suspend Interrupt Status**

0 = No UDP Suspend Interrupt pending.

1 = UDP Suspend Interrupt has been raised.

The USB device sets this bit when it detects no activity for 3ms. The USB device enters Suspend mode.

- **RXRSM: UDP Resume Interrupt Status**

0 = No UDP Resume Interrupt pending.

1 = UDP Resume Interrupt has been raised.

The USB device sets this bit when a UDP resume signal is detected at its port.

After reset, the state of this bit is undefined, the application must clear this bit by setting the RXRSM flag in the UDP\_ICR register.

- **SOFINT: Start of Frame Interrupt Status**

0 = No Start of Frame Interrupt pending.

1 = Start of Frame Interrupt has been raised.

This interrupt is raised each time a SOF token has been detected. It can be used as a synchronization signal by using isochronous endpoints.

- **ENDBUSRES: End of BUS Reset Interrupt Status**

0 = No End of Bus Reset Interrupt pending.

1 = End of Bus Reset Interrupt has been raised.

This interrupt is raised at the end of a UDP reset sequence. The USB device must prepare to receive requests on the endpoint 0. The host starts the enumeration, then performs the configuration.

- **WAKEUP: UDP Resume Interrupt Status**

0 = No Wake-up Interrupt pending.

1 = A Wake-up Interrupt (USB Host Sent a RESUME or RESET) occurred since the last clear.

After reset the state of this bit is undefined, the application must clear this bit by setting the WAKEUP flag in the UDP\_ICR register.

#### 41.7.8 UDP Interrupt Clear Register

**Name:** UDP\_ICR

**Address:** 0x40084020

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	ENDBUSRES	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **RXSUSP: Clear UDP Suspend Interrupt**

0 = No effect.

1 = Clears UDP Suspend Interrupt.

- **RXRSM: Clear UDP Resume Interrupt**

0 = No effect.

1 = Clears UDP Resume Interrupt.

- **SOFINT: Clear Start Of Frame Interrupt**

0 = No effect.

1 = Clears Start Of Frame Interrupt.

- **ENDBUSRES: Clear End of Bus Reset Interrupt**

0 = No effect.

1 = Clears End of Bus Reset Interrupt.

- **WAKEUP: Clear Wake-up Interrupt**

0 = No effect.

1 = Clears Wake-up Interrupt.

#### 41.7.9 UDP Reset Endpoint Register

**Name:** UDP\_RST\_EP

**Address:** 0x40084028

**Access:** Read-write

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
EP7	EP6	EP5	EP4	EP3	EP2	EP1	EP0

- **EP0: Reset Endpoint 0**
- **EP1: Reset Endpoint 1**
- **EP2: Reset Endpoint 2**
- **EP3: Reset Endpoint 3**
- **EP4: Reset Endpoint 4**
- **EP5: Reset Endpoint 5**
- **EP6: Reset Endpoint 6**
- **EP7: Reset Endpoint 7**

This flag is used to reset the FIFO associated with the endpoint and the bit RXBYTECOUNT in the register UDP\_CSRx. It also resets the data toggle to DATA0. It is useful after removing a HALT condition on a BULK endpoint. Refer to Chapter 5.8.5 in the *USB Serial Bus Specification, Rev.2.0*.

**Warning:** This flag must be cleared at the end of the reset. It does not clear UDP\_CSRx flags.

0 = No reset.

1 = Forces the corresponding endpoint FIFO pointers to 0, therefore RXBYTECNT field is read at 0 in UDP\_CSRx register.

Resetting the endpoint is a two-step operation:

1. Set the corresponding EPx field.
2. Clear the corresponding EPx field.

#### 41.7.10 UDP Endpoint Control and Status Register (Control, Bulk Interrupt Endpoints)

**Name:** UDP\_CSRx [x ..7]

**Address:** 0x40084030

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	RXBYTECNT		
23	22	21	20	19	18	17	16
RXBYTECNT							
15	14	13	12	11	10	9	8
EPEDS	-	-	-	DTGLE	EPTYPE		
7	6	5	4	3	2	1	0
DIR	RX_DATA_BK1	FORCESTALL	TXPKTRDY	STALLSENT	RXSETUP	RX_DATA_BK0	TXCOMP

**WARNING:** Due to synchronization between MCK and UDPCK, the software application must wait for the end of the write operation before executing another write by polling the bits which must be set/cleared.

```

#if defined ( __ICCARM__ )
#define nop() ( __no_operation() )

#elif defined ( __GNUC__ )
#define nop() __asm__ __volatile__ ( "nop" )

#endif

/// Bitmap for all status bits in CSR that are not effected by a value 1.
#define REG_NO_EFFECT_1_ALL    AT91C_UDP_RX_DATA_BK0 \
                           | AT91C_UDP_RX_DATA_BK1 \
                           | AT91C_UDP_STALLSENT \
                           | AT91C_UDP_RXSETUP \
                           | AT91C_UDP_TXCOMP

/// Sets the specified bit(s) in the UDP_CSR register.
/// \param endpoint The endpoint number of the CSR to process.
/// \param flags The bitmap to set to 1.
#define SET_CSR(endpoint, flags) \
{ \
    volatile unsigned int reg; \
    reg = AT91C_BASE_UDP->UDP_CSR[endpoint] ; \
    reg |= REG_NO_EFFECT_1_ALL; \
    reg |= (flags); \
    AT91C_BASE_UDP->UDP_CSR[endpoint] = reg; \
    for( nop_count=0; nop_count<15; nop_count++ ) { \
        nop(); \
    } \
}

```

```

/// Clears the specified bit(s) in the UDP_CSR register.
/// \param endpoint The endpoint number of the CSR to process.
/// \param flags The bitmap to clear to 0.
#define CLEAR_CSR(endpoint, flags) \
{ \
    volatile unsigned int reg; \
    reg = AT91C_BASE_UDP->UDP_CSR[endpoint]; \
    reg |= REG_NO_EFFECT_1_ALL; \
    reg &= ~(flags); \
    AT91C_BASE_UDP->UDP_CSR[endpoint] = reg; \
    for( nop_count=0; nop_count<15; nop_count++ ) { \
        nop(); \
    } \
}

```

In a preemptive environment, set or clear the flag and wait for a time of 1 UDPCK clock cycle and 1 peripheral clock cycle. However, RX\_DATA\_BK0, TXPKTRDY, RX\_DATA\_BK1 require wait times of 3 UDPCK clock cycles and 5 peripheral clock cycles before accessing DPR.

- TXCOMP: Generates an IN Packet with Data Previously Written in the DPR**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Clear the flag, clear the interrupt.

1 = No effect.

Read (Set by the USB peripheral):

0 = Data IN transaction has not been acknowledged by the Host.

1 = Data IN transaction is achieved, acknowledged by the Host.

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- RX\_DATA\_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Notify USB peripheral device that data have been read in the FIFO's Bank 0.

1 = To leave the read value unchanged.

Read (Set by the USB peripheral):

0 = No data packet has been received in the FIFO's Bank 0.

1 = A data packet has been received, it has been stored in the FIFO's Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCENT field. Bank 0 FIFO values are read through the UDP\_FDRx register. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX\_DATA\_BK0.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- RXSETUP: Received Setup**

This flag generates an interrupt while it is set to one.

Read:

0 = No setup packet available.

1 = A setup data packet has been sent by the host and is available in the FIFO.

Write:

0 = Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1 = No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the UDP\_FDRx register to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transaction is not accepted while RXSETUP is set.

- **STALLSENT: Stall Sent**

This flag generates an interrupt while it is set to one.

This ends a STALL handshake.

Read:

0 = The host has not acknowledged a STALL.

1 = Host has acknowledged the stall.

Write:

0 = Resets the STALLSENT flag, clears the interrupt.

1 = No effect.

This is mandatory for the device firmware to clear this flag. Otherwise the interrupt remains.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

- **TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read:

0 = There is no data to send.

1 = The data is waiting to be sent upon reception of token IN.

Write:

0 = Can be used in the procedure to cancel transmission data. (See, [Section 41.6.2.5 "Transmit Data Cancellation" on page 1100](#))

1 = A new data payload has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the UDP\_FDRx register. Once the data payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0 = Normal state.

1 = Stall state.

Write:

0 = Return to normal state.

1 = Send STALL to the host.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX\_DATA\_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Notifies USB device that data have been read in the FIFO's Bank 1.

1 = To leave the read value unchanged.

Read (Set by the USB peripheral):

0 = No data packet has been received in the FIFO's Bank 1.

1 = A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP\_FDRx register. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX\_DATA\_BK1.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **DIR: Transfer Direction (only available for control endpoints)**

Read-write

0 = Allows Data OUT transactions in the control data stage.

1 = Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP\_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE[2:0]: Endpoint Type**

Read-Write

Value	Name	Description
000	CTRL	Control
001	ISO_OUT	Isochronous OUT
101	ISO_IN	Isochronous IN
010	BULK_OUT	Bulk OUT

Value	Name	Description
110	BULK_IN	Bulk IN
011	INT_OUT	Interrupt OUT
111	INT_IN	Interrupt IN

- **DTGLE: Data Toggle**

Read-only

0 = Identifies DATA0 packet.

1 = Identifies DATA1 packet.

Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read:

0 = Endpoint disabled.

1 = Endpoint enabled.

Write:

0 = Disables endpoint.

1 = Enables endpoint.

Control endpoints are always enabled. Reading or writing this field has no effect on control endpoints.

**Note:** After reset, all endpoints are configured as control endpoints (zero).

- **RXBYTECNT[10:0]: Number of Bytes Available in the FIFO**

Read-only

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the UDP\_FDRx register.

#### 41.7.11 UDP Endpoint Control and Status Register (Isochronous Endpoints)

**Name:** UDP\_CSRx [x = 0..7] (ISOENDPT)

**Address:** 0x40084030

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	RXBYTCNT		
23	22	21	20	19	18	17	16
RXBYTCNT							
15	14	13	12	11	10	9	8
EPEDS	–	–	–	DTGLE	EPTYPE		
7	6	5	4	3	2	1	0
DIR	RX_DATA_BK1	FORCESTALL	TXPKTRDY	ISOERROR	RXSETUP	RX_DATA_BK0	TXCOMP

- **TXCOMP: Generates an IN Packet with Data Previously Written in the DPR**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Clear the flag, clear the interrupt.

1 = No effect.

Read (Set by the USB peripheral):

0 = Data IN transaction has not been acknowledged by the Host.

1 = Data IN transaction is achieved, acknowledged by the Host.

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- **RX\_DATA\_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Notify USB peripheral device that data have been read in the FIFO's Bank 0.

1 = To leave the read value unchanged.

Read (Set by the USB peripheral):

0 = No data packet has been received in the FIFO's Bank 0.

1 = A data packet has been received, it has been stored in the FIFO's Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCNT field. Bank 0 FIFO values are read through the UDP\_FDRx register. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX\_DATA\_BK0.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **RXSETUP: Received Setup**

This flag generates an interrupt while it is set to one.

Read:

0 = No setup packet available.

1 = A setup data packet has been sent by the host and is available in the FIFO.

Write:

0 = Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1 = No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the UDP\_FDRx register to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transaction is not accepted while RXSETUP is set.

- ISOERROR: A CRC error has been detected in an isochronous transfer**

This flag generates an interrupt while it is set to one.

Read:

0 = No error in the previous isochronous transfer.

1 = CRC error has been detected, data available in the FIFO are corrupted.

Write:

0 = Resets the ISOERROR flag, clears the interrupt.

1 = No effect.

- TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read:

0 = There is no data to send.

1 = The data is waiting to be sent upon reception of token IN.

Write:

0 = Can be used in the procedure to cancel transmission data. (See, [Section 41.6.2.5 “Transmit Data Cancellation” on page 1100](#))

1 = A new data payload has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the UDP\_FDRx register. Once the data payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0 = Normal state.

1 = Stall state.

Write:

0 = Return to normal state.

1 = Send STALL to the host.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX\_DATA\_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Notifies USB device that data have been read in the FIFO's Bank 1.

1 = To leave the read value unchanged.

Read (Set by the USB peripheral):

0 = No data packet has been received in the FIFO's Bank 1.

1 = A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP\_FDRx register. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX\_DATA\_BK1.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **DIR: Transfer Direction (only available for control endpoints)**

Read-write

0 = Allows Data OUT transactions in the control data stage.

1 = Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP\_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE[2:0]: Endpoint Type**

Read-Write

Value	Name	Description
000	CTRL	Control
001	ISO_OUT	Isochronous OUT
101	ISO_IN	Isochronous IN
010	BULK_OUT	Bulk OUT
110	BULK_IN	Bulk IN
011	INT_OUT	Interrupt OUT
111	INT_IN	Interrupt IN

- **DTGLE: Data Toggle**

Read-only

0 = Identifies DATA0 packet.

1 = Identifies DATA1 packet.

Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read:

0 = Endpoint disabled.

1 = Endpoint enabled.

Write:

0 = Disables endpoint.

1 = Enables endpoint.

Control endpoints are always enabled. Reading or writing this field has no effect on control endpoints.

**Note:** After reset, all endpoints are configured as control endpoints (zero).

- **RXBYTECNT[10:0]: Number of Bytes Available in the FIFO**

Read-only

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the UDP\_FDRx register.

#### 41.7.12 UDP FIFO Data Register

**Name:** UDP\_FDRx [x = 0..7]

**Address:** 0x40084050

**Access:** Read-write

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
FIFO_DATA							

- **FIFO\_DATA[7:0]: FIFO Data Value**

The microcontroller can push or pop values in the FIFO through this register.

RXBYTECNT in the corresponding UDP\_CSRx register is the number of bytes to be read from the FIFO (sent by the host).

The maximum number of bytes to write is fixed by the Max Packet Size in the Standard Endpoint Descriptor. It can not be more than the physical memory size associated to the endpoint. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information.

#### 41.7.13 UDP Transceiver Control Register

**Name:** UDP\_TXVC

**Address:** 0x40084074

**Access:** Read-write

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

**WARNING:** The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP\_TXVC register.

- **TXVDIS: Transceiver Disable**

When UDP is disabled, power consumption can be reduced significantly by disabling the embedded transceiver. This can be done by setting TXVDIS field.

To enable the transceiver, TXVDIS must be cleared.

- **PUON: Pull-up On**

0: The 1.5KΩ integrated pull-up on DDP is disconnected.

1: The 1.5 KΩ integrated pull-up on DDP is connected.

**NOTE:** If the USB pull-up is not connected on DDP, the user should not write in any UDP register other than the UDP\_TXVC register. This is because if DDP and DDM are floating at 0, or pulled down, then SE0 is received by the device with the consequence of a USB Reset.

## 42. Analog Comparator Controller (ACC)

### 42.1 Description

The Analog Comparator Controller (ACC) configures the Analog Comparator and generates an interrupt according to the user settings. The analog comparator embeds 8 to 1 multiplexers on both inputs.

The Analog Comparator compares two voltages and the result of this comparison gives a compare output. The user can select a high-speed or low-power option. Additionally, the hysteresis level, edge detection and polarity are configurable.

The ACC can also generate a compare event which can be used by the PWM.

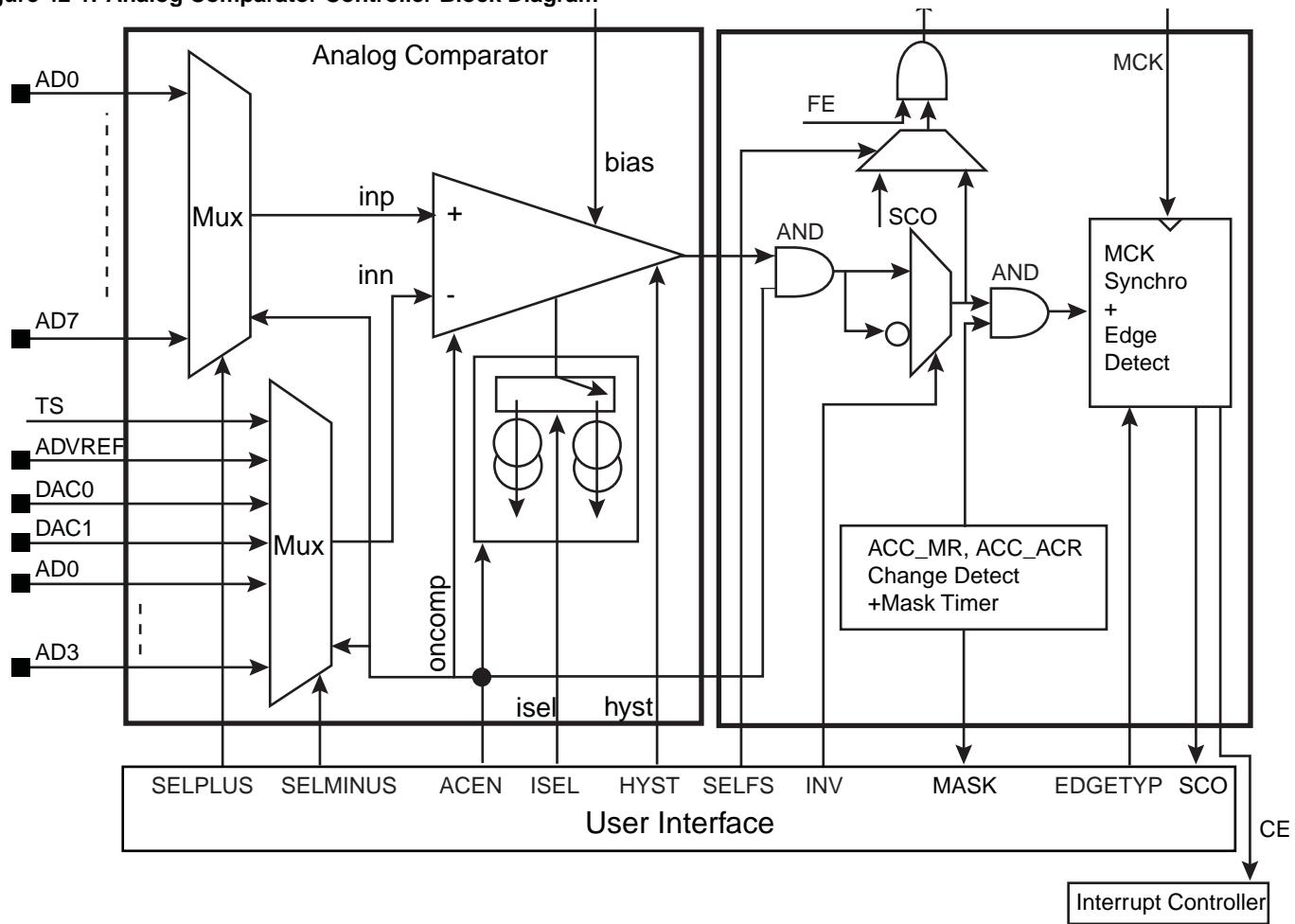
Please refer to [Figure 42-1 on page 1128](#) for detailed schematics.

### 42.2 Embedded characteristics

- 8 User Analog Inputs Selectable for Comparison
- 4 Voltage References Selectable for Comparison: Temperature Sensor, ADVREF, DAC0 and DAC1
- Interrupt Generation
- Compare Event Fault Generation for PWM

## 42.3 Block Diagram

Figure 42-1. Analog Comparator Controller Block Diagram



## 42.4 Pin Name List

Table 42-1. ACC Pin List

Pin Name	Description	Type
AD0..AD7	Analog Inputs	Input
TS	On-Chip Temperature Sensor	Input
ADVREF	ADC Voltage Reference.	Input
DAC0, DAC1	On-Chip DAC Outputs	Input
FAULT	Drives internal fault input of PWM	Output

## 42.5 Product Dependencies

### 42.5.1 I/O Lines

The analog input pins (AD0-AD7 and DAC0-1) are multiplexed with PIO lines. In this case, the assignment of the ACC inputs is automatically done as soon as the corresponding input is enabled by writing the ACC Mode register (SELMINUS and SELPLUS).

Table 42-2. Analog Comparator Inputs

Multiplexer Plus input	Multiplexer Minus input
AFE0_ADO	Temperature sensor
AFE0_AD1	ADVREF
AFE0_AD2	DAC0
AFE0_AD3	DAC1
AFE0_AD4	AFE0_ADO
AFE0_AD5	AFE0_AD1
AFE1_AD0	AFE0_AD2
AFE1_AD1	AFE0_AD3

### 42.5.2 Power Management

The ACC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the Analog Comparator Controller clock.

Note that the voltage regulator needs to be activated to use the Analog Comparator.

### 42.5.3 Interrupt

The ACC has an interrupt line connected to the Interrupt Controller (IC). Handling the ACC interrupt requires programming the Interrupt Controller before configuring the ACC.

Table 42-3. Peripheral IDs

Instance	ID
ACC	33

### 42.5.4 Fault Output

The ACC has the FAULT output connected to the FAULT input of PWM. Please refer to chapter [Section 42.6.5 "Fault Mode"](#) and implementation of the PWM in the product.

## 42.6 Functional Description

### 42.6.1 ACC Description

The Analog Comparator Controller mainly controls the analog comparator settings. There is also post processing of the analog comparator output.

The output of the analog comparator is masked for the time the output may be invalid. This situation is encountered as soon as the analog comparator settings are modified.

A comparison flag is triggered by an event on the output of the analog comparator and an interrupt can be generated accordingly. The event on the analog comparator output can be selected among falling edge, rising edge or any edge.

The registers for programming are listed in [Table 42-4 on page 1131](#).

### 42.6.2 Analog Settings

The user can select the input hysteresis and configure high-speed or low-speed options.

- Shortest propagation delay/highest current consumption
- Longest propagation delay/lowest current consumption

### 42.6.3 Write Protection System

In order to provide security to the Analog Comparator Controller, a write protection system has been implemented.

The write protection mode prevents writing [ACC Mode Register](#) and [ACC Analog Control Register](#). When this mode is enabled and one of the protected registers is written, the register write request is canceled.

Due to the nature of the write protection feature, enabling and disabling the write protection mode requires a security code. Thus when enabling or disabling the write protection mode, the WPKEY field of the ACC\_WPMR register must be filled with the "ACC" ASCII code (corresponding to 0x414343), otherwise the register write will be canceled.

### 42.6.4 Automatic Output Masking Period

As soon as the analog comparator settings change, the output is invalid for a duration depending on ISEL current.

A masking period is automatically triggered as soon as a write access is performed on ACC\_MR or ACC\_ACR registers (whatever the register data content).

When ISEL = 0, the mask period is  $8 \cdot t_{MCK}$ , else  $128 \cdot t_{MCK}$ .

The masking period is reported by reading a negative value (bit 31 set) on ACC\_ISR register

### 42.6.5 Fault Mode

The FAULT output can be used to propagate a comparison match and act immediately via combinatorial logic by using the FAULT output which is directly connected to the FAULT input of the PWM.

The source of the FAULT output can be configured to be either a combinational value derived from the analog comparator output or the MCK resynchronized value (Refer to [Figure 42-1 "Analog Comparator Controller Block Diagram"](#)).

## 42.7 Analog Comparator Controller (ACC) User Interface

Table 42-4. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	ACC_CR	Write-only	
0x04	Mode Register	ACC_MR	Read-write	0
0x08-0x20	Reserved			
0x24	Interrupt Enable Register	ACC_IER	Write-only	
0x28	Interrupt Disable Register	ACC_IDR	Write-only	
0x2C	Interrupt Mask Register	ACC_IMR	Read-only	0
0x30	Interrupt Status Register	ACC_ISR	Read-only	0
0x34-0x90	Reserved			
0x94	Analog Control Register	ACC_ACR	Read-write	0
0x98-0xE0	Reserved			
0xE4	Write Protect Mode Register	ACC_WPMR	Read-write	0
0xE8	Write Protect Status Register	ACC_WPSR	Read-only	0
0xEC-0xF8	Reserved			
0xFC	Reserved	-	-	-

#### 42.7.1 ACC Control Register

**Name:** ACC\_CR  
**Address:** 0x400BC000  
**Access:** Write-only

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
-	-	-	-	-	-	-	SWRST

- **SWRST: SoftWare ReSeT**

0 = No effect.

1 = Resets the module.

#### 42.7.2 ACC Mode Register

**Name:** ACC\_MR

**Address:** 0x400BC004

**Access:** Read-write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	FE	SELFS	INV	-	EDGETYP	ACEN	
7	6	5	4	3	2	1	0
-		SELPLUS		-		SELMINUS	

This register can only be written if the WPEN bit is cleared in the [ACC Write Protect Mode Register](#).

- **SELMINUS: SELECTION for MINUS comparator input**

0..7 = selects the input to apply on analog comparator SELMINUS comparison input.

Value	Name	Description
0	TS	Select TS
1	ADVREF	Select ADVREF
2	DAC0	Select DAC0
3	DAC1	Select DAC1
4	AD0	Select AD0
5	AD1	Select AD1
6	AD2	Select AD2
7	AD3	Select AD3

- **SELPLUS: SELECTION for PLUS comparator input**

0..7 = selects the input to apply on analog comparator SELPLUS comparison input.

Value	Name	Description
0	AD0	Select AD0
1	AD1	Select AD1
2	AD2	Select AD2
3	AD3	Select AD3
4	AD4	Select AD4
5	AD5	Select AD5
6	AD6	Select AD6
7	AD7	Select AD7

- **ACEN: Analog Comparator ENable**

0 (DIS) = Analog Comparator Disabled.

1 (EN) = Analog Comparator Enabled.

- **EDGETYP: EDGE TYPe**

Value	Name	Description
0	RISING	only rising edge of comparator output
1	FALLING	falling edge of comparator output
2	ANY	any edge of comparator output

- **INV: INVert comparator output**

0 (DIS) = Analog Comparator output is directly processed.

1 (EN) = Analog Comparator output is inverted prior to being processed.

- **SELFS: SELECTION of Fault Source**

0 (CF) = The CF flag is used to drive the FAULT output.

1 (OUTPUT) = The output of the Analog Comparator flag is used to drive the FAULT output.

- **FE: Fault Enable**

0 (DIS) = The FAULT output is tied to 0.

1 (EN) = The FAULT output is driven by the signal defined by SELFS.

#### 42.7.3 ACC Interrupt Enable Register

**Name:** ACC\_IER  
**Address:** 0x400BC024  
**Access:** Write-only

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
-	-	-	-	-	-	-	CE

- **CE: Comparison Edge**

0 = No effect.

1 = Enables the interruption when the selected edge (defined by EDGETYP) occurs.

#### 42.7.4 ACC Interrupt Disable Register

**Name:** ACC\_IDR  
**Address:** 0x400BC028  
**Access:** Write-only

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
-	-	-	-	-	-	-	CE

- **CE: Comparison Edge**

0 = No effect.

1 = Disables the interruption when the selected edge (defined by EDGETYP) occurs.

#### 42.7.5 ACC Interrupt Mask Register

**Name:** ACC\_IMR

**Address:** 0x400BC02C

**Access:** Read-only

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
-	-	-	-	-	-	-	CE

- **CE: Comparison Edge**

0 = The interruption is disabled.

1 = The interruption is enabled.

#### 42.7.6 ACC Interrupt Status Register

**Name:** ACC\_ISR

**Address:** 0x400BC030

**Access:** Read-only

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

- **CE: Comparison Edge**

0 = No edge occurred (defined by EDGETYP) on analog comparator output since the last read of ACC\_ISR register.

1 = A selected edge (defined by EDGETYP) on analog comparator output occurred since the last read of ACC\_ISR register.

- **SCO: Synchronized Comparator Output**

Returns an image of Analog Comparator Output after being pre-processed (refer to [Figure 42-1 on page 1128](#)).

If INV = 0

SCO = 0 if inn > inp

SCO = 1 if inp > inn

If INV = 1

SCO = 1 if inn > inp

SCO = 0 if inp > inn

- **MASK:**

0 = The CE flag is valid.

1 = The CE flag and SCO value are invalid.

#### 42.7.7 ACC Analog Control Register

**Name:** ACC\_ACR

**Address:** 0x400BC094

**Access:** Read-write

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
-	-	-	-	-	HYST	ISEL	

This register can only be written if the WPEN bit is cleared in [ACC Write Protect Mode Register](#).

- **ISEL: Current SELECTION**

Refer to the product Electrical Characteristics.

0 (LOPW) = low power option.

1 (HISP) = high speed option.

- **HYST: HYSTeresis selection**

0 to 3: Refer to the product Electrical Characteristics.

#### 42.7.8 ACC Write Protect Mode Register

**Name:** ACC\_WPMR

**Address:** 0x400BC0E4

**Access:** Read-write

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x414343 (“ACC” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x414343 (“ACC” in ASCII).

Protects the registers:

- [“ACC Mode Register” on page 1133](#)
- [“ACC Analog Control Register” on page 1138](#)

- **WPKEY: Write Protect KEY**

This security code is needed to set/reset the WPROT bit value (see [Section 42.6.3 “Write Protection System”](#) for details).

Must be filled with “ACC” ASCII code.

#### 42.7.9 ACC Write Protect Status Register

**Name:** ACC\_WPSR

**Address:** 0x400BC0E8

**Access:** Read-only

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
-	-	-	-	-	-	-	WPROTERR

- **WPROTERR: Write PROtection ERRoR**

0 = No Write Protect Violation has occurred since the last read of the ACC\_WPSR register.

1 = A Write Protect Violation (WPEN = 1) has occurred since the last read of the ACC\_WPSR register.

## 43. Analog-Front-End Controller (AFEC)

### 43.1 Description

The Analog-Front-End Controller (AFEC) is based on a 12-bit Analog-to-Digital Converter (ADC) managed by an AFE Controller. Refer to the Block Diagram: [Figure 43-1](#). It also integrates a 16-to-1 analog multiplexer, making possible the analog-to-digital conversions of 16 analog lines. The conversions extend from 0V to ADVREF. The AFEC supports an 10-bit or 12-bit resolution mode which can be extended up to a 16-bit resolution by digital averaging, and conversion results are reported in a common register for all channels, as well as in a channel-dedicated register. Software trigger, external trigger on rising edge of the ADTRG pin or internal triggers from Timer Counter output(s) are configurable.

The comparison circuitry allows automatic detection of values below a threshold, higher than a threshold, in a given range or outside the range, thresholds and ranges being fully configurable.

The AFE Controller internal fault output is directly connected to PWM Fault input. This input can be asserted by means of comparison circuitry in order to immediately put the PWM outputs in a safe state (pure combinational path).

The AFEC also integrates a Sleep Mode and a conversion sequencer and connects with a PDC channel. These features reduce both power consumption and processor intervention.

This AFEC has a selectable single-ended or fully differential input and benefits from a 2-bit programmable gain. A whole set of reference voltages is generated internally from a single external reference voltage node that may be equal to the analog supply voltage. An external decoupling capacitance is required for noise filtering.

A digital error correction circuit based on the multi-bit redundant signed digit (RSD) algorithm is employed in order to reduce INL and DNL errors.

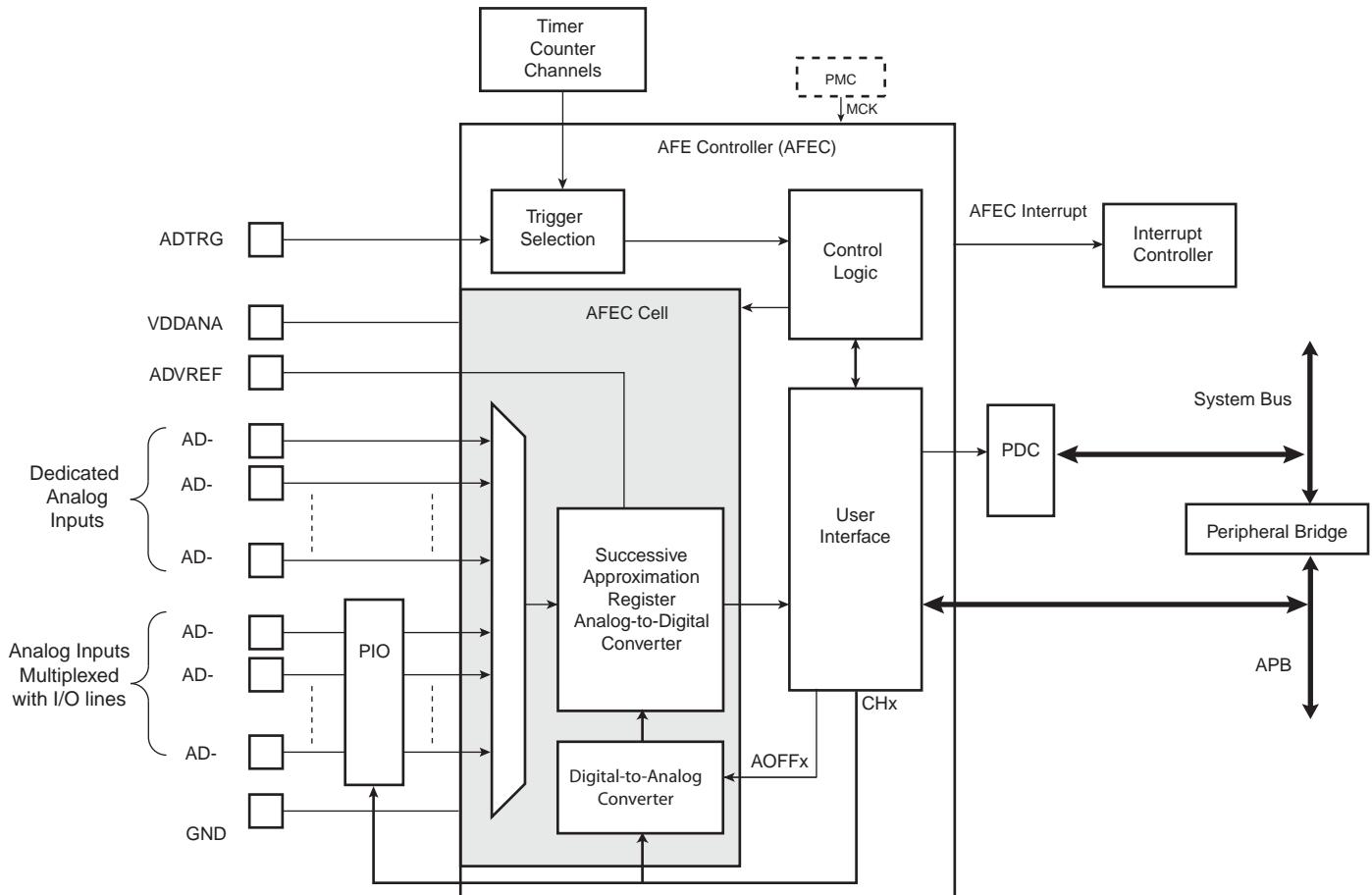
Finally, the user can configure AFEC timings, such as Startup Time and Tracking Time.

### 43.2 Embedded Characteristics

- 12-bit Resolution up to 16-bit resolution by digital averaging
- 1 MHz Conversion Rate
- Wide Range Power Supply Operation
- Selectable Single Ended or Differential Input Voltage
- Programmable Gain For Maximum Full Scale Input Range 0 - VDD
- Programmable Offset per channel
- Integrated Multiplexer Offering Up to 16 Independent Analog Inputs
- Individual Enable and Disable of Each Channel
- Hardware or Software Trigger
  - External Trigger Pin
  - Timer Counter Outputs (Corresponding TIOA Trigger)
  - PWM Event Line
- Drive of PWM Fault Input
- PDC Support
- Possibility of AFEC Timings Configuration
- Two Sleep Modes and Conversion Sequencer
  - Automatic Wake-up on Trigger and Back to Sleep Mode after Conversions of all Enabled Channels
  - Possibility of Customized Channel Sequence
- Standby Mode for Fast Wake-up Time Response
  - Power Down Capability
- Automatic Window Comparison of Converted Values
- Write Protect Registers

### 43.3 Block Diagram

**Figure 43-1. Analog-Front-End Converter Block Diagram**



Note: 1. DMA is sometimes referenced as PDC (Peripheral DMA Controller).

## 43.4 Signal Description

**Table 43-1. AFEC Pin Description**

Pin Name	Description
ADVREF	Reference voltage
AD0 - AD15 <sup>(1)</sup>	Analog input channels
ADTRG	External trigger

Note: 1. AD15 is not an actual pin but is connected to a temperature sensor.

## 43.5 Product Dependencies

### 43.5.1 Power Management

The AFE Controller is not continuously clocked. The programmer must first enable the AFE Controller MCK in the Power Management Controller (PMC) before using the AFE Controller. However, if the application does not require AFEC operations, the AFEC clock can be stopped when not needed and restarted when necessary. Configuring the AFE Controller does not require the AFEC clock to be enabled.

### 43.5.2 Interrupt Sources

The AFEC interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the AFEC interrupt requires the interrupt controller to be programmed first.

**Table 43-2. Peripheral IDs**

Instance	ID
AFEC0	30
AFEC1	31

### 43.5.3 Analog Inputs

The analog input pins can be multiplexed with PIO lines. In this case, the assignment of the AFEC input is automatically done as soon as the corresponding channel is enabled by writing the register AFEC\_CHER. By default, after reset, the PIO line is configured as input with its pull-up enabled and the AFEC input is connected to the GND.

### 43.5.4 Temperature Sensor

The temperature sensor is connected to Channel 15 of the AFEC.

The temperature sensor provides an output voltage  $V_T$  that is proportional to absolute temperature (PTAT). To activate the temperature sensor, TSON bit (AFEC\_ACR) needs to be set.

### 43.5.5 I/O Lines

The pin ADTRG may be shared with other peripheral functions through the PIO Controller. In this case, the PIO Controller should be set accordingly to assign the pin ADTRG to the AFEC function.

**Table 43-3. I/O Lines**

Instance	Signal	I/O Line	Peripheral
AFEC0	AFE0_ADTRG	PA8	B
AFEC0	AFE0_ADO	PA17	X1
AFEC0	AFE0_AD1	PA18	X1
AFEC0	AFE0_AD2/WKUP9	PA19	X1
AFEC0	AFE0_AD3/WKUP10	PA20	X1
AFEC0	AFE0_AD4/RTCOUT0	PB0	X1
AFEC0	AFE0_AD5/RTCOUT1	PB1	X1
AFEC0	AFE0_AD6	PC13	X1
AFEC0	AFE0_AD7	PC15	X1
AFEC0	AFE0_AD8	PC12	X1
AFEC0	AFE0_AD9	PC29	X1
AFEC0	AFE0_AD10	PC30	X1

**Table 43-3. I/O Lines**

AFEC0	AFE0_AD11	PC31	X1
AFEC0	AFE0_AD12	PC26	X1
AFEC0	AFE0_AD13	PC27	X1
AFEC0	AFE0_AD14	PC0	X1
AFEC1	AFE1_AD0/WKUP12	PB2	X1
AFEC1	AFE1_AD1	PB3	X1
AFEC1	AFE1_AD2	PA21	X1
AFEC1	AFE1_AD3	PA22	X1
AFEC1	AFE1_AD4	PC1	X1
AFEC1	AFE1_AD5	PC2	X1
AFEC1	AFE1_AD6	PC3	X1
AFEC1	AFE1_AD7	PC4	X1

#### 43.5.6 Timer Triggers

Timer Counters may or may not be used as hardware triggers depending on user requirements. Thus, some or all of the timer counters may be unconnected.

#### 43.5.7 PWM Event Line

PWM Event Lines may or may not be used as hardware triggers depending on user requirements.

#### 43.5.8 Fault Output

The AFE Controller has the FAULT output connected to the FAULT input of PWM. Please refer to [Section 43.6.15 "Fault Output"](#) and implementation of the PWM in the product.

#### 43.5.9 Conversion Performances

For performance and electrical characteristics of the AFEC, see the product DC Characteristics section.

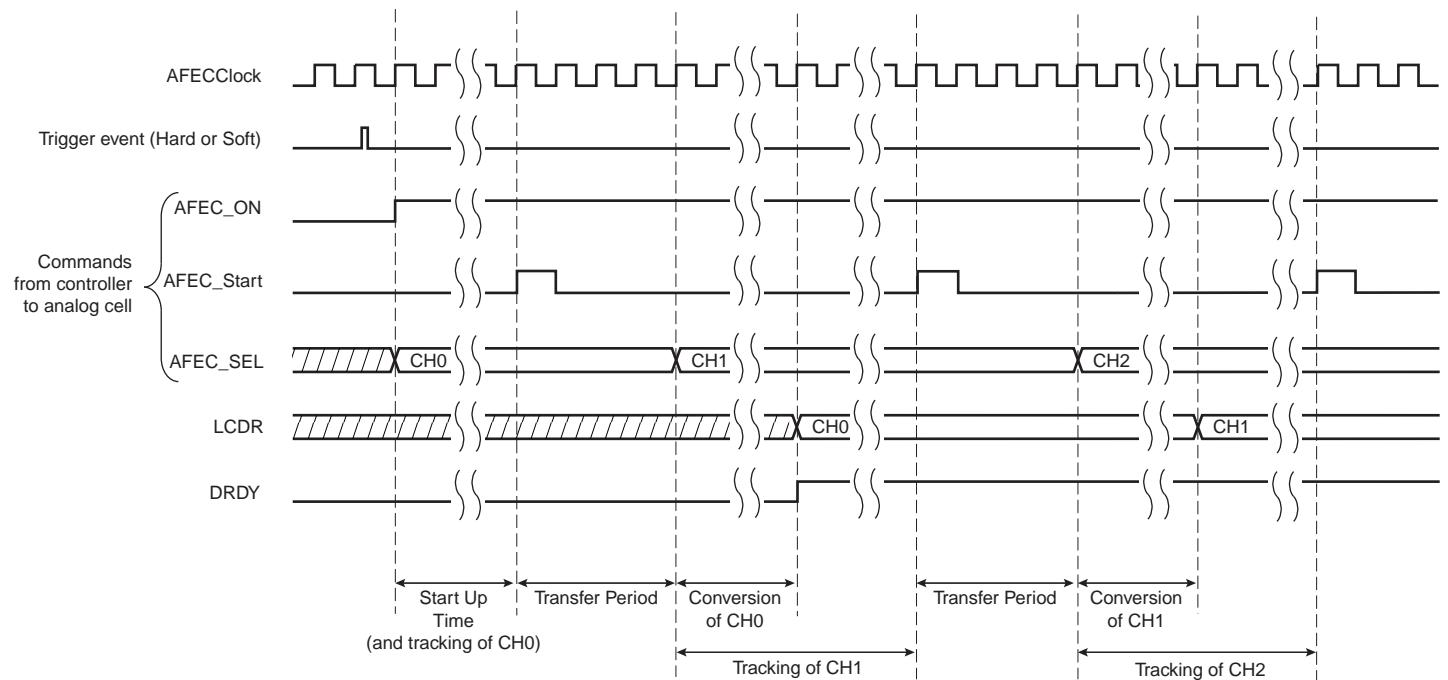
## 43.6 Functional Description

### 43.6.1 Analog-Front-End Conversion

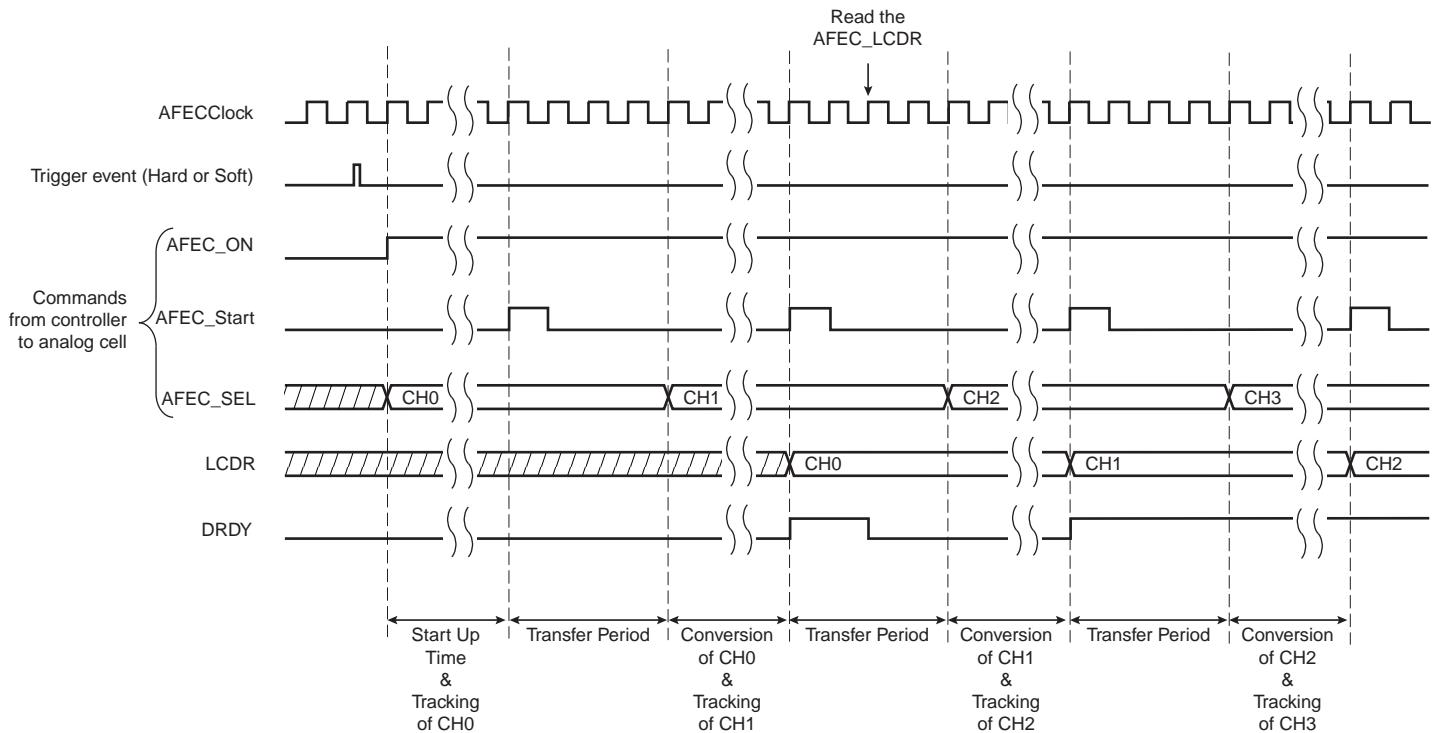
The AFEC uses the AFEC Clock to perform conversions. Converting a single analog value to a 12-bit digital data requires Tracking Clock cycles as defined in the field TRACKTIM of the “[AFEC Mode Register](#)” and Transfer Clock cycles as defined in the field TRANSFER of the same register. The AFEC Clock frequency is selected in the PRESCL field of the Mode Register (AFEC\_MR). The tracking phase starts during the conversion of the previous channel. If the tracking time is longer than the conversion time, the tracking phase is extended to the end of the previous conversion.

The AFEC clock range is between MCK/2, if PRESCL is 0, and MCK/512, if PRESCL is set to 255 (0xFF). PRESCL must be programmed in order to provide an AFEC clock frequency according to the parameters given in the product Electrical Characteristics section.

**Figure 43-2. Sequence of AFEC conversions when Tracking time > Conversion time**



**Figure 43-3. Sequence of AFEC conversions when Tracking time < Conversion time**



#### 43.6.2 Conversion Reference

The conversion is performed on a full range between 0V and the reference voltage pin ADVREF. Analog inputs between these voltages convert to values based on a linear conversion.

#### 43.6.3 Conversion Resolution

The AFEC supports 10-bit or 12-bit native resolutions. The 10-bit selection is performed by writing one to the RES field in the AFEC Extended Mode Register (AFEC\_EMR). By default, after a reset, the resolution is the highest and the DATA field in the data registers is fully used. By writing one to the RES field, the AFEC switches to the lowest resolution and the conversion results can be read in the lowest significant bits of the data register. The highest bits of the DATA field of the AFEC\_CDR register and of the LDATA field in the AFEC\_LCDR register read 0. Writing two, three or more to the RES field automatically enables the Enhanced Resolution Mode, see [Section 43.6.3.1](#) for details.

Moreover, when a PDC channel is connected to the AFEC, 12-bit or 10-bit resolution sets the transfer request size to 16 bits.

##### 43.6.3.1 Enhanced Resolution Mode

The Enhanced Resolution Mode is automatically enabled when 13-bit, 14-bit, 15-bit and 16-bit mode are selected in the AFEC Extended Mode Register (AFEC\_EMR). In this mode the AFE Controller will trade conversion performance for accuracy by averaging multiple samples.

To be able to increase the accuracy by averaging multiple samples it is important that some noise is present in the input signal. The noise level should be between one and two LSB peak-to-peak to get good averaging performance.

The performance cost of enabling 13-bit mode is 4 AFEC samples, which reduces the effective AFEC performance by a factor 4. This factor equals to 16 if the 14-bit mode is selected, to 64 if the 15-bit mode is selected, and to 256 if the 16-bit mode is selected. For 14-bit mode the effective sample rate is maximum AFEC sample rate divided by 16.

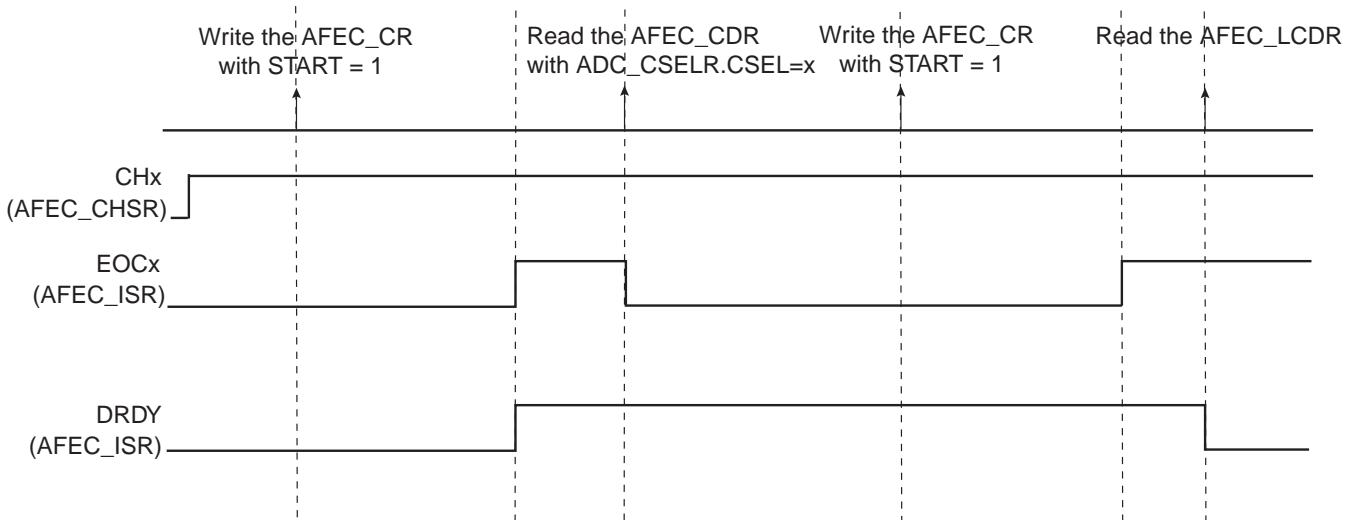
#### 43.6.4 Conversion Results

When a conversion is completed, the resulting 12-bit digital value is stored in an internal register (1 register for each channel) that can be read by means of the Channel Data Register (AFEC\_CDR) and in the AFEC Last Converted Data Register (AFEC\_LCDR). By setting the TAG option in the AFEC\_EMR, the AFEC\_LCDR presents the channel number associated to the last converted data in the CHNB field.

The channel EOC bit in the Status Register (AFEC\_ISR) is set and the DRDY is set. In the case of a connected PDC channel, DRDY rising triggers a data transfer request. In any case, either EOC and DRDY can trigger an interrupt.

Reading the AFEC\_CDR register clears the EOC bit which index corresponds to the value prior programmed in CSEL field of the AFEC\_CSELR register. Reading AFEC\_LCDR clears the DRDY bit and EOC bit corresponding to the last converted channel.

**Figure 43-4. EOCx and DRDY Flag Behavior**

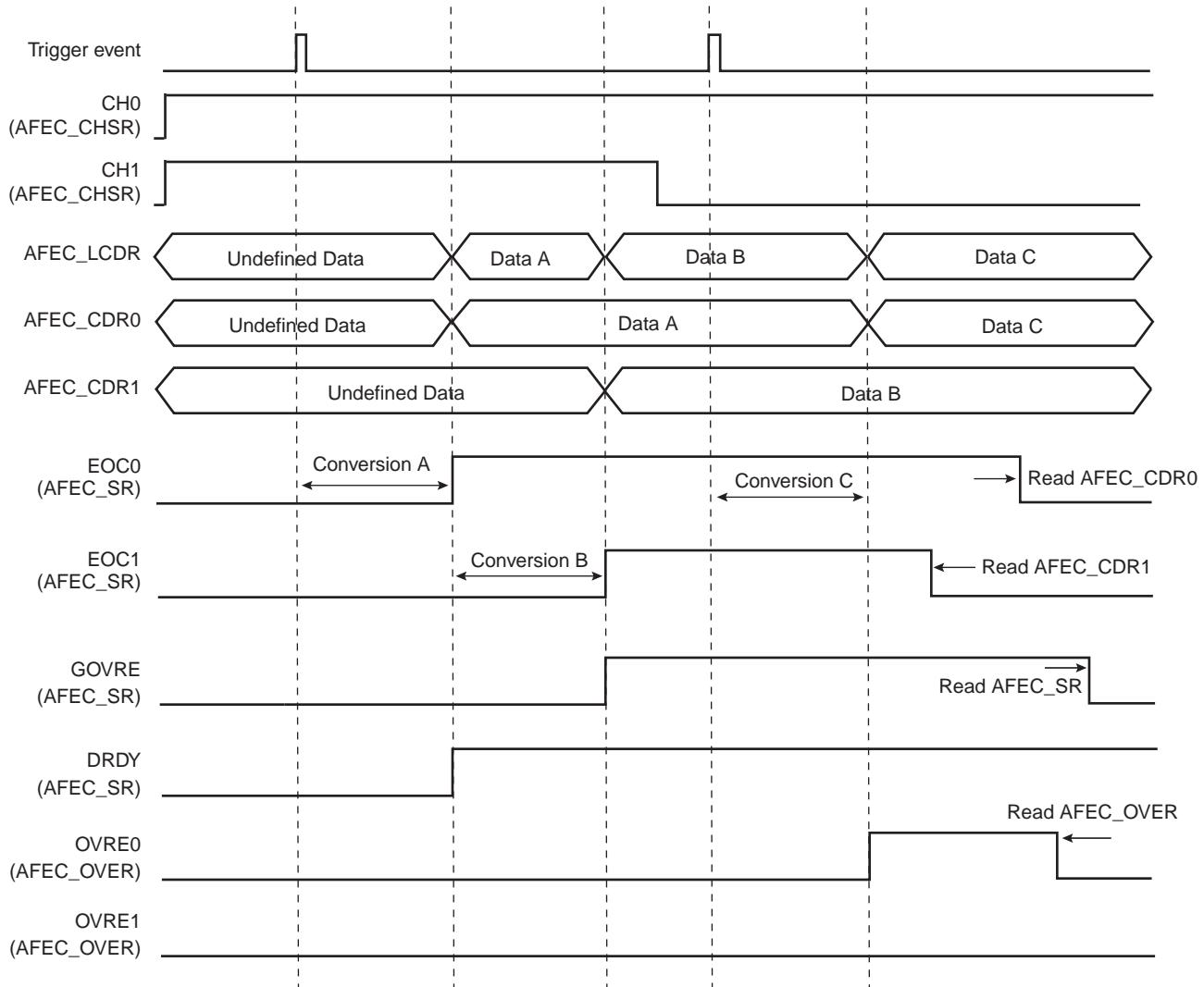


If the AFEC\_CDR is not read before further incoming data is converted, the corresponding Overrun Error (OVREx) flag is set in the Overrun Status Register (AFEC\_OVER).

Likewise, new data converted when DRDY is high sets the GOVRE bit (General Overrun Error) in AFEC\_ISR.

The OVREx flag is automatically cleared when AFEC\_OVER is read, and GOVRE flag is automatically cleared when AFEC\_ISR is read.

**Figure 43-5. GOVRE and OVREx Flag Behavior**



**Warning:** If the corresponding channel is disabled during a conversion or if it is disabled and then reenabled during a conversion, its associated data and its corresponding EOC and OVRE flags in AFEC\_SR are unpredictable.

#### 43.6.5 Conversion Triggers

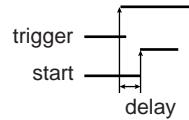
Conversions of the active analog channels are started with a software or hardware trigger. The software trigger is provided by writing the Control Register (AFEC\_CR) with the START bit at 1.

The hardware trigger can be one of the TIOA outputs of the Timer Counter channels, PWM Event line, or the external trigger input of the AFEC (ADTRG). The hardware trigger is selected with the TRGSEL field in the Mode Register (AFEC\_MR). The selected hardware trigger is enabled with the TRGEN bit in the Mode Register (AFEC\_MR).

The minimum time between 2 consecutive trigger events must be strictly greater than the duration time of the longest conversion sequence according to configuration of registers AFEC\_MR, AFEC\_CHSR, AFEC\_SEQR1, AFEC\_SEQR2.

If a hardware trigger is selected, the start of a conversion is triggered after a delay starting at each rising edge of the selected signal. Due to asynchronous handling, the delay may vary in a range of 2 MCK clock periods to 1 AFEC clock period.

**Figure 43-6. Conversion Start with the Hardware Trigger**



If one of the TIOA outputs is selected, the corresponding Timer Counter channel must be programmed in Waveform Mode.

Only one start command is necessary to initiate a conversion sequence on all the channels. The AFEC hardware logic automatically performs the conversions on the active channels, then waits for a new request. The Channel Enable (AFEC\_CHER) and Channel Disable (AFEC\_CHDR) Registers permit the analog channels to be enabled or disabled independently.

If the AFEC is used with a PDC, only the transfers of converted data from enabled channels are performed and the resulting data buffers should be interpreted accordingly.

#### 43.6.6 Sleep Mode and Conversion Sequencer

The AFEC Sleep Mode maximizes power saving by automatically deactivating the AFEC when it is not being used for conversions. Sleep Mode is selected by setting the SLEEP bit in the Mode Register AFEC\_MR.

The Sleep mode is automatically managed by a conversion sequencer, which can automatically process the conversions of all channels at lowest power consumption.

This mode can be used when the minimum period of time between 2 successive trigger events is greater than the startup period of Analog-Digital converter (See the product AFEC Characteristics section).

When a start conversion request occurs, the AFEC is automatically activated. As the analog cell requires a start-up time, the logic waits during this time and starts the conversion on the enabled channels. When all conversions are complete, the AFEC is deactivated until the next trigger. Triggers occurring during the sequence are not taken into account.

A fast wake-up mode is available in the AFEC Mode Register (AFEC\_MR) as a compromise between power saving strategy and responsiveness. Setting the FWUP bit to '1' enables the fast wake-up mode. In fast wake-up mode the AFEC cell is not fully deactivated while no conversion is requested, thereby providing less power saving but faster wake-up.

The conversion sequencer allows automatic processing with minimum processor intervention and optimized power consumption. Conversion sequences can be performed periodically using a Timer/Counter output or the PWM event line. The periodic acquisition of several samples can be processed automatically without any intervention of the processor thanks to the PDC.

The sequence can be customized by programming the Sequence Channel Registers, AFEC\_SEQR1 and AFEC\_SEQR2 and setting to 1 the USEQ bit of the Mode Register (AFEC\_MR). The user can choose a specific order of channels and can program up to 16 conversions by sequence. The user is totally free to create a personal sequence, by writing channel numbers in AFEC\_SEQR1 and AFEC\_SEQR2. Not only can channel numbers be written in any sequence, channel numbers can be repeated several times. Only enabled sequence bitfields are converted, consequently to program a 15-conversion sequence, the user can simply put a disable in AFEC\_CHSR[15], thus disabling the 16THCH field of AFEC\_SEQR2.

Note: The reference voltage pins always remain connected in normal mode as in sleep mode.

### 43.6.7 Comparison Window

The AFE Controller features automatic comparison functions. It compares converted values to a low threshold or a high threshold or both, according to the CMPMODE function chosen in the Extended Mode Register (AFEC\_EMR). The comparison can be done on all channels or only on the channel specified in CMPSEL field of AFEC\_EMR. To compare all channels the CMP\_ALL parameter of AFEC\_EMR should be set.

Moreover a filtering option can be set by writing the number of consecutive comparison errors needed to raise the flag. This number can be written and read in the CMPPFILTER field of AFEC\_EMR.

The flag can be read on the COMPE bit of the Interrupt Status Register (AFEC\_IS1R) and can trigger an interrupt.

The High Threshold and the Low Threshold can be read/write in the Comparison Window Register (AFEC\_CWR).

### 43.6.8 Differential Inputs

The AFEC can be used either as a single ended AFEC (DIFF bit equal to 0) or as a fully differential AFEC (DIFF bit equal to 1) as shown in [Figure 43-7](#). By default, after a reset, the AFEC is in single ended mode.

If ANACH is set in AFEC\_MR the AFEC can apply a different mode on each channel. Otherwise the parameters of CH0 are applied to all channels.

The same inputs are used in single ended or differential mode.

In single ended mode, inputs are managed by a 16:1 channels analog multiplexer. In the fully differential mode, inputs are managed by an 8:1 channels analog multiplexer. See [Table 43-4](#) and [Table 43-5](#).

**Table 43-5. Input Pins and Channel Number In Differential Mode**

Input Pins	Channel Number
AD0-AD1	CH0
AD2-AD3	CH2
AD4-AD5	CH4
AD6-AD7	CH6
AD8-AD9	CH8
AD10-AD11	CH10
AD12-AD13	CH12
AD14-AD15	CH14

**Table 43-4. Input Pins and Channel Number in Single Ended Mode**

Input Pins	Channel Number
AD0	CH0
AD1	CH1
AD2	CH2
AD3	CH3
AD4	CH4
AD5	CH5
AD6	CH6
AD7	CH7
AD8	CH8
AD9	CH9
AD10	CH10
AD11	CH11
AD12	CH12

**Table 43-4. Input Pins and Channel Number in Single Ended Mode (Continued)**

Input Pins	Channel Number
AD13	CH13
AD14	CH14
AD15	CH15

### 43.6.9 Input Gain and Offset

The AFEC has a built in Programmable Gain Amplifier (PGA) and Programmable Offset per channel (through a DAC).

The Programmable Gain Amplifier can be set to gains of 1/2, 1, 2 and 4. The Programmable Gain Amplifier can be used either for single ended applications or for fully differential applications.

If ANACH is set in AFEC\_MR the AFEC can apply different gain and offset on each channel. Otherwise the parameters of CH0 are applied to all channels.

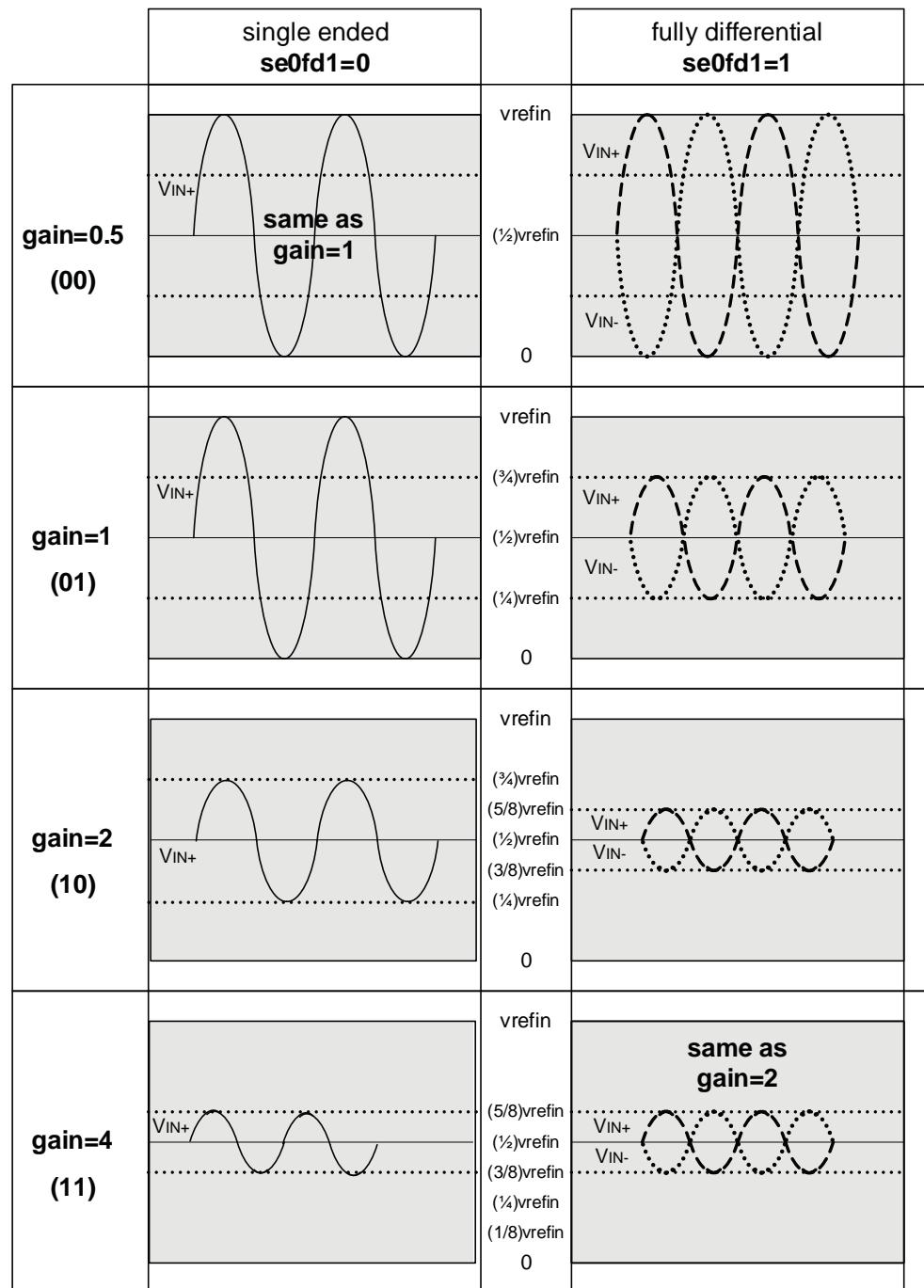
The gain is configurable through the GAIN bit of the Channel Gain Register (AFEC\_CGR) as shown in [Table 43-6](#).

**Table 43-6. Gain of the Sample and Hold Unit: GAIN Bits and DIFF Bit.**

GAIN<0:1>	GAIN (DIFF = 0)	GAIN (DIFF = 1)
00	1	0.5
01	1	1
10	2	2
11	4	2

Analog offset of the AFEC can be configured by the AOFF bitfield in the Channel Offset Compensation Register (see [Section 43.7.21 "AFEC Channel Offset Compensation Register"](#)). The Offset is only available in Single Ended Mode. When AOFF is configured to 0, the offset equals 0, when 4095 the offset equals ADVREF-1LSB. All possible offset values are provided between these 2 limits according to the following formula: AOFF\*(VREF/4096).

Figure 43-7. Analog Full Scale Ranges in Single Ended/Differential Applications Versus Gain



#### 43.6.10 AFEC Timings

Each AFEC has its own minimal Startup Time that is programmed through the field STARTUP in the Mode Register, AFEC\_MR.

A minimal Tracking Time is necessary for the AFEC to guarantee the best converted final value between two channel selections. This time has to be programmed through the TRACKTIM bit field in the Mode Register, AFEC\_MR.

When the gain, offset or differential input parameters of the analog cell change between two channels, the analog cell may need a specific settling time before starting the tracking phase. In that case, the controller automatically waits during the settling time defined in the "[AFEC Mode Register](#)". Obviously, if the ANACH option is not set, this time is unused.

**Warning:** No input buffer amplifier to isolate the source is included in the AFEC. This must be taken into consideration to program a precise value in the TRACKTIM field. See the product AFEC Characteristics section.

#### 43.6.11 Temperature Sensor

The temperature sensor is internally connected to channel 15.

The measure of the temperature can be made in different ways through the AFE Controller. The different methods for the measure depends on the configuration bits TRGEN in the AFEC\_MR register and CH15 in the AFEC\_CHSR register.

Temperature measure can be triggered like the other channels by enabling its associated conversion channel 15 by writing 1 in CH15 of the AFEC\_CHER register.

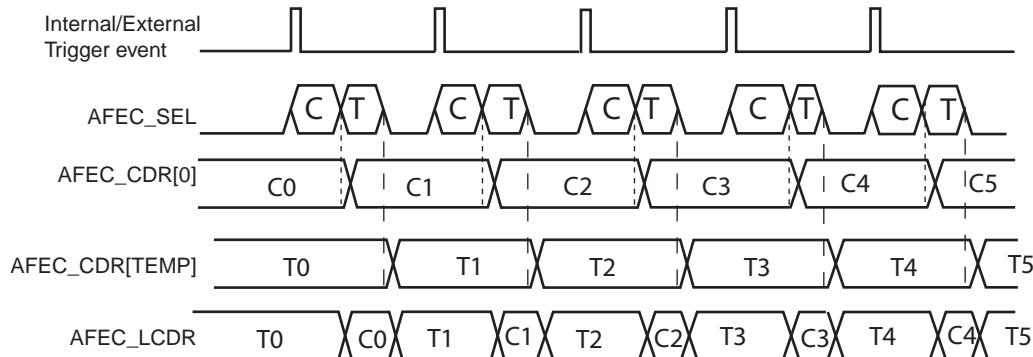
In any case enabling CH15 in AFEC\_CHSR enables the temperature sensor channel even if user sequence is used (in such case last element of the sequence is always the temperature sensor channel). If temperature sensor is to be used with user sequence it should be enabled using CH15 bit in AFEC\_CHSR.

Manual start can be only performed if TRGEN bit in AFEC\_MR is disabled. When START bit in the AFEC\_CR register is written to 1, the temperature sensor channel conversion is scheduled together with the other enabled channels (if any). The result of the conversion is placed in an internal register that can be read in the AFEC\_CDR register (AFEC\_CSEL must be programmed accordingly prior to read AFEC\_CDR) and the associated flag EOC15 is set in the AFEC\_ISR register.

If the TRGEN bit is set in AFEC\_MR, the channel of the temperature sensor can be periodically converted together with the other enabled channels if the RTCT bit is set in the AFEC\_TEMPREG register, the result is placed on AFEC\_LCDR and an internal register (can be read in AFEC\_CDR register). Thus the temperature conversion result is a part of the Peripheral DMA Controller buffer. The temperature channel can be enabled/disabled at anytime, but this may not be optimal for downstream processing.

**Figure 43-8. Non-Optimized Temperature Conversion**

AFEC\_CHSR[TEMP]= 1, AFEC\_MR.TRGEN=1 and AFEC\_TEMPMR.RTCT = 0



C: Classic AFEC Conversion Sequence - T: Temperature Sensor Channel

Assuming AFEC\_CHSR[0] = 1 and AFEC\_CHSR[TEMP] = 1  
where TEMP is the index of the temperature sensor channel

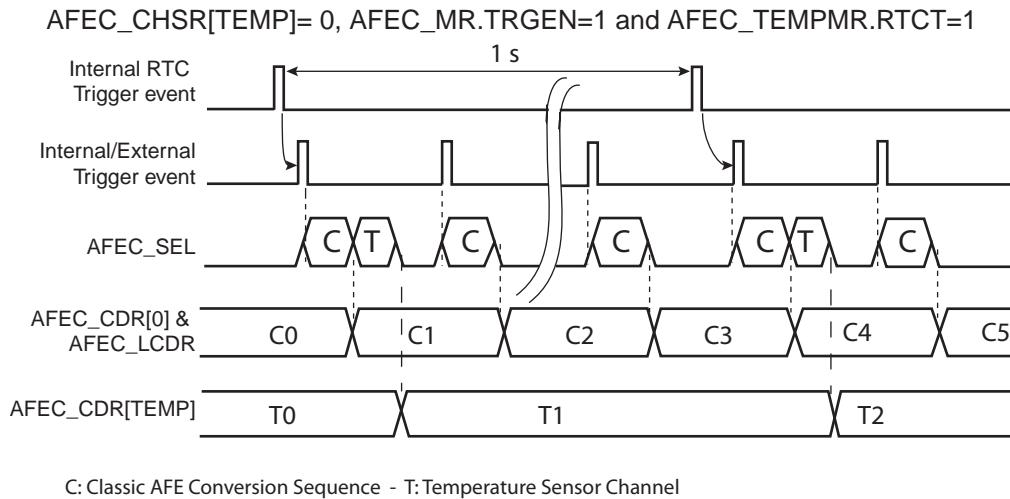
trig.event1 →	0	AFEC_CDR[0]	DMA Transfer Base Address (BA)
	0	AFEC_CDR[TEMP]	BA + 0x02
trig.event2 →	0	AFEC_CDR[0]	BA + 0x04
	0	AFEC_CDR[TEMP]	BA + 0x06
trig.event3 →	0	AFEC_CDR[0]	BA + 0x08
	0	AFEC_CDR[TEMP]	BA + 0x0A

The temperature factor having a slow variation rate and potentially being totally different from the other conversion channels, the AFE Controller allows a different way of triggering the measure when RTCT is set in the AFEC\_TEMPMR register but the CH15 is not set in the AFEC\_CHSR register.

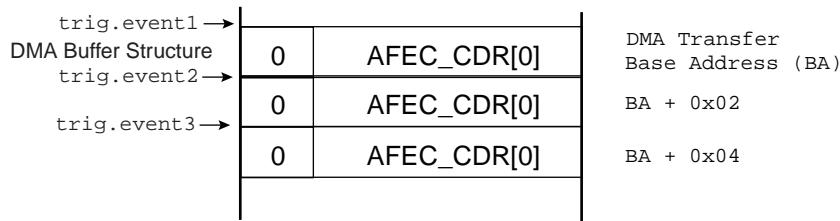
Under these conditions the measure is triggered every second by means of an internal trigger generated by RTC, always enabled and totally independent of the triggers used for other channels and selected through TRGSEL bitfield of the AFEC\_MR register. In this mode of operation the temperature sensor is only powered for a period of time covering startup time and conversion time.

Every second, a conversion is scheduled for channel 15 but the result of the conversion is only uploaded on an internal register (read by means of AFEC\_CDR register) and not in the AFEC\_LCDR register. Therefore there is no change in the structure of the Peripheral DMA Controller buffer due to the conversion of the temperature channel, only the enabled channel are kept in the buffer. The end of conversion of the temperature channel is reported by means of EOC15 flag in the AFEC\_ISR register.

**Figure 43-9. Optimized Temperature Conversion combined with classical conversions**



Assuming AFEC\_CHSR[0] = 1 and A\_CHSR[TEMP] = 1  
where TEMP is the index of the temperature sensor channel

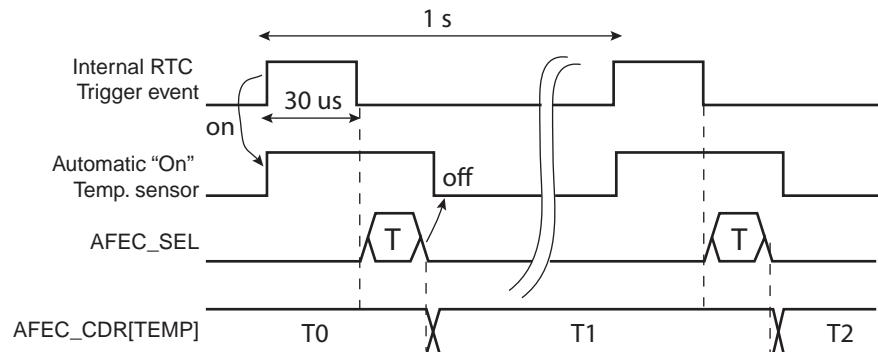


If RTCT = 1, then if TRGEN is disabled and all channels are disabled (AFEC\_CHSR = 0), then only channel 15 is converted at a rate of 1 conversion per second.

This mode of operation, when combined with the sleep mode operation of the AFE Controller, provides a low power mode for temperature measure (assuming there is no other AFEC conversion to schedule at high sampling rate or simply no other channel to convert).

**Figure 43-10. Temperature Conversion Only**

AFEC\_CHSR= 0, AFE\_MR.TRGEN=0 and AFEC\_TEMPMR.RTCT = 1  
AFEC\_TEMPMR.RTCT=1



Moreover, it is possible to raise a flag only if there is predefined change in the temperature measure. The user can define a range of temperature or a threshold in the AFEC\_TEMPWR register and the mode of comparison that can be programmed into the AFEC\_TEMPMR register by means of TEMPCMPMOD bitfield. These values define the way the TEMPCHG flag will be raised in the AFEC\_IS1R register.

The TEMPCHG flag can be used to trigger an interrupt if there is something to update/modify in the system resulting from a temperature change.

In any case, if temperature sensor measure is configured, the temperature can be read at anytime in AFEC\_CDR (AFEC\_CSEL must be programmed accordingly prior to read AFEC\_CDR) without any specific software intervention.

#### 43.6.12 Enhanced Resolution Mode and Digital Averaging Function

The Enhanced Resolution Mode is enabled if RES field is set to 13-bits resolution or more in the AFEC Extended Mode Register (AFEC\_EMR). FREERUN mode is not supported if Enhanced Resolution Mode is used.

There is no averaging on the temperature sensor channel if the temperature sensor measure is triggered on RTC event (see “[Temperature Sensor](#)” on page 1153).

In this mode the AFE Controller trades conversion performance for accuracy by averaging multiple samples, thus providing a digital low-pass filter function.

If 1-bit enhancement resolution is selected (RES=2 in the AFEC\_EMR register), the AFEC effective sample rate is maximum AFEC sample rate divided by 4, therefore the oversampling ratio is 4.

When 2-bit enhancement resolution is selected (RES=3 in the AFEC\_EMR register), the AFEC effective sample rate is maximum AFEC sample rate divided by 16 (oversampling ratio is 16).

When 3-bit enhancement resolution is selected (RES=4 in the AFEC\_EMR register), the AFEC effective sample rate is maximum AFEC sample rate divided by 64 (oversampling ratio is 64).

When 4-bit enhancement resolution is selected (RES=5 in the AFEC\_EMR register), the AFEC effective sample rate is maximum AFEC sample rate divided by 256 (oversampling ratio is 256).

The selected oversampling ratio applies to all enabled channels (except temperature sensor channel if triggered by RTC event).

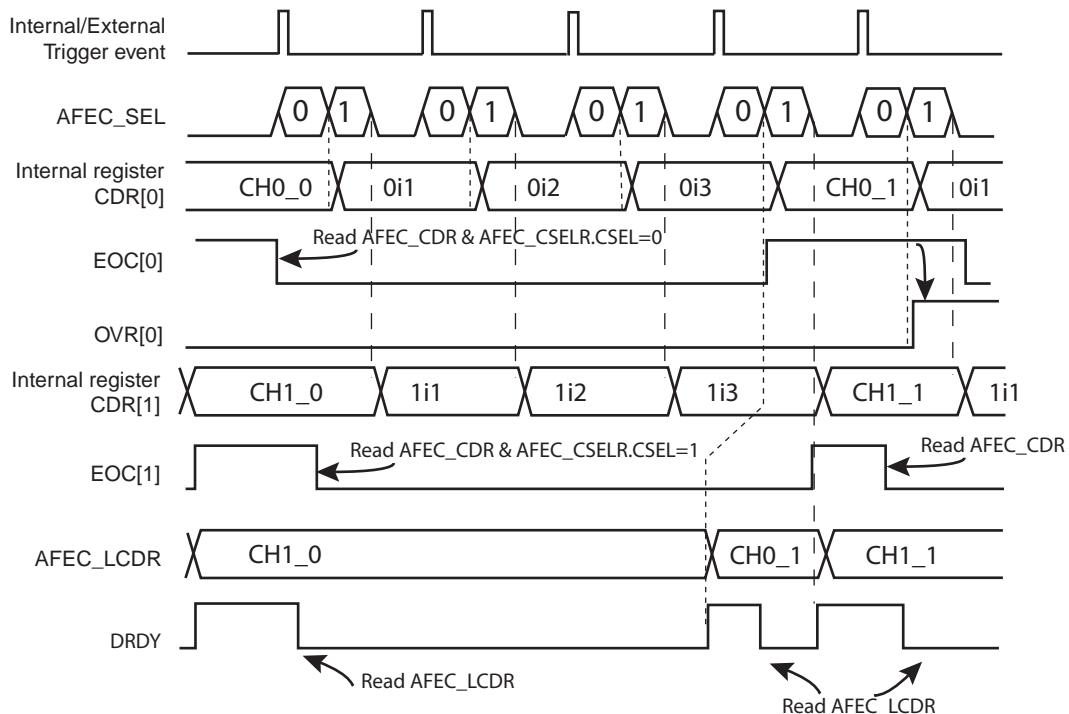
The average result is valid into an internal register (read by means of the AFEC\_CDR register) only if EOCn (n corresponding to the index of the channel) flag is set in AFEC\_ISR and OVREN flag is cleared in the AFEC\_OVER register. The average result is valid for all channels in the AFEC\_LCDR register only if DRDY is set and GOVRE is cleared in the AFEC\_ISR register.

The samples can be defined in different ways for the averaging function depending on the configuration of the STM bit in the Extended Mode register (AFEC\_EMR) and USEQ bit in the Mode register (AFEC\_MR).

When USEQ is cleared, there are 2 possible ways to generate the averaging through the trigger event. If the STM bit is cleared in the AFEC\_EMR register, every trigger event generates one sample for each enabled channel as described in [Figure 43-11, on page 1157](#). Therefore 4 trigger events are requested to get the result of averaging if RES=2.

**Figure 43-11. Digital Averaging Function Waveforms over multiple trigger events**

AFEC\_EMR.RES=2 STM=0, AFEC\_CHSR[1:0]=0x3 and AFEC\_MR.USEQ=0

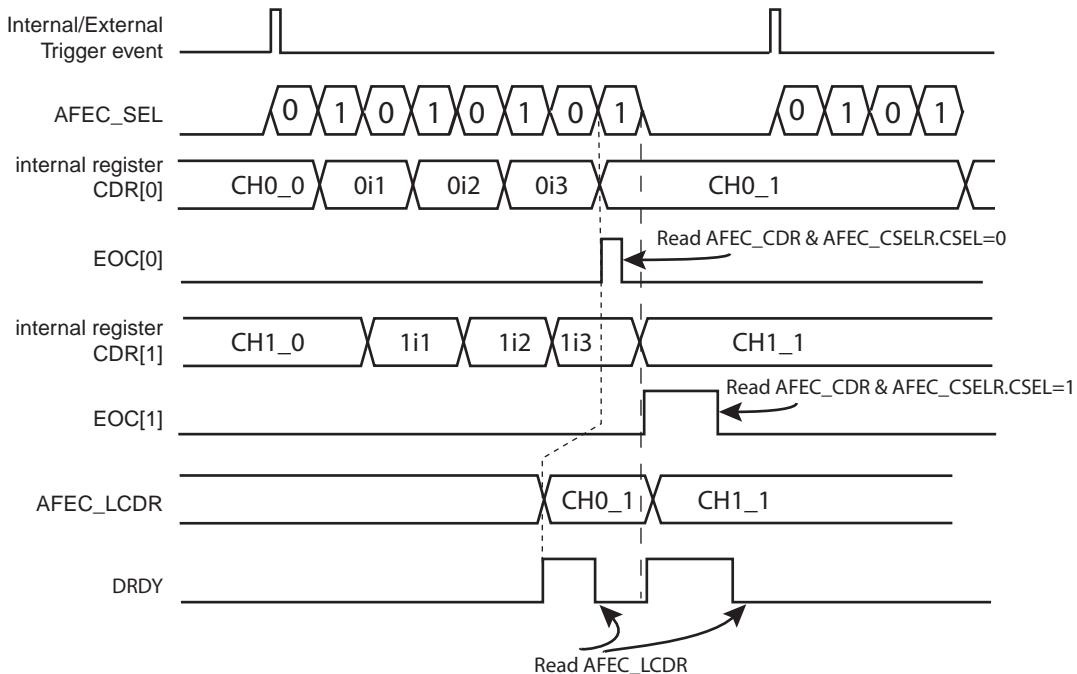


Note: 0i1,0i2,0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1\_0/1 are final result of average function.

If STM = 1 in the AFEC\_EMR register and USEQ=0 in the AFEC\_MR register then the sequence to be converted defined in the AFEC\_CHSR register is automatically repeated n times (where n corresponds to the oversampling ratio defined in RES field in the AFEC\_EMR register). As a consequence, only 1 trigger is required to get the result of the averaging function as described in [Figure 43-11, on page 1157](#).

**Figure 43-12. Digital Averaging Function Waveforms on a single trigger event**

AFEC\_EMR.RES=2 STM=1, AFEC\_CHSR[1:0]= 0x3 and AFEC\_MR.USEQ=0



Note: 0i1, 0i2, 0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1\_0/1 are final result of average function.

When USEQ is set, the user can define the channel sequence to be converted by configuring AFEC\_SEQRx and AFEC\_CHER registers so that channels are not interleaved during the averaging period. Under these conditions, a sample is defined for each end of conversion as described in [Figure 43-13, on page 1159](#).

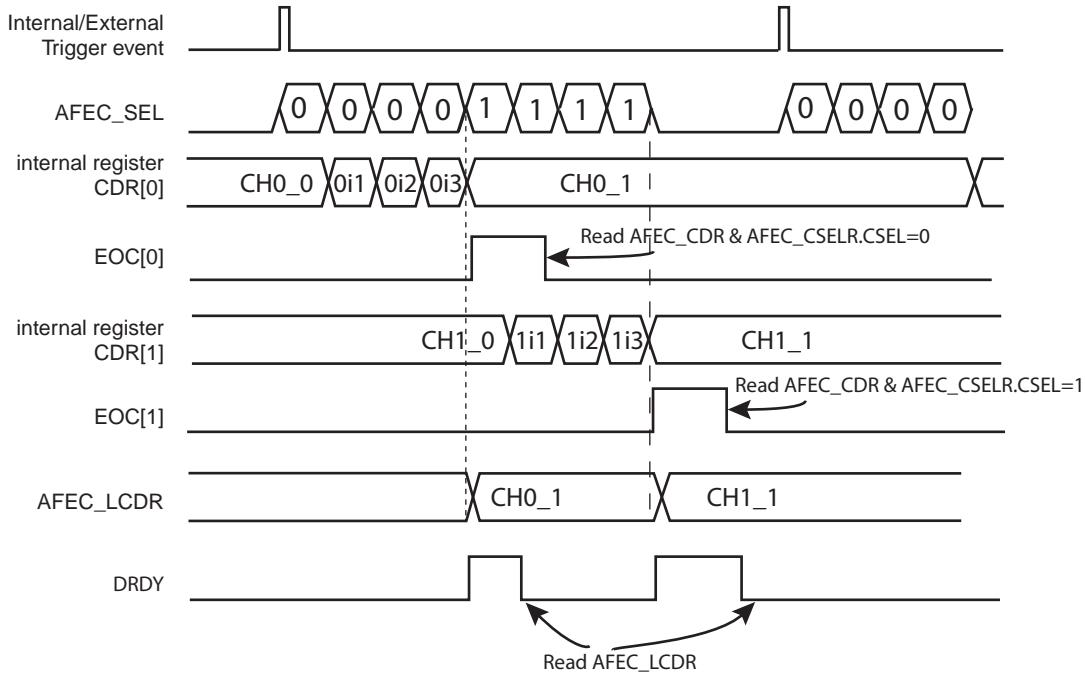
Therefore if the same channel is configured to be converted 4 times consecutively and the RES=2 in the extended mode register (AFEC\_EMR) the averaging result is placed in the corresponding channel internal data register (read by means of the AFEC\_CDR register) and the last converted data register (AFEC\_LCDR) for each trigger event.

In such case, the AFEC effective sample rate remains the maximum AFEC sample rate divided by 4.

When USEQ=1 and the RES field enables the Enhanced Resolution Mode, it is important to notice that the user sequence must be a sequence being an integer multiple of 4 (i.e. the number of the enabled channel in the channel status register AFEC\_CHSR must be an integer multiple of 4 and the AFEC\_SEQRx must be a series of 4 times the same channel index).

**Figure 43-13. Digital Averaging Function Waveforms on single trigger event, non-interleaved**

AFEC\_EMR.EMR=2 , STM = 1, AFEC\_CHSR[7:0]= 0xFF and AFEC\_MR.USEQ=1  
AFEC\_SEQ1R = 0x1111\_0000



Note: 0i1, 0i2, 0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1\_0/1 are final result of average function.

The Enhanced Resolution mode doesn't apply to temperature sensor channel when used with the RTCT bit (see [Figure 43-9](#)).

#### 43.6.13 Automatic Calibration

The AFEC features an automatic calibration (AUTOCALIB) mode for gain errors (calibration).

The automatic calibration sequence can be started at any time writing to '1' the AUTOCAL bit of the AFEC Control Register. The end of calibration sequence is given by the EOICAL bit in the interrupt status register (AFEC\_IS1R), and an interrupt is generated if the EOICAL interrupt has been enabled (AFEC\_IER).

If FREERUN mode is to be used then automatic calibration must be run before enabling the FREERUN mode. In any case automatic calibration should not be started while FREERUN mode is active.

The calibration sequence performs an automatic calibration on all enabled channels. The gain settings of all enabled channels must be set before starting the AUTOCALIB sequence. For each calibrated channels, the corresponding OFF bit in AFEC\_CDOR register must be also programmed to 1 prior to launch the autocalibration sequence.

If the gain (AFEC\_CGR register) for a given channel is changed, the AUTOCALIB sequence must then be started again.

The calibration data (on one or more enabled channels) is stored in the internal AFEC memory.

Then, when a new conversion is started (on one or more enabled channels), the converted value (in the AFEC\_LCDR register or internal data registers read by means of the AFEC\_CDR register) is a calibrated value.

Autocalibration is for settings, not for channels. Therefore, if a specific combination of gain and offset has been already calibrated, and a new channel with the same settings is enabled after the initial calibration, there is no need to restart a calibration. If different enabled channels have different gain and offset settings, the corresponding channels must be enabled before starting the calibration.

If a software reset is performed (SWRST bit in AFEC\_CR) or after power up (or wake-up from Backup mode), the calibration data in the AFEC memory is lost.

Changing the AFEC running mode (in the AFEC\_CR register) does not affect the calibration data.

Changing the AFEC reference voltage (ADVREF pin) requires a new calibration sequence.

For calibration time, offset and gain error after calibration, refer to the 12-bit AFEC electrical characteristics section of the product.

#### 43.6.14 Buffer Structure

The PDC read channel is triggered each time a new data is stored in the AFEC\_LCDR register. The same structure of data is repeatedly stored in the AFEC\_LCDR register each time a trigger event occurs. Depending on user mode of operation (AFEC\_MR, AFEC\_CHSR, AFEC\_SEQR1, AFEC\_SEQR2) the structure differs. Each data transferred to PDC buffer, carried on a half-word (16-bit), consists of last converted data right aligned and when TAG is set in the AFEC\_EMR register, data is carried on a word buffer (32-bit) and CHNB field carries the channel number thus allowing an easier post-processing in the PDC buffer or better checking the PDC buffer integrity.

#### 43.6.15 Fault Output

The AFE Controller internal fault output is directly connected to PWM fault input. Fault output may be asserted according to the configuration of AFEC\_EMR (Extended Mode Register) and AFEC\_CWR (Compare Window Register) and converted values. When the Compare occurs, the AFEC fault output generates a pulse of one Master Clock Cycle to the PWM fault input. This fault line can be enabled or disabled within PWM. Should it be activated and asserted by the AFE Controller, the PWM outputs are immediately placed in a safe state (pure combinational path). Note that the AFEC fault output connected to the PWM is not the COMPE bit. Thus the Fault Mode (FMOD) within the PWM configuration must be FMOD = 1.

#### 43.6.16 Write Protection Registers

To prevent any single software error that may corrupt AFEC behavior, certain address spaces can be write-protected by setting the WPEN bit in the “[AFEC Write Protect Mode Register](#)” (AFEC\_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the AFEC Write Protect Status Register (AFEC\_WPSR) is set and the field WPVSR indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the AFEC Write Protect Mode Register (AFEC\_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- “[AFEC Mode Register](#)” on page 1163
- “[AFEC Extended Mode Register](#)” on page 1166
- “[AFEC Channel Sequence 1 Register](#)” on page 1168
- “[AFEC Channel Sequence 2 Register](#)” on page 1169
- “[AFEC Channel Enable Register](#)” on page 1170
- “[AFEC Channel Disable Register](#)” on page 1171
- “[AFEC Compare Window Register](#)” on page 1179
- “[AFEC Channel Gain Register](#)” on page 1180
- “[AFEC Channel Calibration DC Offset Register](#)” on page 1181
- “[AFEC Channel Selection Register](#)” on page 1183
- “[AFEC Channel Offset Compensation Register](#)” on page 1185
- “[AFEC Temperature Sensor Mode Register](#)” on page 1186
- “[AFEC Temperature Compare Window Register](#)” on page 1187

## 43.7 Analog-Front-End Controller (AFEC) User Interface

Any offset not listed in [Table 43-7](#) must be considered as “reserved”.

**Table 43-7. Register Mapping**

Offset	Register	Name	Access	Reset
0x00	Control Register	AFEC_CR	Write-only	–
0x04	Mode Register	AFEC_MR	Read-write	0x00000000
0x08	Extended Mode Register	AFEC_EMR	Read-write	0x00000000
0x0C	Channel Sequence 1 Register	AFEC_SEQ1R	Read-write	0x00000000
0x10	Channel Sequence 2 Register	AFEC_SEQ2R	Read-write	0x00000000
0x14	Channel Enable Register	AFEC_CHER	Write-only	–
0x18	Channel Disable Register	AFEC_CHDR	Write-only	–
0x1C	Channel Status Register	AFEC_CHSR	Read-only	0x00000000
0x20	Last Converted Data Register	AFEC_LCDR	Read-only	0x00000000
0x24	Interrupt Enable Register	AFEC_IER	Write-only	–
0x28	Interrupt Disable Register	AFEC_IDR	Write-only	–
0x2C	Interrupt Mask Register	AFEC_IMR	Read-only	0x00000000
0x30	Interrupt Status Register	AFEC_ISR	Read-only	0x00000000
0x34 - 0x40	Reserved	–	–	–
0x44 - 0x48	Reserved	–	–	–
0x4C	Overrun Status Register	AFEC_OVER	Read-only	0x00000000
0x50	Compare Window Register	AFEC_CWR	Read-write	0x00000000
0x54	Channel Gain Register	AFEC_CGR	Read-write	0x00000000
0x5C	Channel Calibration DC Offset Register	AFEC_CDOR	Read-write	0x00000000
0x60	Channel Differential Register	AFEC_DIFFR	Read-write	0x00000000
0x64	Channel Register Selection	AFEC_CSELR	Read-write	0x00000000
0x68	Channel Data Register	AFEC_CDR	Read-only	0x00000000
0x6C	Channel Offset Compensation Register	AFEC_COOCR	Read-write	0x00000000
0x70	Temperature Sensor Mode Register	AFEC_TEMPMR	Read-write	0x00000000
0x74	Temperature Compare Window Register	AFEC_TEMPCWR	Read-write	0x00000000
0x94	Analog Control Register	AFEC_ACR	Read-write	0x00000100
0x98 - 0xAC	Reserved	–	–	–
0xC4 - 0xE0	Reserved	–	–	–
0xE4	Write Protect Mode Register	AFEC_WPMR	Read-write	0x00000000
0xE8	Write Protect Status Register	AFEC_WPSR	Read-only	0x00000000
0xEC - 0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–
0x100 - 0x124	Reserved for PDC	–	–	–

#### 43.7.1 AFEC Control Register

**Name:** AFEC\_CR

**Address:** 0x400B0000 (0), 0x400B4000 (1)

**Access:** Write-only

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
-	-	-	-	AUTOCAL	-	START	SWRST

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the AFEC simulating a hardware reset.

- **START: Start Conversion**

0 = No effect.

1 = Begins Analog-Front-End conversion.

- **AUTOCAL: Automatic Calibration of AFEC**

0 = No effect.

1 = Launches an automatic calibration of the AFEC cell on the next sequence.

### 43.7.2 AFEC Mode Register

**Name:** AFEC\_MR

**Address:** 0x400B0004 (0), 0x400B4004 (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
USEQ	-	TRANSFER		TRACKTIM			
23	22	21	20	19	18	17	16
ANACH	-	SETTLING		STARTUP			
15	14	13	12	11	10	9	8
PRESCAL							
7	6	5	4	3	2	1	0
FREERUN	FWUP	SLEEP	-	TRGSEL		TRGEN	

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **TRGEN: Trigger Enable**

Value	Name	Description
0	DIS	Hardware triggers are disabled. Starting a conversion is only possible by software.
1	EN	Hardware trigger selected by TRGSEL field is enabled.

- **TRGSEL: Trigger Selection**

Value	Name	Description
0	AFEC_TRIG0	ADTRG pin
1	AFEC_TRIG1	TIO Output of the Timer Counter Channel 0
2	AFEC_TRIG2	TIO Output of the Timer Counter Channel 1
3	AFEC_TRIG3	TIO Output of the Timer Counter Channel 2
4	AFEC_TRIG4	PWM Event Line 0
5	AFEC_TRIG5	PWM Event Line 1
6	AFEC_TRIG6	Reserved
7	-	Reserved

- **SLEEP: Sleep Mode**

Value	Name	Description
0	NORMAL	Normal Mode: The AFEC Core and reference voltage circuitry are kept ON between conversions
1	SLEEP	Sleep Mode: The AFEC Core and reference voltage circuitry are OFF between conversions

- **FWUP: Fast Wake-up**

Value	Name	Description
0	OFF	Normal Sleep Mode: The sleep mode is defined by the SLEEP bit
1	ON	Fast Wake Up Sleep Mode: The Voltage reference is ON between conversions and AFEC Core is OFF

- **FREERUN: Free Run Mode**

Value	Name	Description
0	OFF	Normal Mode
1	ON	Free Run Mode: Never wait for any trigger.

- **PRESCAL: Prescaler Rate Selection**

$$\text{AFEClock} = \text{MCK} / ((\text{PRESCAL}+1) * 2)$$

- **STARTUP: Start-up Time**

Value	Name	Description
0	SUT0	0 periods of AFEClock
1	SUT8	8 periods of AFEClock
2	SUT16	16 periods of AFEClock
3	SUT24	24 periods of AFEClock
4	SUT64	64 periods of AFEClock
5	SUT80	80 periods of AFEClock
6	SUT96	96 periods of AFEClock
7	SUT112	112 periods of AFEClock
8	SUT512	512 periods of AFEClock
9	SUT576	576 periods of AFEClock
10	SUT640	640 periods of AFEClock
11	SUT704	704 periods of AFEClock
12	SUT768	768 periods of AFEClock
13	SUT832	832 periods of AFEClock
14	SUT896	896 periods of AFEClock
15	SUT960	960 periods of AFEClock

- **SETTLING: Analog Settling Time**

Value	Name	Description
0	AST3	3 periods of AFEClock
1	AST5	5 periods of AFEClock
2	AST9	9 periods of AFEClock
3	AST17	17 periods of AFEClock

- **ANACH: Analog Change**

Value	Name	Description
0	NONE	No analog change on channel switching: DIFF0, GAIN0 are used for all channels
1	ALLOWED	Allows different analog settings for each channel. See AFEC_CGR Register.

- **TRACKTIM: Tracking Time**

$$\text{Tracking Time} = (\text{TRACKTIM} + 1) * \text{AFEClock periods.}$$

- **TRANSFER: Transfer Period**

Transfer Period = (TRANSFER \* 2 + 3) AFEClock periods.

- **USEQ: Use Sequence Enable**

Value	Name	Description
0	NUM_ORDER	Normal Mode: The controller converts channels in a simple numeric order.
1	REG_ORDER	User Sequence Mode: The sequence respects what is defined in AFEC_SEQR1 and AFEC_SEQR2 registers.

### 43.7.3 AFEC Extended Mode Register

**Name:** AFEC\_EMR

**Address:** 0x400B0008 (0), 0x400B4008 (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	STM	TAG
23	22	21	20	19	18	17	16
—	—	—	—	—	—	RES	
15	14	13	12	11	10	9	8
—	—	CMPFILTER		—	—	CMPALL	—
7	6	5	4	3	2	1	0
CMPSEL				—	CMPPMODE		

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **CMPPMODE: Comparison Mode**

Value	Name	Description
0	LOW	Generates an event when the converted data is lower than the low threshold of the window.
1	HIGH	Generates an event when the converted data is higher than the high threshold of the window.
2	IN	Generates an event when the converted data is in the comparison window.
3	OUT	Generates an event when the converted data is out of the comparison window.

- **CMPSEL: Comparison Selected Channel**

If CMPALL = 0: CMPSEL indicates which channel has to be compared.

If CMPALL = 1: No effect.

- **CMPALL: Compare All Channels**

0 = Only channel indicated in CMPSEL field is compared.

1 = All channels are compared.

- **CMPFILTER: Compare Event Filtering**

Number of consecutive compare events necessary to raise the flag = CMPFILTER+1

When programmed to 0, the flag rises as soon as an event occurs.

- **RES: Resolution**

Value	Name	Description
0	NO_AVERAGE	12-bit resolution, AFEC sample rate is maximum (no averaging).
1	LOW_RES	10-bit resolution, AFEC sample rate is maximum (no averaging).
2	OSR4	13-bit resolution, AFEC sample rate divided by 4 (averaging).
3	OSR16	14-bit resolution, AFEC sample rate divided by 16 (averaging).
4	OSR64	15-bit resolution, AFEC sample rate divided by 64 (averaging).
5	OSR256	16-bit resolution, AFEC sample rate divided by 256 (averaging).

- **TAG: TAG of the AFEC\_LDCR Register**

0 = Sets CHNB to zero in AFEC\_LDCR

1 = Appends the channel number to the conversion result in AFEC\_LDCR register

- **STM: Single Trigger Mode**

0 = Multiple triggers are required to get an averaged result

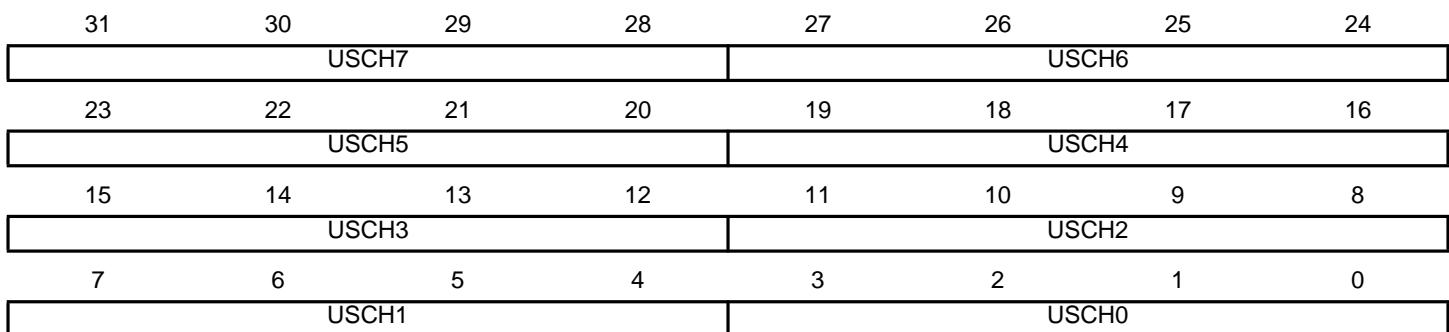
1 = Only a Single Trigger is required to get an averaged value

#### 43.7.4 AFEC Channel Sequence 1 Register

**Name:** AFEC\_SEQ1R

**Address:** 0x400B000C (0), 0x400B400C (1)

**Access:** Read-write



This register can only be written if the WPEN bit is cleared in “[AFEC Write Protect Mode Register](#)” on page 1189.

- **USCHx: User Sequence Number x**

The sequence number x (USCHx) can be programmed by the Channel number CHy where y is the value written in this field. The allowed range is 0 up to 15. So it is only possible to use the sequencer from CH0 to CH15.

This register activates only if AFEC\_MR(USEQ) field is set to ‘1’.

Any USCHx field is taken into account only if the AFEC\_CHSR(CHx) register field reads logical ‘1’ else any value written in USCHx does not add the corresponding channel in the conversion sequence.

Configuring the same value in different fields leads to multiple samples of the same channel during the conversion sequence. This can be done consecutively, or not, according to user needs.

#### 43.7.5 AFEC Channel Sequence 2 Register

**Name:** AFEC\_SEQ2R

**Address:** 0x400B0010 (0), 0x400B4010 (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
USCH15				USCH14			
23	22	21	20	19	18	17	16
USCH13				USCH12			
15	14	13	12	11	10	9	8
USCH11				USCH10			
7	6	5	4	3	2	1	0
USCH9				USCH8			

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **USCHx: User Sequence Number x**

The sequence number x (USCHx) can be programmed by the Channel number CHy where y is the value written in this field. The allowed range is 0 up to 15. So it is only possible to use the sequencer from CH0 to CH15.

This register activates only if AFEC\_MR(USEQ) field is set to ‘1’.

Any USCHx field is taken into account only if the AFEC\_CHSR(CHx) register field reads logical ‘1’ else any value written in USCHx does not add the corresponding channel in the conversion sequence.

Configuring the same value in different fields leads to multiple samples of the same channel during the conversion sequence. This can be done consecutively, or not, according to user needs.

#### 43.7.6 AFEC Channel Enable Register

**Name:** AFEC\_CHER

**Address:** 0x400B0014 (0), 0x400B4014 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **CHx: Channel x Enable**

0 = No effect

1 = Enables the corresponding channel

Note: If USEQ = 1 in the AFEC\_MR register, CHx corresponds to the xth channel of the sequence described in AFEC\_SEQ1R, AFEC\_SEQ2R.

### 43.7.7 AFEC Channel Disable Register

**Name:** AFEC\_CHDR

**Address:** 0x400B0018 (0), 0x400B4018 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **CHx: Channel x Disable**

0 = No effect.

1 = Disables the corresponding channel.

**Warning:** If the corresponding channel is disabled during a conversion or if it is disabled then reenabled during a conversion, its associated data and its corresponding EOC and OVRE flags in AFEC\_SR are unpredictable.

#### 43.7.8 AFEC Channel Status Register

**Name:** AFEC\_CHSR

**Address:** 0x400B001C (0), 0x400B401C (1)

**Access:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

- **CHx: Channel x Status**

0 = The corresponding channel is disabled.

1 = The corresponding channel is enabled.

#### 43.7.9 AFEC Last Converted Data Register

**Name:** AFEC\_LCDR

**Address:** 0x400B0020 (0), 0x400B4020 (1)

**Access:** Read-only

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

- **L DATA: Last Data Converted**

The Analog-Front-End conversion data is placed into this register at the end of a conversion and remains until a new conversion is completed.

- **CHNB: Channel Number**

Indicates the last converted channel when the TAG option is set to 1 in the AFEC\_EMR register. If TAG option is not set, CHNB = 0.

#### 43.7.10 AFEC Interrupt Enable Register

**Name:** AFEC\_IER

**Address:** 0x400B0024 (0), 0x400B4024 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
EOCAL	TEMPCHG	-	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
EOC15	EOC14	EOC13	EOC12	EOC11	EOC10	EOC9	EOC8
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx: End of Conversion Interrupt Enable x**
- **DRDY: Data Ready Interrupt Enable**
- **GOVRE: General Overrun Error Interrupt Enable**
- **COMPE: Comparison Event Interrupt Enable+**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TEMPCHG: Temperature Change Interrupt Enable**
- **EOCAL: End of Calibration Sequence Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

#### 43.7.11 AFEC Interrupt Disable Register

**Name:** AFEC\_IDR

**Address:** 0x400B0028 (0), 0x400B4028 (1)

**Access:** Write-only

31	30	29	28	27	26	25	24
EOCAL	TEMPCHG	-	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
EOC15	EOC14	EOC13	EOC12	EOC11	EOC10	EOC9	EOC8
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx:** End of Conversion Interrupt Disable x
- **DRDY:** Data Ready Interrupt Disable
- **GOVRE:** General Overrun Error Interrupt Disable
- **COMPE:** Comparison Event Interrupt Disable
- **ENDRX:** End of Receive Buffer Interrupt Disable
- **RXBUFF:** Receive Buffer Full Interrupt Disable
- **TEMPCHG:** Temperature Change Interrupt Disable
- **EOCAL:** End of Calibration Sequence Interrupt Disable

0 = No effect.

1 = Disables the corresponding interrupt.

#### 43.7.12 AFEC Interrupt Mask Register

**Name:** AFEC\_IMR

**Address:** 0x400B002C (0), 0x400B402C (1)

**Access:** Read-only

31	30	29	28	27	26	25	24
EOCAL	TEMPCHG	-	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
EOC15	EOC14	EOC13	EOC12	EOC11	EOC10	EOC9	EOC8
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx: End of Conversion Interrupt Mask x**
- **DRDY: Data Ready Interrupt Mask**
- **GOVRE: General Overrun Error Interrupt Mask**
- **COMPE: Comparison Event Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TEMPCHG: Temperature Change Interrupt Mask**
- **EOCAL: End of Calibration Sequence Interrupt Mask**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

### 43.7.13 AFEC Interrupt Status Register

**Name:** AFEC\_ISR

**Address:** 0x400B0030 (0), 0x400B4030 (1)

**Access:** Read-only

31	30	29	28	27	26	25	24
EOCAL	TEMPCHG	-	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
EOC15	EOC14	EOC13	EOC12	EOC11	EOC10	EOC9	EOC8
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx: End of Conversion x**

0 = The corresponding analog channel is disabled, or the conversion is not finished. This flag is cleared when reading the AFEC\_CDR register with the AFEC\_CSEL register programmed with CSEL=x.

1 = The corresponding analog channel is enabled and conversion is complete.

- **TEMPCHG: Temperature Change**

0 = There is no comparison match (defined in the AFEC\_TEMPCMPR register) since the last read of AFEC\_ISR register.

1 = The temperature value reported on AFEC\_CDR (AFEC\_CSEL.CSEL=15) has changed since the last read of AFEC\_ISR according to what is defined in the temperature mode register (AFEC\_TEMPMR) and the temperature compare window register (AFEC\_TEMPCWR).

- **DRDY: Data Ready**

0 = No data has been converted since the last read of AFEC\_LCDR.

1 = At least one data has been converted and is available in AFEC\_LCDR.

- **GOVRE: General Overrun Error**

0 = No General Overrun Error occurred since the last read of AFEC\_ISR.

1 = At least one General Overrun Error has occurred since the last read of AFEC\_ISR.

- **COMPE: Comparison Error**

0 = No Comparison Error since the last read of AFEC\_ISR.

1 = At least one Comparison Error has occurred since the last read of AFEC\_ISR.

- **ENDRX: End of RX Buffer**

0 = The Receive Counter Register has not reached 0 since the last write in AFEC\_RCR or AFEC\_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in AFEC\_RCR or AFEC\_RNCR.

- **RXBUFF: RX Buffer Full**

0 = AFEC\_RCR or AFEC\_RNCR have a value other than 0.

1 = Both AFEC\_RCR and AFEC\_RNCR have a value of 0.

- **EOCAL: End of Calibration Sequence**

0 = Calibration sequence is on going, or no calibration sequence has been requested.

1 = Calibration sequence is complete.

#### 43.7.14 AFEC Overrun Status Register

**Name:** AFEC\_OVER

**Address:** 0x400B004C (0), 0x400B404C (1)

**Access:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
OVRE15	OVRE14	OVRE13	OVRE12	OVRE11	OVRE10	OVRE9	OVRE8
7	6	5	4	3	2	1	0
OVRE7	OVRE6	OVRE5	OVRE4	OVRE3	OVRE2	OVRE1	OVRE0

- **OVRE<sub>x</sub>: Overrun Error x**

0 = No overrun error on the corresponding channel since the last read of AFEC\_OVER.

1 = There has been an overrun error on the corresponding channel since the last read of AFEC\_OVER.

#### 43.7.15 AFEC Compare Window Register

**Name:** AFEC\_CWR

**Address:** 0x400B0050 (0), 0x400B4050 (1)

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **LOWTHRES: Low Threshold**

Low threshold associated to compare settings of the AFEC\_EMR register.

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of the AFEC\_EMR register.

#### 43.7.16 AFEC Channel Gain Register

**Name:** AFEC\_CGR

**Address:** 0x400B0054 (0), 0x400B4054 (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
GAIN15		GAIN14		GAIN13		GAIN12	
23	22	21	20	19	18	17	16
GAIN11		GAIN10		GAIN9		GAIN8	
15	14	13	12	11	10	9	8
GAIN7		GAIN6		GAIN5		GAIN4	
7	6	5	4	3	2	1	0
GAIN3		GAIN2		GAIN1		GAIN0	

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **GAINx: Gain for channel x**

Gain applied on input of Analog-Front-End.

GAINx		Gain applied when DIFFx = 0	Gain applied when DIFFx = 1
0	0	1	0.5
0	1	1	1
1	0	2	2
1	1	4	2

The DIFFx mentioned in this table is described in “AFEC Channel Differential Register” on page 1182.

#### 43.7.17 AFEC Channel Calibration DC Offset Register

**Name:** AFEC\_CDOR

**Address:** 0x400B005C (0), 0x400B405C (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
OFF15	OFF14	OFF13	OFF12	OFF11	OFF10	OFF9	OFF8
7	6	5	4	3	2	1	0
OFF7	OFF6	OFF5	OFF4	OFF3	OFF2	OFF1	OFF0

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **OFFx: Offset for Channel x, used in Automatic Calibration Procedure**

0 = No Offset.

1 = Centers the analog signal on Vrefin/2 before the gain scaling. The applied Offset is:  $(G-1)V_{refin}/2$

where G is the applied gain (see the description of the AFEC\_CGR register).

Note: When a channel requires calibration, the corresponding OFF bit must be configured to 1 prior to launch of the automatic calibration.

#### 43.7.18 AFEC Channel Differential Register

**Name:** AFEC\_DIFFR

**Address:** 0x400B0060 (0), 0x400B4060 (1)

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—
23	22	21	20	19	18	17	16
—	—	—	—	—	—	—	—
15	14	13	12	11	10	9	8
DIFF15	DIFF14	DIFF13	DIFF12	DIFF11	DIFF10	DIFF9	DIFF8
7	6	5	4	3	2	1	0
DIFF7	DIFF6	DIFF5	DIFF4	DIFF3	DIFF2	DIFF1	DIFF0

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **DIFFx: Differential inputs for channel x**

0 = Single Ended Mode.

1 = Fully Differential Mode.

#### 43.7.19 AFEC Channel Selection Register

**Name:** AFEC\_CSELR

**Address:** 0x400B0064 (0), 0x400B4064 (1)

**Access:** Read-write

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
-	-	-				CSEL	

- **CSEL: Channel Selection**

0-15: Selects the channel to be displayed in the AFEC\_CDR and AFEC\_CAOR Register. To be filled with the appropriate channel number.

#### 43.7.20 AFEC Channel Data Register

**Name:** AFEC\_CDR

**Address:** 0x400B0068 (0), 0x400B4068 (1)

**Access:** Read-only

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

- **DATA: Converted Data**

Returns the Analog-Front-End conversion data corresponding to channel CSEL (configured in the “AFEC Channel Selection Register” Register).

At the end of a conversion, the converted data is loaded into one of the 16 internal registers (one for each channel) and remains in this internal register until a new conversion is completed on the same channel index. The AFEC\_CDR together with AFEC\_CSELR registers allows to multiplex all the internal channel data registers.

The data carried on AFEC\_CDR is valid only if AFEC\_CHSR[CSEL] is set to 1, CSEL being programmed in AFEC\_CSELR Register.

#### 43.7.21 AFEC Channel Offset Compensation Register

**Name:** AFEC\_COOR

**Address:** 0x400B006C (0), 0x400B406C (1)

**Access:** Read-write

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

- **AOFF: Analog Offset**

Defines the analog offset to be used for channel CSEL (configured in the “AFEC Channel Selection Register” ). This value is used as an input value for the DAC included in the AFEC.

### 43.7.22 AFEC Temperature Sensor Mode Register

**Name:** AFEC\_TEMPMR

**Address:** 0x400B0070 (0), 0x400B4070 (1)

**Access:** Read-write

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
–	–	TEMPCMPMOD	–	–	–	–	RTCT

- **RTCT: Temperature Sensor RTC Trigger mode**

0 = The temperature sensor measure is not triggered by RTC event.

1 = The temperature sensor measure is triggered by RTC event (if TRGEN=1).

- **TEMPCMPMOD: Temperature Comparison Mode**

Value	Name	Description
0	LOW	Generates an event when the converted data is lower than the low threshold of the window.
1	HIGH	Generates an event when the converted data is higher than the high threshold of the window.
2	IN	Generates an event when the converted data is in the comparison window.
3	OUT	Generates an event when the converted data is out of the comparison window.

### 43.7.23 AFEC Temperature Compare Window Register

**Name:** AFEC\_TEMPCWR

**Address:** 0x400B0074 (0), 0x400B4074 (1)

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **TLOWTHRES: Temperature Low Threshold**

Low threshold associated to compare settings of the AFEC\_TEMPMR register.

- **THIGHTHRES: Temperature High Threshold**

High threshold associated to compare settings of the AFEC\_TEMPMR register.

#### 43.7.24 AFEC Analog Control Register

**Name:** AFEC\_ACR

**Address:** 0x400B0094 (0), 0x400B4094 (1)

**Access:** Read-write

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

This register can only be written if the WPEN bit is cleared in “AFEC Write Protect Mode Register” on page 1189.

- **IBCTL: AFEC Bias Current Control**

Allows to adapt performance versus power consumption. (See the product electrical characteristics for further details.)

### 43.7.25 AFEC Write Protect Mode Register

**Name:** AFEC\_WPMR

**Address:** 0x400B00E4 (0), 0x400B40E4 (1)

**Access:** Read-write

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

Protects the registers:

[“AFEC Mode Register” on page 1163](#)

[“AFEC Extended Mode Register” on page 1166](#)

[“AFEC Channel Sequence 1 Register” on page 1168](#)

[“AFEC Channel Sequence 2 Register” on page 1169](#)

[“AFEC Channel Enable Register” on page 1170](#)

[“AFEC Channel Disable Register” on page 1171](#)

[“AFEC Compare Window Register” on page 1179](#)

[“AFEC Channel Gain Register” on page 1180](#)

[“AFEC Channel Calibration DC Offset Register” on page 1181](#)

[“AFEC Channel Differential Register” on page 1182](#)

[“AFEC Channel Selection Register” on page 1183](#)

[“AFEC Channel Offset Compensation Register” on page 1185](#)

[“AFEC Temperature Sensor Mode Register” on page 1186](#)

[“AFEC Temperature Compare Window Register” on page 1187](#)

[“AFEC Analog Control Register” on page 1188](#)

- **WPKEY: Write Protect KEY**

Value	Name	Description
0x414443	ADC	Should be written at value 0x414443 (“ADC” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

#### 43.7.26 AFEC Write Protect Status Register

**Name:** AFEC\_WPSR

**Address:** 0x400B00E8 (0), 0x400B40E8 (1)

**Access:** Read-only

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

- **WPVS: Write Protect Violation Status**

0 = No Write Protect Violation has occurred since the last read of the AFEC\_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the AFEC\_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Reading AFEC\_WPSR automatically clears all fields.

## 44. Ethernet MAC (GMAC)

### 44.1 Description

The Ethernet MAC (GMAC) module implements a 10/100 Mbps Ethernet MAC compatible with the IEEE 802.3 standard. The GMAC can operate in either half or full duplex mode at all supported speeds. The “[Network Configuration Register](#)” is used to select the speed, duplex mode and interface type (MII).

The GMAC comprises two constituent components:

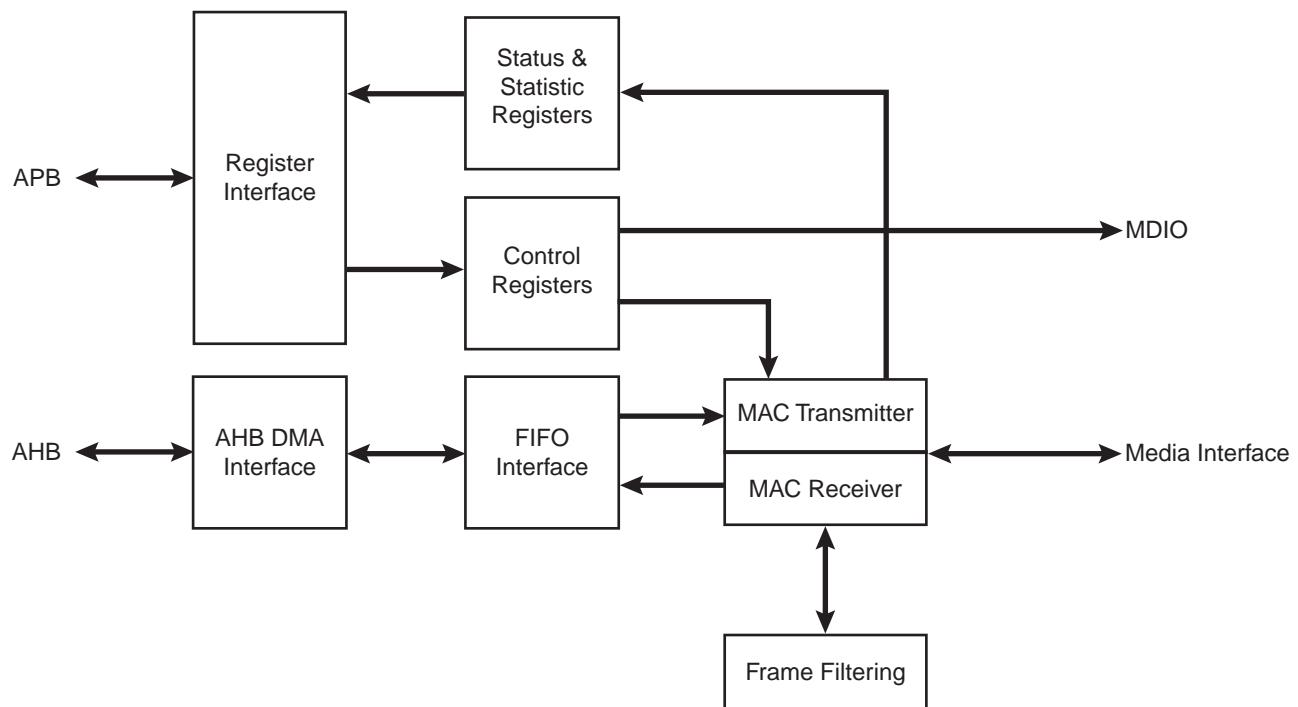
- GEM\_MAC controlling transmit, receive, address checking and loopback
- GEM\_TSU calculates the IEEE 1588 timer values

### 44.2 Embedded Characteristics

- Compatible with IEEE Standard 802.3
- 10, 100 Mbps operation
- Full and half duplex operation at all supported speeds of operation
- Statistics Counter Registers for RMON/MIB
- MII interface to the physical layer
- Integrated physical coding
- Direct memory access (DMA) interface to external memory
- Programmable burst length and endianism for DMA
- Interrupt generation to signal receive and transmit completion, or errors
- Automatic pad and cyclic redundancy check (CRC) generation on transmitted frames
- Automatic discard of frames received with errors
- Receive and transmit IP, TCP and UDP checksum offload. Both IPv4 and IPv6 packet types supported
- Address checking logic for four specific 48-bit addresses, four type IDs, promiscuous mode, hash matching of unicast and multicast destination addresses and Wake-on-LAN
- Management Data Input/Output (MDIO) interface for physical layer management
- Support for jumbo frames up to 10240 bytes
- Full duplex flow control with recognition of incoming pause frames and hardware generation of transmitted pause frames
- Half duplex flow control by forcing collisions on incoming frames
- Support for 802.1Q VLAN tagging with recognition of incoming VLAN and priority tagged frames
- Support for 802.1Qbb priority-based flow control
- Programmable Inter Packet Gap (IPG) Stretch
- Recognition of IEEE 1588 PTP frames
- IEEE 1588 time stamp unit (TSU)
- Support for 802.1AS timing and synchronization

### 44.3 Block Diagram

Figure 44-1. Block Diagram



## 44.4 Signal Interface

The GMAC includes the following signal interfaces

- MII to an external PHY
- MDIO interface for external PHY management
- Slave APB interface for accessing GMAC registers
- Master AHB interface for memory access

**Table 44-1. GMAC connections in the different modes**

Signal Name	Function	MII
GTXCK	Transmit Clock or Reference Clock	TXCK
GTXEN	Transmit Enable	TXEN
GTX[3..0]	Transmit Data	TXD[3:0]
GTXER	Transmit Coding Error	TXER
GRXCK	Receive Clock	RXCK
GRXDV	Receive Data Valid	RXDV
GRX[3..0]	Receive Data	RXD[3:0]
GRXER	Receive Error	RXER
GCRS	Carrier Sense and Data Valid	CRS
GCOL	Collision Detect	COL
GMDC	Management Data Clock	MDC
GMDIO	Management Data Input/Output	MDIO

## 44.5 Functional Description

### 44.5.1 Media Access Controller

The Media Access Controller (MAC) transmit block takes data from FIFO, adds preamble and, if necessary, pad and frame check sequence (FCS). Both half duplex and full duplex Ethernet modes of operation are supported. When operating in half duplex mode, the MAC transmit block generates data according to the carrier sense multiple access with collision detect (CSMA/CD) protocol. The start of transmission is deferred if carrier sense (CRS) is active. If collision (COL) becomes active during transmission, a jam sequence is asserted and the transmission is retried after a random back off. The CRS and COL signals have no effect in full duplex mode.

The MAC receive block checks for valid preamble, FCS, alignment and length, and presents received frames to the MAC address checking block and FIFO. Software can configure the GMAC to receive jumbo frames up to 10240 bytes. It can optionally strip CRC from the received frame prior to transfer to FIFO.

The address checker recognizes four specific 48-bit addresses, can recognize four different type ID values, and contains a 64-bit Hash Register for matching multicast and unicast addresses as required. It can recognize the broadcast address of all ones and copy all frames. The MAC can also reject all frames that are not VLAN tagged and recognise Wake on LAN events.

The MAC receive block supports offloading of IP, TCP and UDP checksum calculations (both IPv4 and IPv6 packet types supported), and can automatically discard bad checksum frames.

### 44.5.2 1588 Time Stamp Unit

The 1588 time stamp unit (TSU) is a timer implemented as a 62-bit register. The 32 upper bits count seconds and the 30 lower bits count nanoseconds. The 30 lower bits roll over when they have counted to one second. The timer increments by a programmable number of nanoseconds with each MCK period and can be adjusted (incremented or decremented) through APB register accesses.

### 44.5.3 AHB Direct Memory Access Interface

GMAC is supplied with an AHB DMA interface. When the GMAC is configured to use the DMA, it is attached to the MAC module's FIFO to provide a scatter gather type capability for packet data storage in the embedded processor system or System on Chip.

The GMAC DMA controller performs six types of operation on the AHB bus. When the GMAC DMA is configured in internal FIFO mode, in order of priority these are as follows:

- Receive buffer manager write/read
- Transmit buffer manager write/read
- Receive data DMA write
- Transmit data DMA read

#### 44.5.3.1 Receive AHB Buffers

Received frames, optionally including FCS, are written to receive AHB buffers stored in memory. The receive buffer depth is programmable in the range of 64 bytes to 16 Kbytes through the DMA Configuration Register, with the default being 128 bytes.

The start location for each receive AHB buffer is stored in memory in a list of receive buffer descriptors at an address location pointed to by the receive buffer queue pointer. The base address for the receive buffer queue pointer is configured in software using the Receive Buffer Queue Base Address Register.

Each list entry consists of two words. The first is the address of the receive AHB buffer and the second the receive status. If the length of a receive frame exceeds the AHB buffer length, the status word for the used buffer is written with zeroes except for the "start of frame" bit, which is always set for the first buffer in a frame. Bit zero of the address field is written to 1 to show the buffer has been used. The receive buffer manager then reads the location of the next receive AHB buffer and fills that with the next part of the received frame data. AHB buffers are filled until the frame is complete and the final

buffer descriptor status word contains the complete frame status. Refer to [Table 44-2, “Receive Buffer Descriptor Entry”](#) for details of the receive buffer descriptor list.

Each receive AHB buffer start location is a word address. The start of the first AHB buffer in a frame can be offset by up to three bytes, depending on the value written to bits 14 and 15 of the Network Configuration Register. If the start location of the AHB buffer is offset, the available length of the first AHB buffer is reduced by the corresponding number of bytes.

**Table 44-2. Receive Buffer Descriptor Entry**

Bit	Function
<b>Word 0</b>	
31:2	Address of beginning of buffer
1	Wrap - marks last descriptor in receive buffer descriptor list.
0	Ownership - needs to be zero for the GMAC to write data to the receive buffer. The GMAC sets this to one once it has successfully written a frame to memory. Software has to clear this bit before the buffer can be used again.
<b>Word 1</b>	
31	Global all ones broadcast address detected
30	Multicast hash match
29	Unicast hash match
28	–
27	Specific Address Register match found, bit 25 and bit 26 indicate which Specific Address Register causes the match.
26:25	Specific Address Register match. Encoded as follows: 00 - Specific Address Register 1 match 01 - Specific Address Register 2 match 10 - Specific Address Register 3 match 11 - Specific Address Register 4 match If more than one specific address is matched only one is indicated with priority 4 down to 1.
24	This bit has a different meaning depending on whether RX checksum offloading is enabled. <b>With RX checksum offloading disabled:</b> (bit 24 clear in Network Configuration Register) Type ID register match found, bit 22 and bit 23 indicate which type ID register causes the match. <b>With RX checksum offloading enabled:</b> (bit 24 set in Network Configuration Register) 0 - the frame was not SNAP encoded and/or had a VLAN tag with the Canonical Format Indicator (CFI) bit set. 1 - the frame was SNAP encoded and had either no VLAN tag or a VLAN tag with the CFI bit not set.

**Table 44-2. Receive Buffer Descriptor Entry (Continued)**

Bit	Function
23:22	<p>This bit has a different meaning depending on whether RX checksum offloading is enabled.</p> <p><b>With RX checksum offloading disabled:</b> (bit 24 clear in Network Configuration) Type ID register match. Encoded as follows:</p> <ul style="list-style-type: none"> <li>00 - Type ID register 1 match</li> <li>01 - Type ID register 2 match</li> <li>10 -Type ID register 3 match</li> <li>11 - Type ID register 4 match</li> </ul> <p>If more than one Type ID is matched only one is indicated with priority 4 down to 1.</p> <p><b>With RX checksum offloading enabled:</b> (bit 24 set in Network Configuration Register)</p> <ul style="list-style-type: none"> <li>00 - Neither the IP header checksum nor the TCP/UDP checksum was checked.</li> <li>01 - The IP header checksum was checked and was correct. Neither the TCP nor UDP checksum was checked.</li> <li>10 - Both the IP header and TCP checksum were checked and were correct.</li> <li>11 - Both the IP header and UDP checksum were checked and were correct.</li> </ul>
21	VLAN tag detected - type ID of 0x8100. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a type ID of 0x8100
20	Priority tag detected - type ID of 0x8100 and null VLAN identifier. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a type ID of 0x8100 and a null VLAN identifier.
19:17	VLAN priority - only valid if bit 21 is set.
16	Canonical format indicator (CFI) bit (only valid if bit 21 is set).
15	End of frame - when set the buffer contains the end of a frame. If end of frame is not set, then the only valid status bit is start of frame (bit 14).
14	Start of frame - when set the buffer contains the start of a frame. If both bits 15 and 14 are set, the buffer contains a whole frame.
13	<p>This bit has a different meaning depending on whether jumbo frames and ignore FCS modes are enabled. If neither mode is enabled this bit will be zero.</p> <p><b>With jumbo frame mode enabled:</b> (bit 3 set in Network Configuration Register) Additional bit for length of frame (bit[13]), that is concatenated with bits[12:0]</p> <p><b>With ignore FCS mode enabled and jumbo frames disabled:</b> (bit 26 set in Network Configuration Register and bit 3 clear in Network Configuration Register) This indicates per frame FCS status as follows:</p> <ul style="list-style-type: none"> <li>0 - Frame had good FCS</li> <li>1 - Frame had bad FCS, but was copied to memory as ignore FCS enabled.</li> </ul>
12:0	<p>These bits represent the length of the received frame which may or may not include FCS depending on whether FCS discard mode is enabled.</p> <p><b>With FCS discard mode disabled:</b> (bit 17 clear in Network Configuration Register) Least significant 12 bits for length of frame including FCS. If jumbo frames are enabled, these 12 bits are concatenated with bit[13] of the descriptor above.</p> <p><b>With FCS discard mode enabled:</b> (bit 17 set in Network Configuration Register) Least significant 12 bits for length of frame excluding FCS. If jumbo frames are enabled, these 12 bits are concatenated with bit[13] of the descriptor above.</p>

To receive frames, the AHB buffer descriptors must be initialized by writing an appropriate address to bits 31:2 in the first word of each list entry. Bit 0 must be written with zero. Bit 1 is the wrap bit and indicates the last entry in the buffer descriptor list.

The start location of the receive buffer descriptor list must be written with the receive buffer queue base address before reception is enabled (receive enable in the Network Control Register). Once reception is enabled, any writes to the Receive Buffer Queue Base Address Register are ignored. When read, it will return the current pointer position in the descriptor list, though this is only valid and stable when receive is disabled.

If the filter block indicates that a frame should be copied to memory, the receive data DMA operation starts writing data into the receive buffer. If an error occurs, the buffer is recovered.

An internal counter within the GMAC represents the receive buffer queue pointer and it is not visible through the CPU interface. The receive buffer queue pointer increments by two words after each buffer has been used. It re-initializes to the receive buffer queue base address if any descriptor has its wrap bit set.

As receive AHB buffers are used, the receive AHB buffer manager sets bit zero of the first word of the descriptor to logic one indicating the AHB buffer has been used.

Software should search through the “used” bits in the AHB buffer descriptors to find out how many frames have been received, checking the start of frame and end of frame bits.

For a properly working 10/100 Ethernet system there should be no excessive length frames or frames greater than 128 bytes with CRC errors. Collision fragments will be less than 128 bytes long, therefore it will be a rare occurrence to find a frame fragment in a receive AHB buffer, when using the default value of 128 bytes for the receive buffers size.

If bit zero of the receive buffer descriptor is already set when the receive buffer manager reads the location of the receive AHB buffer, then the buffer has been already used and cannot be used again until software has processed the frame and cleared bit zero. In this case, the “buffer not available” bit in the receive status register is set and an interrupt triggered. The receive resource error statistics register is also incremented.

#### 44.5.3.2 Transmit AHB Buffers

Frames to transmit are stored in one or more transmit AHB buffers. Transmit frames can be between 1 and 16384 bytes long, so it is possible to transmit frames longer than the maximum length specified in the IEEE 802.3 standard. It should be noted that zero length AHB buffers are allowed and that the maximum number of buffers permitted for each transmit frame is 128.

The start location for each transmit AHB buffer is stored in memory in a list of transmit buffer descriptors at a location pointed to by the transmit buffer queue pointer. The base address for this queue pointer is set in software using the Transmit Buffer Queue Base Address Register. Each list entry consists of two words. The first is the byte address of the transmit buffer and the second containing the transmit control and status. For the FIFO-based DMA configured with a 32-bit data path the address of the buffer is a byte address.

Frames can be transmitted with or without automatic CRC generation. If CRC is automatically generated, pad will also be automatically generated to take frames to a minimum length of 64 bytes. When CRC is not automatically generated (as defined in word 1 of the transmit buffer descriptor or through the control bus of FIFO), the frame is assumed to be at least 64 bytes long and pad is not generated.

An entry in the transmit buffer descriptor list is described in [Table 44-3, “Transmit Buffer Descriptor Entry”](#).

To transmit frames, the buffer descriptors must be initialized by writing an appropriate byte address to bits [31:0] in the first word of each descriptor list entry.

The second word of the transmit buffer descriptor is initialized with control information that indicates the length of the frame, whether or not the MAC is to append CRC and whether the buffer is the last buffer in the frame.

After transmission the status bits are written back to the second word of the first buffer along with the used bit. Bit 31 is the used bit which must be zero when the control word is read if transmission is to take place. It is written to one once the frame has been transmitted. Bits[29:20] indicate various transmit error conditions. Bit 30 is the wrap bit which can be set for any buffer within a frame. If no wrap bit is encountered the queue pointer continues to increment.

The Transmit Buffer Queue Base Address Register can only be updated while transmission is disabled or halted; otherwise any attempted write will be ignored. When transmission is halted the transmit buffer queue pointer will maintain its value. Therefore when transmission is restarted the next descriptor read from the queue will be from immediately after the last successfully transmitted frame. While transmit is disabled (bit 3 of the network control set low), the transmit buffer queue pointer resets to point to the address indicated by the Transmit Buffer Queue Base Address Register. Note that disabling receive does not have the same effect on the receive buffer queue pointer.

Once the transmit queue is initialized, transmit is activated by writing to the transmit start bit (bit 9) of the Network Control Register. Transmit is halted when a buffer descriptor with its used bit set is read, a transmit error occurs, or by writing to the transmit halt bit of the Network Control Register. Transmission is suspended if a pause frame is received while the pause enable bit is set in the Network Configuration Register. Rewriting the start bit while transmission is active is allowed. This is implemented with TXGO variable which is readable in the Transmit Status Register at bit location 3. The TXGO variable is reset when:

- Transmit is disabled.
- A buffer descriptor with its ownership bit set is read.
- Bit 10, THALT, of the Network Control Register is written.
- There is a transmit error such as too many retries or a transmit under run.

To set TXGO, write TSTART to the bit 9 of the Network Control Register. Transmit halt does not take effect until any ongoing transmit finishes.

If the DMA is configured for internal FIFO mode, transmission will automatically restart from the first buffer of the frame.

If a used bit is read mid way through transmission of a multi buffer frame this is treated as a transmit error. Transmission stops, GTXER is asserted and the FCS will be bad.

If transmission stops due to a transmit error or a used bit being read, transmission will restart from the first buffer descriptor of the frame being transmitted when the transmit start bit is rewritten.

**Table 44-3. Transmit Buffer Descriptor Entry**

Bit	Function
<b>Word 0</b>	
31:0	Byte address of buffer
<b>Word 1</b>	
31	Used - must be zero for the GMAC to read data to the transmit buffer. The GMAC sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap - marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29	Retry limit exceeded, transmit error detected
28	Transmit under run - occurs when the start of packet data has been written into the FIFO and either HRESP is not OK, or the transmit data could not be fetched in time, or when buffers are exhausted.
27	Transmit frame corruption due to AHB error - set if an error occurs while midway through reading transmit frame from the AHB, including HRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and GTXER asserted).
26	Late collision, transmit error detected.
25:23	Reserved

**Table 44-3. Transmit Buffer Descriptor Entry (Continued)**

<b>Bit</b>	<b>Function</b>
22:20	<p>Transmit IP/TCP/UDP checksum generation offload errors:</p> <ul style="list-style-type: none"> <li>000 No Error.</li> <li>001 The Packet was identified as a VLAN type, but the header was not fully complete, or had an error in it.</li> <li>010 The Packet was identified as a SNAP type, but the header was not fully complete, or had an error in it.</li> <li>011 The Packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/IPv6.</li> <li>100 The Packet was not identified as VLAN, SNAP or IP.</li> <li>101 Non supported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted.</li> <li>110 Packet type detected was not TCP or UDP. TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted.</li> <li>111 A premature end of packet was detected and the TCP/UDP checksum could not be generated.</li> </ul>
19:17	Reserved
16	<p>No CRC to be appended by MAC. When set, this implies that the data in the buffers already contains a valid CRC, hence no CRC or padding is to be appended to the current frame by the MAC.</p> <p>This control bit must be set for the first buffer in a frame and will be ignored for the subsequent buffers of a frame.</p> <p>Note that this bit must be clear when using the transmit IP/TCP/UDP checksum generation offload, otherwise checksum generation and substitution will not occur.</p>
15	Last buffer, when set this bit will indicate the last buffer in the current frame has been reached.
14	Reserved
13:0	Length of buffer

#### 44.5.3.3 DMA Bursting on the AHB

The DMA will always use SINGLE, or INCR type AHB accesses for buffer management operations. When performing data transfers, the AHB burst length used can be programmed using bits [4:0] of the DMA Configuration Register so that either SINGLE, INCR or fixed length incrementing bursts (INCR4, INCR8 or INCR16) are used where possible.

When there is enough space and enough data to be transferred, the programmed fixed length bursts will be used. If there is not enough data or space available, for example when at the beginning or the end of a buffer, SINGLE type accesses are used. Also SINGLE type accesses are used at 1024 byte boundaries, so that the 1 Kbytes boundaries are not burst over as per AHB requirements.

The DMA will not terminate a fixed length burst early, unless an error condition occurs on the AHB or if receive or transmit are disabled in the Network Control Register.

#### 44.5.4 MAC Transmit Block

The MAC transmitter can operate in either half duplex or full duplex mode and transmits frames in accordance with the Ethernet IEEE 802.3 standard. In half duplex mode, the CSMA/CD protocol of the IEEE 802.3 specification is followed.

A small input buffer receives data through FIFO which, depending on the DMA bus width control bits in the Network Configuration Register, will extract data in 32-bit form. All subsequent processing prior to the final output is performed in bytes.

Transmit data can be output using the MII interface.

Frame assembly starts by adding preamble and the start frame delimiter. Data is taken from FIFO a word at a time.

If necessary, padding is added to take the frame length to 60 bytes. CRC is calculated using an order 32-bit polynomial. This is inverted and appended to the end of the frame taking the frame length to a minimum of 64 bytes. If the no CRC bit is set in the second word of the last buffer descriptor of a transmit frame, neither pad nor CRC are appended. The no CRC bit can also be set through FIFO.

In full duplex mode (at all data rates), frames are transmitted immediately. Back to back frames are transmitted at least 96 bit times apart to guarantee the interframe gap.

In half duplex mode, the transmitter checks carrier sense. If asserted, the transmitter waits for the signal to become inactive, and then starts transmission after the interframe gap of 96 bit times. If the collision signal is asserted during transmission, the transmitter will transmit a jam sequence of 32 bits taken from the data register and then retry transmission after the back off time has elapsed. If the collision occurs during either the preamble or Start Frame Delimiter (SFD), then these fields will be completed prior to generation of the jam sequence.

The back off time is based on an XOR of the 10 least significant bits of the data coming from FIFO and a 10 bit pseudo random number generator. The number of bits used depends on the number of collisions seen. After the first collision 1 bit is used, then the second 2 bits and so on up to the maximum of 10 bits. All 10 bits are used above ten collisions. An error will be indicated and no further attempts will be made if 16 consecutive attempts cause collision. This operation is compliant with the description in Clause 4.2.3.2.5 of the IEEE 802.3 standard which refers to the truncated binary exponential back off algorithm.

In 10/100 mode, both collisions and late collisions are treated identically, and back off and retry will be performed up to 16 times. This condition is reported in the transmit buffer descriptor word 1 (late collision, bit 26) and also in the Transmit Status Register (late collision, bit 7). An interrupt can also be generated (if enabled) when this exception occurs, and bit 5 in the Interrupt Status Register will be set.

In all modes of operation, if the transmit DMA under runs, a bad CRC is automatically appended using the same mechanism as jam insertion and the GTXER signal is asserted. For a properly configured system this should never happen.

By setting when bit 28 is set in the Network Configuration Register, the Inter Packet Gap (IPG) may be stretched beyond 96 bits depending on the length of the previously transmitted frame and the value written to the IPG Stretch Register (GMAC\_IPGS). The least significant 8 bits of the IPG Stretch Register multiply the previous frame length (including preamble). The next significant 8 bits (+1 so as not to get a divide by zero) divide the frame length to generate the IPG. IPG stretch only works in full duplex mode and when bit 28 is set in the Network Configuration Register. The IPG Stretch Register cannot be used to shrink the IPG below 96 bits.

If the back pressure bit is set in the Network Control Register, or if the HDFC configuration bit is set in the GMAC\_UR register (10M or 100M half duplex mode), the transmit block transmits 64 bits of data, which can consist of 16 nibbles of 1011 or in bit rate mode 64 1s, whenever it sees an incoming frame to force a collision. This provides a way of implementing flow control in half duplex mode.

#### 44.5.5 MAC Receive Block

All processing within the MAC receive block is implemented using a 16-bit data path. The MAC receive block checks for valid preamble, FCS, alignment and length, presents received frames to FIFO and stores the frame destination address for use by the address checking block.

If, during the frame reception, the frame is found to be too long, a bad frame indication is sent to FIFO. The receiver logic ceases to send data to memory as soon as this condition occurs.

At end of frame reception the receive block indicates to the DMA block whether the frame is good or bad. The DMA block will recover the current receive buffer if the frame was bad.

Ethernet frames are normally stored in DMA memory or to FIFO complete with the FCS. Setting the FCS remove bit in the network configuration (bit 17) causes frames to be stored without their corresponding FCS. The reported frame length field is reduced by four bytes to reflect this operation.

The receive block signals to the register block to increment the alignment, CRC (FCS), short frame, long frame, jabber or receive symbol errors when any of these exception conditions occur.

If bit 26 is set in the network configuration, CRC errors will be ignored and CRC errored frames will not be discarded, though the Frame Check Sequence Errors statistic register will still be incremented. Additionally, if configured to use the DMA and not enabled for jumbo frames mode, then bit[13] of the receiver descriptor word 1 will be updated to indicate the FCS validity for the particular frame. This is useful for applications such as EtherCAT whereby individual frames with FCS errors must be identified.

Received frames can be checked for length field error by setting the length field error frame discard bit of the Network Configuration Register (bit-16). When this bit is set, the receiver compares a frame's measured length with the length field (bytes 13 and 14) extracted from the frame. The frame is discarded if the measured length is shorter. This checking procedure is for received frames between 64 bytes and 1518 bytes in length.

Each discarded frame is counted in the 10 bit length field error statistics register. Frames where the length field is greater than or equal to 0x0600 hex will not be checked.

#### 44.5.6 Checksum Offload for IP, TCP and UDP

The GMAC can be programmed to perform IP, TCP and UDP checksum offloading in both receive and transmit directions, which is enabled by setting bit 24 in the Network Configuration Register for receive and bit 11 in the DMA Configuration Register for transmit.

IPv4 packets contain a 16-bit checksum field, which is the 16-bit 1's complement of the 1's complement sum of all 16-bit words in the header. TCP and UDP packets contain a 16-bit checksum field, which is the 16-bit 1's complement of the 1's complement sum of all 16-bit words in the header, the data and a conceptual IP pseudo header.

To calculate these checksums in software requires each byte of the packet to be processed. For TCP and UDP this can use a large amount of processing power. Offloading the checksum calculation to hardware can result in significant performance improvements.

For IP, TCP or UDP checksum offload to be useful, the operating system containing the protocol stack must be aware that this offload is available so that it can make use of the fact that the hardware can either generate or verify the checksum.

##### 44.5.6.1 Receiver Checksum Offload

When receive checksum offloading is enabled in the GMAC, the IPv4 header checksum is checked as per RFC 791, where the packet meets the following criteria:

- If present, the VLAN header must be four octets long and the CFI bit must not be set.
- Encapsulation must be RFC 894 Ethernet Type Encoding or RFC 1042 SNAP Encoding.
- IPv4 packet.
- IP header is of a valid length.

The GMAC also checks the TCP checksum as per RFC 793, or the UDP checksum as per RFC 768, if the following criteria are met:

- IPv4 or IPv6 packet
- Good IP header checksum (if IPv4).
- No IP fragmentation.
- TCP or UDP packet.

When an IP, TCP or UDP frame is received, the receive buffer descriptor gives an indication if the GMAC was able to verify the checksums. There is also an indication if the frame had SNAP encapsulation. These indication bits will replace the type ID match indication bits when the receive checksum offload is enabled. For details of these indication bits refer to [Table 44-2, “Receive Buffer Descriptor Entry”](#).

If any of the checksums are verified as incorrect by the GMAC, the packet is discarded and the appropriate statistics counter incremented.

#### 44.5.7 MAC Filtering Block

The filter block determines which frames should be written to FIFO and on to the optional DMA.

Whether a frame is passed depends on what is enabled in the Network Configuration Register, the state of the external matching pins, the contents of the specific address, type and Hash Registers and the frame's destination address and type field.

If bit 25 of the Network Configuration Register is not set, a frame will not be copied to memory if the GMAC is transmitting in half duplex mode at the time a destination address is received.

Ethernet frames are transmitted a byte at a time, least significant bit first. The first six bytes (48 bits) of an Ethernet frame make up the destination address. The first bit of the destination address, which is the LSB of the first byte of the frame, is the group or individual bit. This is one for multicast addresses and zero for unicast. The all ones address is the broadcast address and a special case of multicast.

The GMAC supports recognition of four specific addresses. Each specific address requires two registers, Specific Address Register Bottom and Specific Address Register Top. Specific Address Register Bottom stores the first four bytes of the destination address and Specific Address Register Top contains the last two bytes. The addresses stored can be specific, group, local or universal.

The destination address of received frames is compared against the data stored in the Specific Address Registers once they have been activated. The addresses are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written. If a receive frame address matches an active address, the frame is written to FIFO and on to DMA memory if used.

Frames may be filtered using the type ID field for matching. Four type ID registers exist in the register address space and each can be enabled for matching by writing a one to the MSB (bit 31) of the respective register. When a frame is received, the matching is implemented as an OR function of the various types of match.

The contents of each type ID register (when enabled) are compared against the length/type ID of the frame being received (e.g. bytes 13 and 14 in non-VLAN and non-SNAP encapsulated frames) and copied to memory if a match is found. The encoded type ID match bits (Word 0, Bit 22 and Bit 23) in the receive buffer descriptor status are set indicating which type ID register generated the match, if the receive checksum offload is disabled.

The reset state of the type ID registers is zero, hence each is initially disabled.

The following example illustrates the use of the address and type ID match registers for a MAC address of 21:43:65:87:A9:CB:

Preamble	55
SFD	D5
DA (Octet 0 - LSB)	21
DA (Octet 1)	43
DA (Octet 2)	65

DA (Octet 3)	87
DA (Octet 4)	A9
DA (Octet 5 - MSB)	CB
SA (LSB)	00 <sup>(1)</sup>
SA	00 <sup>(1)</sup>
SA (MSB)	00 <sup>(1)</sup>
Type ID (MSB)	43
Type ID (LSB)	21

Note: 1. Contains the address of the transmitting device

The sequence above shows the beginning of an Ethernet frame. Byte order of transmission is from top to bottom as shown. For a successful match to specific address 1, the following address matching registers must be set up:

Specific Address 1 Bottom [31:0] Register (GMAC\_SAB1)(Address 0x088) 0x87654321

Specific Address 1 Top [47:32] Register (GMAC\_SAT1) (Address 0x08C) 0x0000CBA9

And for a successful match to the type ID, the following type ID match 1 register must be set up:

Type ID Match 1 Register (GMAC\_TIDM1) (Address 0x0A8) 0x80004321

#### 44.5.8 Broadcast Address

Frames with the broadcast address of 0xFFFFFFFFFFFF are stored to memory only if the 'no broadcast' bit in the Network Configuration Register is set to zero.

#### 44.5.9 Hash Addressing

The hash address register is 64 bits long and takes up two locations in the memory map. The least significant bits are stored in Hash Register Bottom and the most significant bits in Hash Register Top.

The unicast hash enable and the multicast hash enable bits in the Network Configuration Register enable the reception of hash matched frames. The destination address is reduced to a 6-bit index into the 64-bit Hash Register using the following hash function: The hash function is an XOR of every sixth bit of the destination address.

```

hash_index[05] = da[05] ^ da[11] ^ da[17] ^ da[23] ^ da[29] ^ da[41] ^
da[47]
hash_index[04] = da[04] ^ da[10] ^ da[16] ^ da[22] ^ da[28] ^ da[34] ^ da[40] ^
da[46]
hash_index[03] = da[03] ^ da[09] ^ da[15] ^ da[21] ^ da[27] ^ da[33] ^ da[39] ^
da[45]
hash_index[02] = da[02] ^ da[08] ^ da[14] ^ da[20] ^ da[26] ^ da[32] ^ da[38] ^
da[44]
hash_index[01] = da[01] ^ da[07] ^ da[13] ^ da[19] ^ da[25] ^ da[31] ^ da[37] ^
da[43]
hash_index[00] = da[00] ^ da[06] ^ da[12] ^ da[18] ^ da[24] ^ da[30] ^ da[36] ^
da[42]

```

da[0] represents the least significant bit of the first byte received, that is, the multicast/unicast indicator, and da[47] represents the most significant bit of the last byte received.

If the hash index points to a bit that is set in the Hash Register then the frame will be matched according to whether the frame is multicast or unicast.

A multicast match will be signalled if the multicast hash enable bit is set, da[0] is logic 1 and the hash index points to a bit set in the Hash Register.

A unicast match will be signalled if the unicast hash enable bit is set, da[0] is logic 0 and the hash index points to a bit set in the Hash Register.

To receive all multicast frames, the Hash Register should be set with all ones and the multicast hash enable bit should be set in the Network Configuration Register.

#### 44.5.10 Copy all Frames (Promiscuous Mode)

If the copy all frames bit is set in the Network Configuration Register then all frames (except those that are too long, too short, have FCS errors or have GRXER asserted during reception) will be copied to memory. Frames with FCS errors will be copied if bit 26 is set in the Network Configuration Register.

#### 44.5.11 Disable Copy of Pause Frames

Pause frames can be prevented from being written to memory by setting the disable copying of pause frames control bit 23 in the Network Configuration Register. When set, pause frames are not copied to memory regardless of the copy all frames bit, whether a hash match is found, a type ID match is identified or if a destination address match is found.

#### 44.5.12 VLAN Support

An Ethernet encoded 802.1Q VLAN tag looks like this:

Table 44-4. 802.1Q VLAN Tag

TPID (Tag Protocol Identifier) 16 bits	TCI (Tag Control Information) 16 bits
0x8100	First 3 bits priority, then CFI bit, last 12 bits VID

The VLAN tag is inserted at the 13th byte of the frame adding an extra four bytes to the frame. To support these extra four bytes, the GMAC can accept frame lengths up to 1536 bytes by setting bit 8 in the Network Configuration Register.

If the VID (VLAN identifier) is null (0x000) this indicates a priority-tagged frame.

The following bits in the receive buffer descriptor status word give information about VLAN tagged frames:-

- Bit 21 set if receive frame is VLAN tagged (i.e. type ID of 0x8100).
- Bit 20 set if receive frame is priority tagged (i.e. type ID of 0x8100 and null VID). (If bit 20 is set, bit 21 will be set also.)
- Bit 19, 18 and 17 set to priority if bit 21 is set.
- Bit 16 set to CFI if bit 21 is set.

The GMAC can be configured to reject all frames except VLAN tagged frames by setting the discard non-VLAN frames bit in the Network Configuration Register.

#### 44.5.13 IEEE 1588 Support

IEEE 1588 is a standard for precision time synchronization in local area networks. It works with the exchange of special Precision Time Protocol (PTP) frames. The PTP messages can be transported over IEEE 802.3/Ethernet, over Internet Protocol Version 4 or over Internet Protocol Version 6 as described in the annex of IEEE P1588.D2.1.

GMAC output pins indicate the message time-stamp point (asserted on the start packet delimiter and de-asserted at end of frame) for all frames and the passage of PTP event frames (asserted when a PTP event frame is detected and de-asserted at end of frame).

Synchronization between master and slave clocks is a two stage process.

**First**, the offset between the master and slave clocks is corrected by the master sending a sync frame to the slave with a follow up frame containing the exact time the sync frame was sent. Hardware assist modules at the master and slave side detect exactly when the sync frame was sent by the master and received by the slave. The slave then corrects its clock to match the master clock.

**Second**, the transmission delay between the master and slave is corrected. The slave sends a delay request frame to the master which sends a delay response frame in reply. Hardware assist modules at the master and slave side detect exactly when the delay request frame was sent by the slave and received by the master. The slave will now have enough information to adjust its clock to account for delay. For example, if the slave was assuming zero delay, the actual delay will be half the difference between the transmit and receive time of the delay request frame (assuming equal transmit and receive times) because the slave clock will be lagging the master clock by the delay time already.

The time-stamp is taken when the message time-stamp point passes the clock time-stamp point. For Ethernet the message time-stamp point is the SFD and the clock time-stamp point is the MII interface. (The IEEE 1588 specification refers to sync and delay\_req messages as event messages as these require time-stamping. Follow up, delay response and management messages do not require time-stamping and are referred to as general messages.)

1588 version 2 defines two additional PTP event messages. These are the peer delay request (Pdelay\_Req) and peer delay response (Pdelay\_Resp) messages. These messages are used to calculate the delay on a link. Nodes at both ends of a link send both types of frames (regardless of whether they contain a master or slave clock). The Pdelay\_Resp message contains the time at which a Pdelay\_Req was received and is itself an event message. The time at which a Pdelay\_Resp message is received is returned in a Pdelay\_Resp\_Follow\_Up message.

1588 version 2 introduces transparent clocks of which there are two kinds, peer-to-peer (P2P) and end-to-end (E2E). Transparent clocks measure the transit time of event messages through a bridge and amend a correction field within the message to allow for the transit time. P2P transparent clocks additionally correct for the delay in the receive path of the link using the information gathered from the peer delay frames. With P2P transparent clocks delay\_req messages are not used to measure link delay. This simplifies the protocol and makes larger systems more stable.

The GMAC recognizes four different encapsulations for PTP event messages:

1. 1588 version 1 (UDP/IPv4 multicast)
2. 1588 version 2 (UDP/IPv4 multicast)
3. 1588 version 2 (UDP/IPv6 multicast)
4. 1588 version 2 (Ethernet multicast)

Example of a sync frame in the 1588 version 1 format:

Preamble/SFD	55555555555555D5
DA (Octets 0 - 5)	
SA (Octets 6 - 11)	
Type (Octets 12-13)	0800
IP stuff (Octets 14-22)	
UDP (Octet 23)	11
IP stuff (Octets 24-29)	
IP DA (Octets 30-32)	E00001
IP DA (Octet 33)	81 or 82 or 83 or 84
source IP port (Octets 34-35)	
dest IP port (Octets 36-37)	013F
other stuff (Octets 38-42)	
version PTP (Octet 43)	01
other stuff (Octets 44-73)	
control (Octet 74)	00
other stuff (Octets 75-168)	

Example of a delay request frame in the 1588 version 1 format:

Preamble/SFD	55555555555555D5
DA (Octets 0 - 5)	
SA (Octets 6 - 11)	
Type (Octets 12-13)	0800
IP stuff (Octets 14-22)	
UDP (Octet 23)	11
IP stuff (Octets 24-29)	
IP DA (Octets 30-32)	E00001
IP DA (Octet 33)	81 or 82 or 83 or 84
source IP port (Octets 34-35)	
dest IP port (Octets 36-37)	013F
other stuff (Octets 38-42)	
version PTP (Octet 43)	01
other stuff (Octets 44-73)	
control (Octet 74)	01
other stuff (Octets 75-168)	

For 1588 version 1 messages, sync and delay request frames are indicated by the GMAC if the frame type field indicates TCP/IP, UDP protocol is indicated, the destination IP address is 224.0.1.129/130/131 or 132, the destination UDP port is 319 and the control field is correct.

The control field is 0x00 for sync frames and 0x01 for delay request frames.

For 1588 version 2 messages, the type of frame is determined by looking at the message type field in the first byte of the PTP frame. Whether a frame is version 1 or version 2 can be determined by looking at the version PTP field in the second byte of both version 1 and version 2 PTP frames.

In version 2 messages sync frames have a message type value of 0x0, delay\_req have 0x1, pdelay\_req have 0x2 and pdelay\_resp have 0x3.

Example of a sync frame in the 1588 version 2 (UDP/IPv4) format

Preamble/SFD	55555555555555D5
DA (Octets 0 - 5)	
SA (Octets 6 - 11)	
Type (Octets 12-13)	0800
IP stuff (Octets 14-22)	
UDP (Octet 23)	11
IP stuff (Octets 24-29)	
IP DA (Octets 30-33)	E0000181
source IP port (Octets 34-35)	
dest IP port (Octets 36-37)	013F

other stuff (Octets 38-41)	
message type (Octet 42)	00
version PTP (Octet 43)	02

Example of a pdelay\_req frame in the 1588 version 2 (UDP/IPv4) format

Preamble/SFD	55555555555555D5
DA (Octets 0 - 5)	
SA (Octets 6 - 11)	
Type (Octets 12-13)	0800
IP stuff (Octets 14-22)	
UDP (Octet 23)	11
IP stuff (Octets 24-29)	
IP DA (Octets 30-33)	E000006B
source IP port (Octets 34-35)	
dest IP port (Octets 36-37)	013F
other stuff (Octets 38-41)	
message type (Octet 42)	02
version PTP (Octet 43)	02

Example of a synch frame in the 1588 version 2 (UDP/IPv6) format

Preamble/SFD	55555555555555D5
DA (Octets 0 - 5)	
SA (Octets 6 - 11)	
Type (Octets 12-13)	86dd
IP stuff (Octets 14-19)	
UDP (Octet 20)	11
IP stuff (Octets 21-37)	
IP DA (Octets 38-53)	FF0X000000000018
source IP port (Octets 54-55)	
dest IP port (Octets 56-57)	013F
other stuff (Octets 58-61)	
message type (Octet 62)	00
other stuff (Octets 63-93)	
version PTP (Octet 94)	02

Example of a pdelay\_resp frame in the 1588 version 2 (UDP/IPv6) format

Preamble/SFD	55555555555555D5
DA (Octets 0 - 5)	
SA (Octets 6 - 11)	
Type (Octets 12-13)	86dd
IP stuff (Octets 14-19)	
UDP (Octet 20)	11
IP stuff (Octets 21-37)	
IP DA (Octets 38-53)	FF0200000000006B
source IP port (Octets 54-55)	
dest IP port (Octets 56-57)	013F
other stuff (Octets 58-61)	
message type (Octet 62)	03
other stuff (Octets 63-93)	
version PTP (Octet 94)	02

Example of a sync frame in the 1588 version 2 (Ethernet multicast) format. For the multicast address 011B19000000 sync and delay request frames are recognized depending on the message type field, 00 for sync and 01 for delay request:

Preamble/SFD	55555555555555D5
DA (Octets 0 - 5)	011B19000000
SA (Octets 6 - 11)	
Type (Octets 12-13)	88F7
message type (Octet 14)	00
version PTP (Octet 15)	02

Example of a pdelay\_req frame in the 1588 version 2 (Ethernet multicast) format, these need a special multicast address so they can get through ports blocked by the spanning tree protocol. For the multicast address 0180C200000E sync, pdelay request and pdelay response frames are recognized depending on the message type field, 00 for sync, 02 for pdelay request and 03 for pdelay response.

Preamble/SFD	55555555555555D5
DA (Octets 0 - 5)	0180C200000E
SA (Octets 6 - 11)	
Type (Octets 12-13)	88F7
message type (Octet 14)	00
version PTP (Octet 15)	02

#### 44.5.14 Time Stamp Unit

The TSU consists of a timer and registers to capture the time at which PTP event frames cross the message timestamp point. An interrupt is issued when a capture register is updated.

The timer is implemented as a 62-bit register with the upper 32 bits counting seconds and the lower 30 bits counting nanoseconds. The lower 30 bits roll over when they have counted to one second. An interrupt is generated when the seconds increment. The timer value can be read, written and adjusted through the APB interface.

The timer is clocked by MCK.

The amount by which the timer increments each clock cycle is controlled by the timer increment register. Bits [7:0] are the default increment value in nanoseconds. If the rest of the register is written with zero the timer increments by the value in [7:0] each clock cycle.

Bits [15:8] of the increment register are the alternative increment value in nanoseconds and bits [23:16] are the number of increments after which the alternative increment value is used. If [23:16] are zero then the alternative increment value will never be used.

Taking the example of 10.2 MHz, there are 102 cycles every ten microseconds or 51 every five microseconds. So a timer with a 10.2 MHz clock source is constructed by incrementing by 98 ns for fifty cycles and then incrementing by 100 ns ( $98 \times 50 + 100 = 5000$ ). This is programmed by setting the 1588 Timer Increment Register to 0x00326462.

For a 49.8 MHz clock source it would be 20 ns for 248 cycles followed by an increment of 40 ns ( $20 \times 248 + 40 = 5000$ ) programmed as 0x00F82814.

Having eight bits for the “number of increments” field allows frequencies up to 50 MHz to be supported with 200 kHz resolution

Without the alternative increment field the period of the clock would be limited to an integer number of nanoseconds, resulting in supported clock frequencies of 8, 10, 20, 25, 40, 50, 100, 125, 200 and 250 MHz.

GMAC is configured to operate as PTP slave. The timer register increments as normal but the timer value is copied to the sync strobe register.

There are six additional 62-bit registers that capture the time at which PTP event frames are transmitted and received. An interrupt is issued when these registers are updated.

#### 44.5.15 MAC 802.3 Pause Frame Support

The start of a MAC 802.3 pause frame is as follows;

Table 44-5. Start of an 802.3 Pause Frame

Destination Address	Source Address	Type (Mac Control Frame)	Pause Opcode	Pause Time
0x0180C2000001	6 bytes	0x8808	0x0001	2 bytes

The GMAC supports both hardware controlled pause of the transmitter, upon reception of a pause frame, and hardware generated pause frame transmission.

Note: Please refer to Clause 31, and Annex 31A and 31B of the IEEE standard 802.3 for a full description of MAC 802.3 pause operation.

#### 44.5.15.1 802.3 Pause Frame Reception

Bit 13 of the Network Configuration Register is the pause enable control for reception. If this bit is set, transmission will pause if a non zero pause quantum frame is received.

If a valid pause frame is received then the Pause Time Register is updated with the new frame's pause time, regardless of whether a previous pause frame is active or not. An interrupt (either bit 12 or bit 13 of the Interrupt Status Register) is triggered when a pause frame is received, but only if the interrupt has been enabled (bit 12 and bit 13 of the Interrupt Mask Register). Pause frames received with non zero quantum are indicated through the interrupt bit 12 of the Interrupt Status Register. Pause frames received with zero quantum are indicated on bit 13 of the Interrupt Status Register.

Once the Pause Time Register is loaded and the frame currently being transmitted has been sent, no new frames are transmitted until the pause time reaches zero. The loading of a new pause time, and hence the pausing of transmission, only occurs when the GMAC is configured for full duplex operation. If the GMAC is configured for half duplex there will be no transmission pause, but the pause frame received interrupt will still be triggered. A valid pause frame is defined as having a destination address that matches either the address stored in Specific Address Register 1 or if it matches the reserved address of 0x0180C2000001. It must also have the MAC control frame type ID of 0x8808 and have the pause opcode of 0x0001.

Pause frames that have frame check sequence (FCS) or other errors will be treated as invalid and will be discarded. 802.3 Pause frames that are received after Priority-based Flow Control (PFC) has been negotiated will also be discarded. Valid pause frames received will increment the pause frames received statistic register.

The pause time register decrements every 512 bit times once transmission has stopped. For test purposes, the retry test bit can be set (bit 12 in the Network Configuration Register) which causes the Pause Time Register to decrement every GTXCK cycle once transmission has stopped.

The interrupt (bit 13 in the Interrupt Status Register) is asserted whenever the Pause Time Register decrements to zero (assuming it has been enabled by bit 13 in the Interrupt Mask Register). This interrupt is also set when a zero quantum pause frame is received.

#### 44.5.15.2 802.3 Pause Frame Transmission

Automatic transmission of pause frames is supported through the transmit pause frame bits of the Network Control Register. If either bit 11 or bit 12 of the Network Control Register is written with logic 1, an 802.3 pause frame will be transmitted, providing full duplex is selected in the Network Configuration Register and the transmit block is enabled in the Network Control Register.

Pause frame transmission will happen immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise the following:

- A destination address of 01-80-C2-00-00-01
- A source address taken from Specific Address Register 1
- A type ID of 88-08 (MAC control frame)
- A pause opcode of 00-01
- A pause quantum register
- Fill of 00 to take the frame to minimum frame length
- Valid FCS

The pause quantum used in the generated frame will depend on the trigger source for the frame as follows:

- If bit 11 is written with a one, the pause quantum will be taken from the Transmit Pause Quantum Register. The Transmit Pause Quantum Register resets to a value of 0xFFFF giving maximum pause quantum as default.
- If bit 12 is written with a one, the pause quantum will be zero.

After transmission, a pause frame transmitted interrupt will be generated (bit 14 of the Interrupt Status Register) and the only statistics register that will be incremented will be the Pause Frames Transmitted Register.

Pause frames can also be transmitted by the MAC using normal frame transmission methods.

#### 44.5.16 MAC PFC Priority-based Pause Frame Support

The start of a Priority-based Flow Control (PFC) pause frame is as follows:

**Table 44-6. Start of a PFC Pause Frame**

Destination Address	Source Address	Type (Mac Control Frame)	Pause Opcode	Priority Enable Vector	Pause Time
0x0180C2000001	6 bytes	0x8808	0x1001	2 bytes	8*2 bytes

The GMAC supports PFC priority-based pause transmission and reception. Before PFC pause frames can be received, bit 16 of the Network Control Register must be set.

Note: Refer to the 802.1Qbb standard for a full description of priority-based pause operation.

##### 44.5.16.1 PFC Pause Frame Reception

The ability to receive and decode priority-based pause frames is enabled by setting bit 16 of the Network Control Register. When this bit is set, the GMAC will match either classic 802.3 pause frames or PFC priority-based pause frames. Once a priority-based pause frame has been received and matched, then from that moment on the GMAC will only match on priority-based pause frames (this is a 802.1Qbb requirement, known as PFC negotiation). Once priority-based pause has been negotiated, any received 802.3x format pause frames will not be acted upon.

If a valid priority-based pause frame is received then the GMAC will decode the frame and determine which, if any, of the 8 priorities require to be paused. Up to 8 Pause Time Registers are then updated with the 8 pause times extracted from the frame regardless of whether a previous pause operation is active or not. An interrupt (either bit 12 or bit 13 of the Interrupt Status Register) is triggered when a pause frame is received, but only if the interrupt has been enabled (bit 12 and bit 13 of the Interrupt Mask Register). Pause frames received with non zero quantum are indicated through the interrupt bit 12 of the Interrupt Status Register. Pause frames received with zero quantum are indicated on bit 13 of the Interrupt Status Register. The loading of a new pause time only occurs when the GMAC is configured for full duplex operation. If the GMAC is configured for half duplex, the pause time counters will not be loaded, but the pause frame received interrupt will still be triggered. A valid pause frame is defined as having a destination address that matches either the address stored in Specific Address Register 1 or if it matches the reserved address of 0x0180C2000001. It must also have the MAC control frame type ID of 0x8808 and have the pause opcode of 0x0101.

Pause frames that have frame check sequence (FCS) or other errors will be treated as invalid and will be discarded. Valid pause frames received will increment the Pause Frames Received Statistic Register.

The Pause Time Registers decrement every 512 bit times immediately following the PFC frame reception. For test purposes, the retry test bit can be set (bit 12 in the Network Configuration Register) which causes the Pause Time Register to decrement every GRXCK cycle once transmission has stopped.

The interrupt (bit 13 in the Interrupt Status Register) is asserted whenever the Pause Time Register decrements to zero (assuming it has been enabled by bit 13 in the Interrupt Mask Register). This interrupt is also set when a zero quantum pause frame is received.

#### 44.5.16.2 PFC Pause Frame Transmission

Automatic transmission of pause frames is supported through the transmit priority-based pause frame bit of the Network Control Register. If bit 17 of the Network Control Register is written with logic 1, a PFC pause frame will be transmitted providing full duplex is selected in the Network Configuration Register and the transmit block is enabled in the Network Control Register. When bit 17 of the Network Control Register is set, the fields of the priority-based pause frame will be built using the values stored in the Transmit PFC Pause Register.

Pause frame transmission will happen immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise the following:

- A destination address of 01-80-C2-00-00-01
- A source address taken from Specific Address Register 1
- A type ID of 88-08 (MAC control frame)
- A pause opcode of 01-01
- A priority enable vector taken from Transmit PFC Pause Register
- 8 pause quantum registers
- Fill of 00 to take the frame to minimum frame length
- Valid FCS

The pause quantum registers used in the generated frame will depend on the trigger source for the frame as follows:

- If bit 17 of the Network Control Register is written with a one, then the priority enable vector of the priority-based pause frame will be set equal to the value stored in the Transmit PFC Pause Register [7:0]. For each entry equal to zero in the Transmit PFC Pause Register[15:8], the pause quantum field of the pause frame associated with that entry will be taken from the transmit pause quantum register. For each entry equal to one in the Transmit PFC Pause Register[15:8], the pause quantum associated with that entry will be zero.
- The Transmit Pause Quantum Register resets to a value of 0xFFFF giving maximum pause quantum as default.

After transmission, a pause frame transmitted interrupt will be generated (bit 14 of the Interrupt Status Register) and the only statistics register that will be incremented will be the Pause Frames Transmitted Register.

PFC Pause frames can also be transmitted by the MAC using normal frame transmission methods.

#### 44.5.17 PHY Interface

The following PHY interface is supported by the Ethernet MAC:

- MII

The MII interface is provided for 10/100 operation and uses txd[3:0] and rxd[3:0].

#### 44.5.18 10/100 Operation

The 10/100 Mbps speed bit in the Network Configuration Register is used to select between 10 Mbps and 100 Mbps.

#### 44.5.19 Jumbo Frames

The jumbo frames enable bit in the Network Configuration Register allows the GMAC, in its default configuration, to receive jumbo frames up to 10240 bytes in size. This operation does not form part of the IEEE 802.3 specification and is normally disabled. When jumbo frames are enabled, frames received with a frame size greater than 10240 bytes are discarded.

## 44.6 Programming Interface

### 44.6.1 Initialization

#### 44.6.1.1 Configuration

Initialization of the GMAC configuration (e.g. loop back mode, frequency ratios) must be done while the transmit and receive circuits are disabled. See the description of the Network Control Register and Network Configuration Register earlier in this document.

To change loop back mode, the following sequence of operations must be followed:

1. Write to Network Control Register to disable transmit and receive circuits.
2. Write to Network Control Register to change loop back mode.
3. Write to Network Control Register to re-enable transmit or receive circuits.

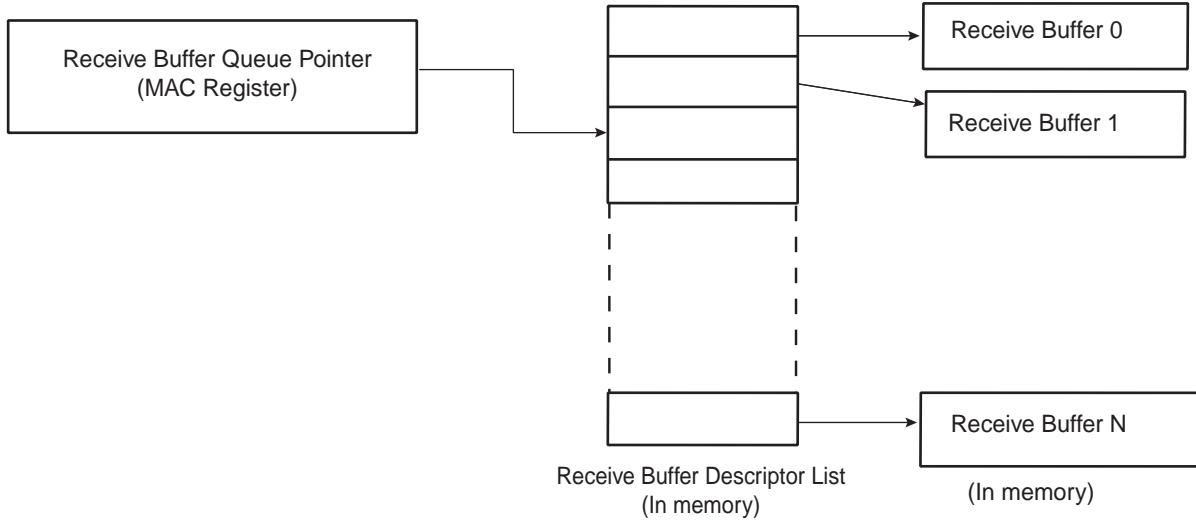
Note: These writes to Network Control Register cannot be combined in any way.

#### 44.6.1.2 Receive Buffer List

Receive data is written to areas of data (i.e. buffers) in system memory. These buffers are listed in another data structure that also resides in main memory. This data structure (receive buffer queue) is a sequence of descriptor entries as defined in [Table 44-2, "Receive Buffer Descriptor Entry"](#)

The Receive Buffer Queue Pointer Register points to this data structure.

**Figure 44-2. Receive Buffer List**



To create the list of buffers:

1. Allocate a number ( $N$ ) of buffers of  $X$  bytes in system memory, where  $X$  is the DMA buffer length programmed in the DMA Configuration Register.
2. Allocate an area  $8N$  bytes for the receive buffer descriptor list in system memory and create  $N$  entries in this list. Mark all entries in this list as owned by GMAC, i.e., bit 0 of word 0 set to 0.
3. Mark the last descriptor in the queue with the wrap bit (bit 1 in word 0 set to 1).
4. Write address of receive buffer descriptor list and control information to GMAC register receive buffer queue pointer
5. The receive circuits can then be enabled by writing to the address recognition registers and the Network Control Register.

#### 44.6.1.3 Transmit Buffer List

Transmit data is read from areas of data (the buffers) in system memory. These buffers are listed in another data structure that also resides in main memory. This data structure (Transmit Buffer Queue) is a sequence of descriptor entries as defined in [Table 44-3, “Transmit Buffer Descriptor Entry”](#).

The Transmit Buffer Queue Pointer Register points to this data structure.

To create this list of buffers:

1. Allocate a number (N) of buffers of between 1 and 2047 bytes of data to be transmitted in system memory. Up to 128 buffers per frame are allowed.
2. Allocate an area  $8N$  bytes for the transmit buffer descriptor list in system memory and create N entries in this list. Mark all entries in this list as owned by GMAC, i.e., bit 31 of word 1 set to 0.
3. Mark the last descriptor in the queue with the wrap bit (bit 30 in word 1 set to 1).
4. Write address of transmit buffer descriptor list and control information to GMAC register transmit buffer queue pointer.
5. The transmit circuits can then be enabled by writing to the Network Control Register.

#### 44.6.1.4 Address Matching

The GMAC register pair hash address and the four Specific Address Register-pairs must be written with the required values. Each register pair comprises of a bottom register and top register with the bottom register being written first. The address matching is disabled for a particular register pair after the bottom register has been written and re-enabled when the top register is written. Each register pair may be written at any time, regardless of whether the receive circuits are enabled or disabled.

As an example, to set Specific Address Register 1 to recognize destination address 21:43:65:87:A9:CB, the following values are written to Specific Address Register 1 bottom and Specific Address Register 1 top:

- Specific Address Register 1 bottom bits 31:0 (0x98): 8765\_4321 hex.
- Specific Address Register 1 top bits 31:0 (0x9C): 0000\_cba9 hex.

#### 44.6.1.5 PHY Maintenance

The PHY Maintenance Register is implemented as a shift register. Writing to the register starts a shift operation which is signalled as complete when bit two is set in the Network Status Register (about 2000 MCK cycles later when bits [18:16] are set to 010 in the Network Configuration Register). An interrupt is generated as this bit is set.

During this time, the MSB of the register is output on the MDIO pin and the LSB updated from the MDIO pin with each Management Data Clock (MDC) cycle. This causes the transmission of a PHY management frame on MDIO. See section 22.2.4.5 of the IEEE 802.3 standard.

Reading during the shift operation will return the current contents of the shift register. At the end of the management operation the bits will have shifted back to their original locations. For a read operation the data bits are updated with data read from the PHY. It is important to write the correct values to the register to ensure a valid PHY management frame is produced.

The Management Data Clock (MDC) should not toggle faster than 2.5 MHz (minimum period of 400 ns), as defined by the IEEE 802.3 standard. MDC is generated by dividing down MCK. Three bits in the Network Configuration Register determine by how much MCK should be divided to produce MDC.

#### 44.6.1.6 Interrupts

There are 18 interrupt conditions that are detected within the GMAC. These are ORed to make a single interrupt. Depending on the overall system design this may be passed through a further level of interrupt collection (interrupt controller). On receipt of the interrupt signal, the CPU enters the interrupt handler. Refer to the device interrupt controller documentation to identify that it is the GMAC that is generating the interrupt. To ascertain which interrupt, read the Interrupt Status Register. Note that in the default configuration this register will clear itself after being read, though this may be configured to be write-one-to-clear if desired.

At reset all interrupts are disabled. To enable an interrupt, write to Interrupt Enable Register with the pertinent interrupt bit set to 1. To disable an interrupt, write to Interrupt Disable Register with the pertinent interrupt bit set to 1. To check whether an interrupt is enabled or disabled, read Interrupt Mask Register. If the bit is set to 1, the interrupt is disabled.

#### 44.6.1.7 Transmitting Frames

To set up a frame for transmission:

1. Enable transmit in the Network Control Register.
2. Allocate an area of system memory for transmit data. This does not have to be contiguous, varying byte lengths can be used if they conclude on byte borders.
3. Set-up the transmit buffer list by writing buffer addresses to word zero of the transmit buffer descriptor entries and control and length to word one.
4. Write data for transmission into the buffers pointed to by the descriptors.
5. Write the address of the first buffer descriptor to transmit buffer descriptor queue pointer.
6. Enable appropriate interrupts.
7. Write to the transmit start bit (TSTART) in the Network Control Register.

#### 44.6.1.8 Receiving Frames

When a frame is received and the receive circuits are enabled, the GMAC checks the address and, in the following cases, the frame is written to system memory:

- If it matches one of the four Specific Address Registers.
- If it matches one of the four type ID registers.
- If it matches the hash address function.
- If it is a broadcast address (0xFFFFFFFFFFFF) and broadcasts are allowed.
- If the GMAC is configured to “copy all frames”.

The register receive buffer queue pointer points to the next entry in the receive buffer descriptor list and the GMAC uses this as the address in system memory to write the frame to.

Once the frame has been completely and successfully received and written to system memory, the GMAC then updates the receive buffer descriptor entry (see [Table 44-2, “Receive Buffer Descriptor Entry”](#)) with the reason for the address match and marks the area as being owned by software. Once this is complete, a receive complete interrupt is set. Software is then responsible for copying the data to the application area and releasing the buffer (by writing the ownership bit back to 0).

If the GMAC is unable to write the data at a rate to match the incoming frame, then a receive overrun interrupt is set. If there is no receive buffer available, i.e., the next buffer is still owned by software, a receive buffer not available interrupt is set. If the frame is not successfully received, a Statistics Register is incremented and the frame is discarded without informing software.

#### 44.6.2 Statistics Registers

Statistics Registers are described in the User Interface beginning with [Section 44.7.36 on page 1256](#) and ending with [Section 44.7.80 on page 1278](#).

The Statistics Register Block begins at 0x100 and runs to 0x1B0, and is comprised of the registers listed below.

Octets Transmitted [31:0] Register	Broadcast Frames Received Register
Octets Transmitted [47:32] Register	Multicast Frames Received Register
Frames Transmitted Register	Pause Frames Received Register
Broadcast Frames Transmitted Register	64 Byte Frames Received Register
Multicast Frames Transmitted Register	65 to 127 Byte Frames Received Register
Pause Frames Transmitted Register	128 to 255 Byte Frames Received Register
64 Byte Frames Transmitted Register	256 to 511Byte Frames Received Register
65 to 127 Byte Frames Transmitted Register	512 to 1023 Byte Frames Received Register
128 to 255 Byte Frames Transmitted Register	1024 to 1518 Byte Frames Received Register
256 to 511 Byte Frames Transmitted Register	1519 to Maximum Byte Frames Received Register
512 to 1023 Byte Frames Transmitted Register	Undersize Frames Received Register
1024 to 1518 Byte Frames Transmitted Register	Oversize Frames Received Register
Greater Than 1518 Byte Frames Transmitted Register	Jabbers Received Register
Transmit Under Runs Register	Frame Check Sequence Errors Register
Single Collision Frames Register	Length Field Frame Errors Register
Multiple Collision Frames Register	Receive Symbol Errors Register
Excessive Collisions Register	Alignment Errors Register
Late Collisions Register	Receive Resource Errors Register
Deferred Transmission Frames Register	Receive Overrun Register
Carrier Sense Errors Register	IP Header Checksum Errors Register
Octets Received [31:0] Received	TCP Checksum Errors Register
Octets Received [47:32] Received	UDP Checksum Errors Register
Frames Received Register	

These registers reset to zero on a read and stick at all ones when they count to their maximum value. They should be read frequently enough to prevent loss of data.

In order to reduce overall design area, the Statistics Registers may be optionally removed in the configuration file if they are deemed unnecessary for a particular design.

The Receive Statistics Registers are only incremented when the receive enable bit is set in the Network Control Register.

Once a Statistics Register has been read, it is automatically cleared. When reading the Octets Transmitted and Octets Received Registers, bits 31:0 should be read prior to bits 47:32 to ensure reliable operation.

## 44.7 Ethernet MAC (GMAC) User Interface

Table 44-7. Register Mapping

Offset	Register	Name	Access	Reset
0x000	Network Control Register	GMAC_NCR	Read-write	0x0000_0000
0x004	Network Configuration Register	GMAC_NCFGR	Read-write	0x0008_0000
0x008	Network Status Register	GMAC_NSRR	Read-only	0b01x0
0x00C	User Register	GMAC_UR	Read-write	0x0000_0000
0x010	DMA Configuration Register	GMAC_DCFGR	Read-write	0x0002_0004
0x014	Transmit Status Register	GMAC_TSRR	Read-write	0x0000_0000
0x018	Receive Buffer Queue Base Address	GMAC_RBQB	Read-write	0x0000_0000
0x01C	Transmit Buffer Queue Base Address	GMAC_TBQB	Read-write	0x0000_0000
0x020	Receive Status Register	GMAC_RSR	Read-write	0x0000_0000
0x024	Interrupt Status Register	GMAC_ISR	Read-only	0x0000_0000
0x028	Interrupt Enable Register	GMAC_IER	Write-only	–
0x02C	Interrupt Disable Register	GMAC_IDR	Write-only	–
0x030	Interrupt Mask Register	GMAC_IMR	Read-only	0x07FF_FFFF
0x034	PHY Maintenance Register	GMAC_MAN	Read-write	0x0000_0000
0x038	Received Pause Quantum Register	GMAC_RPQ	Read-only	0x0000_0000
0x03C	Transmit Pause Quantum Register	GMAC_TPQ	Read-write	0x0000_FFFF
0x040-0x07C	Reserved	–	–	–
0x080	Hash Register Bottom [31:0]	GMAC_HRB	Read-write	0x0000_0000
0x084	Hash Register Top [63:32]	GMAC_HRT	Read-write	0x0000_0000
0x088	Specific Address 1 Bottom [31:0] Register	GMAC_SAB1	Read-write	0x0000_0000
0x08C	Specific Address 1 Top [47:32] Register	GMAC_SAT1	Read-write	0x0000_0000
0x090	Specific Address 2 Bottom [31:0] Register	GMAC_SAB2	Read-write	0x0000_0000
0x094	Specific Address 2 Top [47:32] Register	GMAC_SAT2	Read-write	0x0000_0000
0x098	Specific Address 3 Bottom [31:0] Register	GMAC_SAB3	Read-write	0x0000_0000
0x09C	Specific Address 3 Top [47:32] Register	GMAC_SAT3	Read-write	0x0000_0000
0x0A0	Specific Address 4 Bottom [31:0] Register	GMAC_SAB4	Read-write	0x0000_0000
0x0A4	Specific Address 4 Top [47:32] Register	GMAC_SAT4	Read-write	0x0000_0000
0x0A8	Type ID Match 1 Register	GMAC_TIDM1	Read-write	0x0000_0000
0x0AC	Type ID Match 2 Register	GMAC_TIDM2	Read-write	0x0000_0000
0x0B0	Type ID Match 3 Register	GMAC_TIDM3	Read-write	0x0000_0000
0x0B4	Type ID Match 4 Register	GMAC_TIDM4	Read-write	0x0000_0000
0x0BC	IPG Stretch Register	GMAC_IPGS	Read-write	0x0000_0000
0x0C0	Stacked VLAN Register	GMAC_SVLAN	Read-write	0x0000_0000
0x0C4	Transmit PFC Pause Register	GMAC_TPFCP	Read-write	0x0000_0000
0x0C8	Specific Address 1 Mask Bottom [31:0] Register	GMAC_SAMB1	Read-write	0x0000_0000

**Table 44-7. Register Mapping (Continued)**

<b>Offset</b>	<b>Register</b>	<b>Name</b>	<b>Access</b>	<b>Reset</b>
0x0CC	Specific Address 1 Mask Top [47:32] Register	GMAC_SAMT1	Read-write	0x0000_0000
0x0FC	Reserved	-	-	-
0x100	Octets Transmitted [31:0] Register	GMAC_OTLO	Read-only	0x0000_0000
0x104	Octets Transmitted [47:32] Register	GMAC_OTHI	Read-only	0x0000_0000
0x108	Frames Transmitted Register	GMAC_FT	Read-only	0x0000_0000
0x10C	Broadcast Frames Transmitted Register	GMAC_BCFT	Read-only	0x0000_0000
0x110	Multicast Frames Transmitted Register	GMAC_MFT	Read-only	0x0000_0000
0x114	Pause Frames Transmitted Register	GMAC_PFT	Read-only	0x0000_0000
0x118	64 Byte Frames Transmitted Register	GMAC_BFT64	Read-only	0x0000_0000
0x11C	65 to 127 Byte Frames Transmitted Register	GMAC_TBFT127	Read-only	0x0000_0000
0x120	128 to 255 Byte Frames Transmitted Register	GMAC_TBFT255	Read-only	0x0000_0000
0x124	256 to 511 Byte Frames Transmitted Register	GMAC_TBFT511	Read-only	0x0000_0000
0x128	512 to 1023 Byte Frames Transmitted Register	GMAC_TBFT1023	Read-only	0x0000_0000
0x12C	1024 to 1518 Byte Frames Transmitted Register	GMAC_TBFT1518	Read-only	0x0000_0000
0x130	Greater Than 1518 Byte Frames Transmitted Register	GMAC_GTBFT1518	Read-only	0x0000_0000
0x134	Transmit Under Runs Register	GMAC_TUR	Read-only	0x0000_0000
0x138	Single Collision Frames Register	GMAC_SCF	Read-only	0x0000_0000
0x13C	Multiple Collision Frames Register	GMAC_MCF	Read-only	0x0000_0000
0x140	Excessive Collisions Register	GMAC_EC	Read-only	0x0000_0000
0x144	Late Collisions Register	GMAC_LC	Read-only	0x0000_0000
0x148	Deferred Transmission Frames Register	GMAC_DTF	Read-only	0x0000_0000
0x14C	Carrier Sense Errors Register	GMAC_CSE	Read-only	0x0000_0000
0x150	Octets Received [31:0] Received	GMAC_ORLO	Read-only	0x0000_0000
0x154	Octets Received [47:32] Received	GMAC_ORHI	Read-only	0x0000_0000
0x158	Frames Received Register	GMAC_FR	Read-only	0x0000_0000
0x15C	Broadcast Frames Received Register	GMAC_BCFR	Read-only	0x0000_0000
0x160	Multicast Frames Received Register	GMAC_MFR	Read-only	0x0000_0000
0x164	Pause Frames Received Register	GMAC_PFR	Read-only	0x0000_0000
0x168	64 Byte Frames Received Register	GMAC_BFR64	Read-only	0x0000_0000
0x16C	65 to 127 Byte Frames Received Register	GMAC_TBFR127	Read-only	0x0000_0000
0x170	128 to 255 Byte Frames Received Register	GMAC_TBFR255	Read-only	0x0000_0000
0x174	256 to 511Byte Frames Received Register	GMAC_TBFR511	Read-only	0x0000_0000
0x178	512 to 1023 Byte Frames Received Register	GMAC_TBFR1023	Read-only	0x0000_0000
0x17C	1024 to 1518 Byte Frames Received Register	GMAC_TBFR1518	Read-only	0x0000_0000
0x180	1519 to Maximum Byte Frames Received Register	GMAC_TMXBFR	Read-only	0x0000_0000
0x184	Undersize Frames Received Register	GMAC_UFR	Read-only	0x0000_0000
0x188	Oversize Frames Received Register	GMAC_OFR	Read-only	0x0000_0000

**Table 44-7. Register Mapping (Continued)**

<b>Offset</b>	<b>Register</b>	<b>Name</b>	<b>Access</b>	<b>Reset</b>
0x18C	Jabbers Received Register	GMAC_JR	Read-only	0x0000_0000
0x190	Frame Check Sequence Errors Register	GMAC_FCSE	Read-only	0x0000_0000
0x194	Length Field Frame Errors Register	GMAC_LFFE	Read-only	0x0000_0000
0x198	Receive Symbol Errors Register	GMAC_RSE	Read-only	0x0000_0000
0x19C	Alignment Errors Register	GMAC_AE	Read-only	0x0000_0000
0x1A0	Receive Resource Errors Register	GMAC_RRE	Read-only	0x0000_0000
0x1A4	Receive Overrun Register	GMAC_ROE	Read-only	0x0000_0000
0x1A8	IP Header Checksum Errors Register	GMAC_IHCE	Read-only	0x0000_0000
0x1AC	TCP Checksum Errors Register	GMAC_TCE	Read-only	0x0000_0000
0x1B0	UDP Checksum Errors Register	GMAC_UCE	Read-only	0x0000_0000
0x1C8	1588 Timer Sync Strobe Seconds Register	GMAC_TSSS	Read-write	0x0000_0000
0x1CC	1588 Timer Sync Strobe Nanoseconds Register	GMAC_TSSN	Read-write	0x0000_0000
0x1D0	1588 Timer Seconds Register	GMAC_TS	Read-write	0x0000_0000
0x1D4	1588 Timer Nanoseconds Register	GMAC_TN	Read-write	0x0000_0000
0x1D8	1588 Timer Adjust Register	GMAC_TA	Write-only	
0x1DC	1588 Timer Increment Register	GMAC_TI	Read-write	0x0000_0000
0x1E0	PTP Event Frame Transmitted Seconds	GMAC_EFTS	Read-only	0x0000_0000
0x1E4	PTP Event Frame Transmitted Nanoseconds	GMAC_EFTN	Read-only	0x0000_0000
0x1E8	PTP Event Frame Received Seconds	GMAC_EFRS	Read-only	0x0000_0000
0x1EC	PTP Event Frame Received Nanoseconds	GMAC_EFRN	Read-only	0x0000_0000
0x1F0	PTP Peer Event Frame Transmitted Seconds	GMAC_PEFTS	Read-only	0x0000_0000
0x1F4	PTP Peer Event Frame Transmitted Nanoseconds	GMAC_PEFTN	Read-only	0x0000_0000
0x1F8	PTP Peer Event Frame Received Seconds	GMAC_PEFRS	Read-only	0x0000_0000
0x1FC	PTP Peer Event Frame Received Nanoseconds	GMAC_PEFRN	Read-only	0x0000_0000
0x200-0x23C	Reserved	-	-	-
0x280-0x298	Reserved	-	-	-

#### 44.7.1 Network Control Register

**Name:** GMAC\_NCR

**Address:** 0x40034000

**Access:** Read-write

31	30	29	28	27	26	25	24
—							
23	22	21	20	19	18	17	16
FNP TXPBPF ENPBPR							
15	14	13	12	11	10	9	8
SRTSM	RDS	—	TXZQPF	TXPF	THALT	TSTART	BP
7	6	5	4	3	2	1	0
WESTAT	INCSTAT	CLRSTAT	MPE	TXEN	RXEN	LBL	—

- **LBL: Loop Back Local**

Connects GTX to GRX , GTXEN to GRXDV and forces full duplex mode. GRXCK and GTXCK may malfunction as the GMAC is switched into and out of internal loop back. It is important that receive and transmit circuits have already been disabled when making the switch into and out of internal loop back.

- **RXEN: Receive Enable**

When set, RXEN enables the GMAC to receive data. When reset frame reception stops immediately and the receive pipeline will be cleared. The Receive Queue Pointer Register is unaffected.

- **TXEN: Transmit Enable**

When set, TXEN enables the GMAC transmitter to send data. When reset transmission will stop immediately, the transmit pipeline and control registers will be cleared and the Transmit Queue Pointer Register will reset to point to the start of the transmit descriptor list.

- **MPE: Management Port Enable**

Set to one to enable the management port. When zero, forces MDIO to high impedance state and MDC low.

- **CLRSTAT: Clear Statistics Registers**

This bit is write only. Writing a one clears the statistics registers.

- **INCSTAT: Increment Statistics Registers**

This bit is write only. Writing a one increments all the Statistics Registers by one for test purposes.

- **WESTAT: Write Enable for Statistics Registers**

Setting this bit to one makes the Statistics Registers writable for functional test purposes.

- **BP: Back pressure**

If set in 10M or 100M half duplex mode, forces collisions on all received frames.

- **TSTART: Start Transmission**

Writing one to this bit starts transmission.

- **THALT: Transmit Halt**

Writing one to this bit halts transmission as soon as any ongoing frame transmission ends.

- **TXPF: Transmit Pause Frame**

Writing one to this bit causes a pause frame to be transmitted.

- **TXZQPF: Transmit Zero Quantum Pause Frame**

Writing one to this bit causes a pause frame with zero quantum to be transmitted.

- **RDS: Read Snapshot**

Writing a one means that the snapshot value of the statistics register will be read back, otherwise the raw statistic register will be read. The default GMAC configuration does not support this function.

- **SRTSM: Store Receive Time Stamp to Memory**

0 = normal operation.

1 = causes the CRC of every received frame to be replaced with the value of the nanoseconds field of the 1588 timer that was captured as the receive frame passed the message time stamp point.

- **ENPBPR: Enable PFC Priority-based Pause Reception**

Enables PFC Priority Based Pause Reception capabilities. Setting this bit enables PFC negotiation and recognition of priority-based pause frames.

- **TXPBPF: Transmit PFC Priority-based Pause Frame**

Takes the values stored in the Transmit PFC Pause Register.

- **FNP: Flush Next Packet**

Flush the next packet from the external RX DPRAM. Writing one to this bit will only have an effect if the DMA is not currently writing a packet already stored in the DPRAM to memory.

#### 44.7.2 Network Configuration Register

**Name:** GMAC\_NCFGR

**Address:** 0x40034004

**Access:** Read-write

31	30	29	28	27	26	25	24
–	IRXER	RXBPP	IPGSEN	–	IRXFCS	EFRHD	RXCOEN
23	22	21	20	19	18	17	16
DCPF	DBW			CLK		RFCS	LFERD
15	14	13	12	11	10	9	8
RXBUFO	PEN	RTY	–	–	–	–	MAXFS
7	6	5	4	3	2	1	0
UNIHEN	MTI HEN	NBC	CAF	JFRAME	DNVLAN	FD	SPD

- **SPD: Speed**

Set to logic one to indicate 100 Mbps operation, logic zero for 10 Mbps.

- **FD: Full Duplex**

If set to logic one, the transmit block ignores the state of collision and carrier sense and allows receive while transmitting.

- **DNVLAN: Discard Non-VLAN FRAMES**

When set only VLAN tagged frames will be passed to the address matching logic.

- **JFRAME: Jumbo Frame Size**

Set to one to enable jumbo frames up to 10240 bytes to be accepted. The default length is 10240 bytes.

- **CAF: Copy All Frames**

When set to logic one, all valid frames will be accepted.

- **NBC: No Broadcast**

When set to logic one, frames addressed to the broadcast address of all ones will not be accepted.

- **MTIHEN: Multicast Hash Enable**

When set, multicast frames will be accepted when the 6-bit hash function of the destination address points to a bit that is set in the Hash Register.

- **UNIHEN: Unicast Hash Enable**

When set, unicast frames will be accepted when the 6-bit hash function of the destination address points to a bit that is set in the Hash Register.

- **MAXFS: 1536 Maximum Frame Size**

Setting this bit means the GMAC will accept frames up to 1536 bytes in length. Normally the GMAC would reject any frame above 1518 bytes.

- **RTY: Retry Test**

Must be set to zero for normal operation. If set to one the backoff between collisions will always be one slot time. Setting this bit to one helps test the too many retries condition. Also used in the pause frame tests to reduce the pause counter's decrement time from 512 bit times, to every GRXCK cycle.

- PEN: Pause Enable**

when set, transmission will pause if a non zero 802.3 classic pause frame is received and PFC has not been negotiated.

- RXBUTO: Receive Buffer Offset**

Indicates the number of bytes by which the received data is offset from the start of the receive buffer

- LFERD: Length Field Error Frame Discard**

Setting this bit causes frames with a measured length shorter than the extracted length field (as indicated by bytes 13 and 14 in a non-VLAN tagged frame) to be discarded. This only applies to frames with a length field less than 0x0600.

- RFCS: Remove FCS**

Setting this bit will cause received frames to be written to memory without their frame check sequence (last 4 bytes). The frame length indicated will be reduced by four bytes in this mode.

- CLK: MDC Clock Division**

Set according to MCK speed. These three bits determine the number MCK will be divided by to generate Management Data Clock (MDC). For conformance with the 802.3 specification, MDC must not exceed 2.5 MHz (MDC is only active during MDIO read and write operations).

Value	Name	Description
0	MCK_8	MCK divided by 8 (MCK up to 20 MHz)
1	MCK_16	MCK divided by 16 (MCK up to 40 MHz)
2	MCK_32	MCK divided by 32 (MCK up to 80 MHz)
3	MCK_48	MCK divided by 48 (MCK up to 120MHz)
4	MCK_64	MCK divided by 64 (MCK up to 160 MHz)
5	MCK_96	MCK divided by 96 (MCK up to 240 MHz)

- DBW: Data Bus Width**

Should be always written to 0.

- DCPF: Disable Copy of Pause Frames**

Set to one to prevent valid pause frames being copied to memory. When set, pause frames are not copied to memory regardless of the state of the copy all frames bit, whether a hash match is found or whether a type ID match is identified. If a destination address match is found, the pause frame will be copied to memory. Note that valid pause frames received will still increment pause statistics and pause the transmission of frames as required.

- RXCOEN: Receive Checksum Offload Enable**

When set, the receive checksum engine is enabled. Frames with bad IP, TCP or UDP checksums are discarded.

- EFRHD: Enable Frames Received in Half Duplex**

Enable frames to be received in half-duplex mode while transmitting.

- IRXFCS: Ignore RX FCS**

When set, frames with FCS/CRC errors will not be rejected. FCS error statistics will still be collected for frames with bad FCS and FCS status will be recorded in frame's DMA descriptor. For normal operation this bit must be set to zero.

- **IPGSEN: IP Stretch Enable**

When set, the transmit IPG can be increased above 96 bit times depending on the previous frame length using the IPG Stretch Register.

- **RXBP: Receive Bad Preamble**

When set, frames with non-standard preamble are not rejected.

- **IRXER: Ignore IPG GRXER**

When set, GRXER has no effect on the GMAC's operation when GRXDV is low.

#### 44.7.3 Network Status Register

**Name:** GMAC\_NSR

**Address:** 0x40034008

**Access:** Read-only

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
—	—	—	—	—	IDLE	MDIO	—

- **MDIO: MDIO Input Status**

Returns status of the MDIO pin.

- **IDLE: PHY Management Logic Idle**

The PHY management logic is idle (i.e. has completed).

#### 44.7.4 User Register

**Name:** GMAC\_UR  
**Address:** 0x4003400C  
**Access:** Read-write

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
-	-	-	-	-	-	-	MII

- **MII: MII Mode**

This bit must be set to 1.

Warning: The default value of this bit is 0.

#### 44.7.5 DMA Configuration Register

**Name:** GMAC\_DCFGR

**Address:** 0x40034010

**Access:** Read-write

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

- FBLDO: Fixed Burst Length for DMA Data Operations:**

Selects the burst length to attempt to use on the AHB when transferring frame data. Not used for DMA management operations and only used where space and data size allow. Otherwise SINGLE type AHB transfers are used.

Upper bits become non-writable if the configured DMA TX and RX FIFO sizes are smaller than required to support the selected burst size.

One-hot priority encoding enforced automatically on register writes as follows, where 'x' represents don't care:

Value	Name	Description
0	—	Reserved
1	SINGLE	00001: Always use SINGLE AHB bursts
2	—	Reserved
4	INCR4	001xx: Attempt to use INCR4 AHB bursts (Default)
8	INCR8	01xxx: Attempt to use INCR8 AHB bursts
16	INCR16	1xxxx: Attempt to use INCR16 AHB bursts

- ESMA: Endian Swap Mode Enable for Management Descriptor Accesses**

When set, selects swapped endianism for AHB transfers. When clear, selects little endian mode.

- ESPA: Endian Swap Mode Enable for Packet Data Accesses**

When set, selects swapped endianism for AHB transfers. When clear, selects little endian mode.

- TXCOEN: Transmitter Checksum Generation Offload Enable**

Transmitter IP, TCP and UDP checksum generation offload enable. When set, the transmitter checksum generation engine is enabled to calculate and substitute checksums for transmit frames. When clear, frame data is unaffected.

- DRBS: DMA Receive Buffer Size**

DMA receive buffer size in AHB system memory. The value defined by these bits determines the size of buffer to use in main AHB system memory when writing received data.

The value is defined in multiples of 64 bytes, thus a value of 0x01 corresponds to buffers of 64 bytes, 0x02 corresponds to 128 bytes, etc.

For example:

- 0x02: 128 bytes
- 0x18: 1536 bytes (1\*max length frame/buffer)
- 0xA0: 10240 bytes (1\*10K jumbo frame/buffer)

Note that this value should never be written as zero.

#### 44.7.6 Transmit Status Register

**Name:** GMAC\_TSR

**Address:** 0x40034014

**Access:** Read-write

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
—	UND	TXCOMP	TFC	TXGO	RLE	COL	UBR

- **UBR: Used Bit Read**

Set when a transmit buffer descriptor is read with its used bit set. Cleared by writing a one to this bit.

- **COL: Collision Occurred**

Set by the assertion of collision. Cleared by writing a one to this bit. When operating in 10/100 mode, this status indicates either a collision or a late collision.

- **RLE: Retry Limit Exceeded**

Cleared by writing a one to this bit.

- **TXGO: Transmit Go**

Transmit go, if high transmit is active. When using FIFO, this bit represents bit 3 of the Network Control Register. When using the DMA interface this bit represents the TXGO variable as specified in the transmit buffer description.

- **TFC: Transmit Frame Corruption due to AHB error**

Transmit frame corruption due to AHB error. Set if an error occurs while midway through reading transmit frame from the AHB, including HRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and GTXER asserted).

Cleared by writing a one to this bit.

- **TXCOMP: Transmit Complete**

Set when a frame has been transmitted. Cleared by writing a one to this bit.

- **UND: Transmit Under Run**

This bit is set if the transmitter was forced to terminate a frame that it had already began transmitting due to further data being unavailable.

This bit is set if a transmitter status write back has not completed when another status write back is attempted.

When using the DMA interface configured for internal FIFO mode, this bit is also set when the transmit DMA has written the SOP data into the FIFO and either the AHB bus was not granted in time for further data, or because an AHB not OK response was returned, or because a used bit was read.

- **HRESP: HRESP Not OK**

Set when the DMA block sees HRESP not OK. Cleared by writing a one to this bit.

#### 44.7.7 Receive Buffer Queue Base Address Register

**Name:** GMAC\_RBQB

**Address:** 0x40034018

**Access:** Read-write

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

This register holds the start address of the receive buffer queue (receive buffers descriptor list). The receive buffer queue base address must be initialized before receive is enabled through bit 2 of the Network Control Register. Once reception is enabled, any write to the Receive Buffer Queue Base Address Register is ignored. Reading this register returns the location of the descriptor currently being accessed. This value increments as buffers are used. Software should not use this register for determining where to remove received frames from the queue as it constantly changes as new frames are received. Software should instead work its way through the buffer descriptor queue checking the “used” bits.

In terms of AMBA AHB operation, the descriptors are read from memory using a single 32-bit AHB access. The descriptors should be aligned at 32-bit boundaries and the descriptors are written to using two individual non sequential accesses.

- **ADDR: Receive buffer queue base address**

Written with the address of the start of the receive queue.

#### 44.7.8 Transmit Buffer Queue Base Address Register

**Name:** GMAC\_TBQB

**Address:** 0x4003401C

**Access:** Read-write

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

This register holds the start address of the transmit buffer queue (transmit buffers descriptor list). The Transmit Buffer Queue Base Address Register must be initialized before transmit is started through bit 9 of the Network Control Register. Once transmission has started, any write to the Transmit Buffer Queue Base Address Register is illegal and therefore ignored.

Note that due to clock boundary synchronization, it takes a maximum of four MCK cycles from the writing of the transmit start bit before the transmitter is active. Writing to the Transmit Buffer Queue Base Address Register during this time may produce unpredictable results.

Reading this register returns the location of the descriptor currently being accessed. Since the DMA handles two frames at once, this may not necessarily be pointing to the current frame being transmitted.

In terms of AMBA AHB operation, the descriptors are written to memory using a single 32-bit AHB access. The descriptors should be aligned at 32-bit boundaries and the descriptors are read from memory using two individual non sequential accesses.

- **ADDR: Transmit Buffer Queue Base Address**

Written with the address of the start of the transmit queue.

#### 44.7.9 Receive Status Register

**Name:** GMAC\_RSR

**Address:** 0x40034020

**Access:** Read-write

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
—				HNO	RXOVR	REC	BNA

This register, when read provides details of the status of a receive. Once read, individual bits may be cleared by writing 1 to them. It is not possible to set a bit to 1 by writing to the register.

- **BNA: Buffer Not Available**

An attempt was made to get a new buffer and the pointer indicated that it was owned by the processor. The DMA will reread the pointer each time an end of frame is received until a valid pointer is found. This bit is set following each descriptor read attempt that fails, even if consecutive pointers are unsuccessful and software has in the mean time cleared the status flag. Cleared by writing a one to this bit.

- **REC: Frame Received**

One or more frames have been received and placed in memory. Cleared by writing a one to this bit.

- **RXOVR: Receive Overrun**

This bit is set if RX FIFO is not able to store the receive frame due to a FIFO overflow, or if the receive status was not taken at the end of the frame. For DMA operation, the buffer will be recovered if an overrun occurs. This bit is cleared when set to 1.

- **HNO: HRESP Not OK**

Set when the DMA block sees HRESP not OK. This bit is cleared when set to 1.

#### 44.7.10 Interrupt Status Register

**Name:** GMAC\_ISR

**Address:** 0x40034024

**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	WOL	–	SRI	PDRSFT	PDRQFT
23	22	21	20	19	18	17	16
PDRSFR	PDRQFR	SFT	DRQFT	SFR	DRQFR	–	–
15	14	13	12	11	10	9	8
–	PFTR	PTZ	PFNZ	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TFC	RLEX	TUR	TXUBR	RXUBR	RCOMP	MFS

This register indicates the source of the interrupt. In order the bits of this register are read to 1, the corresponding interrupt source must be enabled in the mask register. If any bit is set in this register, the GMAC interrupt signal will be asserted in the system.

- **MFS: Management Frame Sent**

The PHY Maintenance Register has completed its operation. Cleared on read.

- **RCOMP: Receive Complete**

A frame has been stored in memory. Cleared on read.

- **RXUBR: RX Used Bit Read**

Set when a receive buffer descriptor is read with its used bit set. Cleared on read.

- **TXUBR: TX Used Bit Read**

Set when a transmit buffer descriptor is read with its used bit set. Cleared on read.

- **TUR: Transmit Under Run**

This interrupt is set if the transmitter was forced to terminate a frame that it has already began transmitting due to further data being unavailable.

This interrupt is set if a transmitter status write back has not completed when another status write back is attempted.

This interrupt is also set when the transmit DMA has written the SOP data into the FIFO and either the AHB bus was not granted in time for further data, or because an AHB not OK response was returned, or because the used bit was read.

- **RLEX: Retry Limit Exceeded**

Transmit error. Cleared on read.

- **TFC: Transmit Frame Corruption due to AHB error**

Transmit frame corruption due to AHB error. Set if an error occurs while midway through reading transmit frame from the AHB, including HRESP errors and buffers exhausted mid frame.

- **TCOMP: Transmit Complete**

Set when a frame has been transmitted. Cleared on read.

- **ROVR: Receive Overrun**

Set when the receive overrun status bit is set. Cleared on read.

- **HRESP: HRESP Not OK**

Set when the DMA block sees HRESP not OK. Cleared on read.

- **PFNZ: Pause Frame with Non-zero Pause Quantum Received**

Indicates a valid pause has been received that has a non-zero pause quantum field. Cleared on read.

- **PTZ: Pause Time Zero**

Set when either the Pause Time Register at address 0x38 decrements to zero, or when a valid pause frame is received with a zero pause quantum field. Cleared on read.

- **PFTR: Pause Frame Transmitted**

Indicates a pause frame has been successfully transmitted after being initiated from the Network Control Register. Cleared on read.

- **DRQFR: PTP Delay Request Frame Received**

Indicates a PTP delay\_req frame has been received. Cleared on read.

- **SFR: PTP Sync Frame Received**

Indicates a PTP sync frame has been received. Cleared on read.

- **DRQFT: PTP Delay Request Frame Transmitted**

Indicates a PTP delay\_req frame has been transmitted. Cleared on read.

- **SFT: PTP Sync Frame Transmitted**

Indicates a PTP sync frame has been transmitted. Cleared on read.

- **PDRQFR: PDelay Request Frame Received**

Indicates a PTP pdelay\_req frame has been received. Cleared on read.

- **PDRSFR: PDelay Response Frame Received**

Indicates a PTP pdelay\_resp frame has been received. Cleared on read.

- **PDRQFT: PDelay Request Frame Transmitted**

Indicates a PTP pdelay\_req frame has been transmitted. Cleared on read.

- **PDRSFT: PDelay Response Frame Transmitted**

Indicates a PTP pdelay\_resp frame has been transmitted. Cleared on read.

- **SRI: TSU Seconds Register Increment**

Indicates the register has incremented. Cleared on read.

- **WOL: Wake On LAN**

WOL interrupt. Indicates a WOL event has been received.

#### 44.7.11 Interrupt Enable Register

**Name:** GMAC\_IER

**Address:** 0x40034028

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	WOL	–	SRI	PDRSFT	PDRQFT
23	22	21	20	19	18	17	16
PDRSFR	PDRQFR	SFT	DRQFT	SFR	DRQFR	–	–
15	14	13	12	11	10	9	8
EXINT	PFTR	PTZ	PFNZ	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TFC	RLEX	TUR	TXUBR	RXUBR	RCOMP	MFS

At reset all interrupts are disabled. Writing a one to the relevant bit location enables the required interrupt. This register is write only and when read will return zero.

- **MFS: Management Frame Sent**

Enable management done interrupt.

- **RCOMP: Receive Complete**

Enable receive complete interrupt.

- **RXUBR: RX Used Bit Read**

Enable receive used bit read interrupt.

- **TXUBR: TX Used Bit Read**

Enable transmit used bit read interrupt.

- **TUR: Transmit Under Run**

Enable transmit buffer under run interrupt.

- **RLEX: Retry Limit Exceeded or Late Collision**

Enable retry limit exceeded or late collision interrupt.

- **TFC: Transmit Frame Corruption due to AHB error**

Enable transmit frame corruption due to AHB error interrupt.

- **TCOMP: Transmit Complete**

Enable transmit complete interrupt.

- **ROVR: Receive Overrun**

Enable receive overrun interrupt.

- **HRESP: HRESP Not OK**

Enable HRESP not OK interrupt.

- **PFNZ: Pause Frame with Non-zero Pause Quantum Received**

Enable pause frame with non-zero pause quantum interrupt.

- **PTZ: Pause Time Zero**

Enable pause time zero interrupt.

- **PFTR: Pause Frame Transmitted**

Enable pause frame transmitted interrupt.

- **EXINT: External Interrupt**

Enable external interrupt.

- **DRQFR: PTP Delay Request Frame Received**

Enable PTP delay\_req frame received.

- **SFR: PTP Sync Frame Received**

Enable PTP sync frame received.

- **DRQFT: PTP Delay Request Frame Transmitted**

Enable PTP delay\_req frame transmitted.

- **SFT: PTP Sync Frame Transmitted**

Enable PTP sync frame transmitted.

- **PDRQFR: PDelay Request Frame Received**

Enable PTP pdelay\_req frame received

- **PDRSFR: PDelay Response Frame Received**

Enable PTP pdelay\_resp frame received.

- **PDRQFT: PDelay Request Frame Transmitted**

Enable PTP pdelay\_req frame transmitted.

- **PDRSFT: PDelay Response Frame Transmitted**

Enable PTP pdelay\_resp frame transmitted.

- **SRI: TSU Seconds Register Increment**

Enable TSU Seconds Register increment.

- **WOL: Wake On LAN**

Enable WOL event received interrupt.

#### 44.7.12 Interrupt Disable Register

**Name:** GMAC\_IDR

**Address:** 0x4003402C

**Access:** Write-only

31	30	29	28	27	26	25	24
–	–	–	WOL	–	SRI	PDRSFT	PDRQFT
23	22	21	20	19	18	17	16
PDRSFR	PDRQFR	SFT	DRQFT	SFR	DRQFR	–	–
15	14	13	12	11	10	9	8
EXINT	PFTR	PTZ	PFNZ	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TFC	RLEX	TUR	TXUBR	RXUBR	RCOMP	MFS

Writing a 1 to the relevant bit location disables that particular interrupt. This register is write only and when read will return zero.

- **MFS: Management Frame Sent**

Disable management done interrupt.

- **RCOMP: Receive Complete**

Disable receive complete interrupt.

- **RXUBR: RX Used Bit Read**

Disable receive used bit read interrupt.

- **TXUBR: TX Used Bit Read**

Disable transmit used bit read interrupt.

- **TUR: Transmit Under Run**

Disable transmit buffer under run interrupt.

- **RLEX: Retry Limit Exceeded or Late Collision**

Disable retry limit exceeded or late collision interrupt.

- **TFC: Transmit Frame Corruption due to AHB error**

Disable transmit frame corruption due to AHB error interrupt.

- **TCOMP: Transmit Complete**

Disable transmit complete interrupt.

- **ROVR: Receive Overrun**

Disable receive overrun interrupt.

- **HRESP: HRESP Not OK**

Disable HRESP not OK interrupt.

- **PFNZ: Pause Frame with Non-zero Pause Quantum Received**

Disable pause frame with non-zero pause quantum interrupt.

- **PTZ: Pause Time Zero**

Disable pause time zero interrupt.

- **PFTR: Pause Frame Transmitted**

Disable pause frame transmitted interrupt.

- **EXINT: External Interrupt**

Disable external interrupt.

- **DRQFR: PTP Delay Request Frame Received**

Disable PTP delay\_req frame received.

- **SFR: PTP Sync Frame Received**

Disable PTP sync frame received.

- **DRQFT: PTP Delay Request Frame Transmitted**

Disable PTP delay\_req frame transmitted.

- **SFT: PTP Sync Frame Transmitted**

Disable PTP sync frame transmitted.

- **PDRQFR: PDelay Request Frame Received**

Disable PTP pdelay\_req frame received

- **PDRSFR: PDelay Response Frame Received**

Disable PTP pdelay\_resp frame received.

- **PDRQFT: PDelay Request Frame Transmitted**

Disable PTP pdelay\_req frame transmitted.

- **PDRSFT: PDelay Response Frame Transmitted**

Disable PTP pdelay\_resp frame transmitted.

- **SRI: TSU Seconds Register Increment**

Disable TSU Seconds Register increment.

- **WOL: Wake On LAN**

Disable WOL event received interrupt.

#### 44.7.13 Interrupt Mask Register

**Name:** GMAC\_IMR

**Address:** 0x40034030

**Access:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	PDRSFT	PDRQFT
23	22	21	20	19	18	17	16
PDRSFR	PDRQFR	SFT	DRQFT	SFR	DRQFR	–	–
15	14	13	12	11	10	9	8
EXINT	PFTR	PTZ	PFNZ	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TFC	RLEX	TUR	TXUBR	RXUBR	RCOMP	MFS

The Interrupt Mask Register is a read-only register indicating which interrupts are masked. All bits are set at reset and can be reset individually by writing to the Interrupt Enable Register or set individually by writing to the Interrupt Disable Register. Having separate address locations for enable and disable saves the need for performing a read modify write when updating the Interrupt Mask Register.

For test purposes there is a write-only function to this register that allows the bits in the Interrupt Status Register to be set or cleared, regardless of the state of the mask register.

- **MFS: Management Frame Sent**

A read of this register returns the value of the management done interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **RCOMP: Receive Complete**

A read of this register returns the value of the receive complete interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **RXUBR: RX Used Bit Read**

A read of this register returns the value of the receive used bit read interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **TXUBR: TX Used Bit Read**

A read of this register returns the value of the transmit used bit read interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **TUR: Transmit Under Run**

A read of this register returns the value of the transmit buffer under run interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **RLEX: Retry Limit Exceeded**

A read of this register returns the value of the retry limit exceeded interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **TFC: Transmit Frame Corruption due to AHB error**

A read of this register returns the value of the transmit frame corruption due to AHB error interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **TCOMP: Transmit Complete**

A read of this register returns the value of the transmit complete interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **ROVR: Receive Overrun**

A read of this register returns the value of the receive overrun interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **HRESP: HRESP Not OK**

A read of this register returns the value of the HRESP not OK interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **PFNZ: Pause Frame with Non-zero Pause Quantum Received**

A read of this register returns the value of the pause frame with non-zero pause quantum interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **PTZ: Pause Time Zero**

A read of this register returns the value of the pause time zero interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **PFTR: Pause Frame Transmitted**

A read of this register returns the value of the pause frame transmitted interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **EXINT: External Interrupt**

A read of this register returns the value of the external interrupt mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **DRQFR: PTP Delay Request Frame Received**

A read of this register returns the value of the PTP delay\_req frame received mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **SFR: PTP Sync Frame Received**

A read of this register returns the value of the PTP sync frame received mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **DRQFT: PTP Delay Request Frame Transmitted**

A read of this register returns the value of the PTP delay\_req frame transmitted mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **SFT: PTP Sync Frame Transmitted**

A read of this register returns the value of the PTP sync frame transmitted mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **PDRQFR: PDelay Request Frame Received**

A read of this register returns the value of the PTP pdelay\_req frame received mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **PDRSFR: PDelay Response Frame Received**

A read of this register returns the value of the PTP pdelay\_resp frame received mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **PDRQFT: PDelay Request Frame Transmitted**

A read of this register returns the value of the PTP pdelay\_req frame transmitted mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

- **PDRSFT: PDelay Response Frame Transmitted**

A read of this register returns the value of the PTP pdelay\_resp frame transmitted mask.

0 = Interrupt is enabled.

1 = Interrupt is disabled.

A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

#### 44.7.14 PHY Maintenance Register

**Name:** GMAC\_MAN

**Address:** 0x40034034

**Access:** Read-write

31	30	29	28	27	26	25	24
WZO	CLTTO		OP			PHYA	
23	22	21	20	19	18	17	16
PHYA			REGA			WTN	
15	14	13	12	11	10	9	8
			DATA				
7	6	5	4	3	2	1	0
			DATA				

The PHY Maintenance Register is implemented as a shift register. Writing to the register starts a shift operation which is signalled as complete when bit 2 is set in the Network Status Register. It takes about 2000 MCK cycles to complete, when MDC is set for MCK divide by 32 in the Network Configuration Register. An interrupt is generated upon completion.

During this time, the MSB of the register is output on the MDIO pin and the LSB updated from the MDIO pin with each MDC cycle. This causes transmission of a PHY management frame on MDIO. See [Section 22.2.4.5 of the IEEE 802.3 standard](#).

Reading during the shift operation will return the current contents of the shift register. At the end of management operation, the bits will have shifted back to their original locations. For a read operation, the data bits will be updated with data read from the PHY. It is important to write the correct values to the register to ensure a valid PHY management frame is produced.

The MDIO interface can read IEEE 802.3 clause 45 PHYs as well as clause 22 PHYs. To read clause 45 PHYs, bit 30 should be written with a 0 rather than a 1.

For a description of MDC generation, see [Section 44.7.2 "Network Configuration Register"](#).

- **DATA: PHY Data**

For a write operation this field is written with the data to be written to the PHY. After a read operation this field contains the data read from the PHY.

- **WTN: Write Ten**

Must be written to 10.

- **REGA: Register Address**

Specifies the register in the PHY to access.

- **PHYA: PHY Address**

- **OP: Operation**

10 is read. 01 is write.

- **CLTTO: Clause 22 Operation**

Must be written to 1 for Clause 22 operation. To read clause 45 PHYs, bit 30 should be written with a 0 rather than a 1.

- **WZO: Write ZERO**

Must be written with 0.

#### 44.7.15 Receive Pause Quantum Register

**Name:** GMAC\_RPQ

**Address:** 0x40034038

**Access:** Read-only

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

- **RPQ: Received Pause Quantum**

Stores the current value of the Receive Pause Quantum Register which is decremented every 512 bit times.

#### 44.7.16 Transmit Pause Quantum Register

**Name:** GMAC\_TPQ

**Address:** 0x4003403C

**Access:** Read-write

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

- **TPQ: Transmit Pause Quantum**

Written with the pause quantum value for pause frame transmission.

#### 44.7.17 Hash Register Bottom [31:0]

**Name:** GMAC\_HRB

**Address:** 0x40034080

**Access:** Read-only

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

The unicast hash enable (UNIHEN) and the multicast hash enable (MITIHEN) bits in the Network Configuration Register ([Section 44.7.2 "Network Configuration Register"](#)) enable the reception of hash matched frames. See [Section 44.5.9 "Hash Addressing"](#).

- **ADDR: Hash Address**

The first 32 bits of the Hash Address Register.

#### 44.7.18 Hash Register Top [63:32]

**Name:** GMAC\_HRT

**Address:** 0x40034084

**Access:** Read-only

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

The unicast hash enable (UNIHEN) and the multicast hash enable (MITIHEN) bits in the ["Network Configuration Register"](#) enable the reception of hash matched frames. See [Section 44.5.9 "Hash Addressing"](#).

- **ADDR: Hash Address**

Bits 63 to 32 of the Hash Address Register.

#### 44.7.19 Specific Address 1 Bottom [31:0] Register

**Name:** GMAC\_SAB1

**Address:** 0x40034088

**Access:** Read-write

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

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 1**

Least significant 32 bits of the destination address, that is bits 31:0. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

#### 44.7.20 Specific Address 1 Top [47:32] Register

**Name:** GMAC\_SAT1

**Address:** 0x4003408C

**Access:** Read-write

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

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 1**

The most significant bits of the destination address, that is bits 47:32.

#### 44.7.21 Specific Address 2 Bottom [31:0] Register

**Name:** GMAC\_SAB2

**Address:** 0x40034090

**Access:** Read-write

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

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 2**

Least significant 32 bits of the destination address, that is bits 31:0. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

#### 44.7.22 Specific Address 2 Top [47:32] Register

**Name:** GMAC\_SAT2

**Address:** 0x40034094

**Access:** Read-write

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

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 2**

The most significant bits of the destination address, that is bits 47:32.

#### 44.7.23 Specific Address 3 Bottom [31:0] Register

**Name:** GMAC\_SAB3

**Address:** 0x40034098

**Access:** Read-write

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

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 3**

Least significant 32 bits of the destination address, that is bits 31:0. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

#### 44.7.24 Specific Address 3 Top [47:32] Register

**Name:** GMAC\_SAT3

**Address:** 0x4003409C

**Access:** Read-write

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

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 3**

The most significant bits of the destination address, that is bits 47:32.

#### 44.7.25 Specific Address 4 Bottom Register[31:0]

**Name:** GMAC\_SAB4

**Address:** 0x400340A0

**Access:** Read-write

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

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 4**

Least significant 32 bits of the destination address, that is bits 31:0. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

#### 44.7.26 Specific Address 4 Top Register[47:32]

**Name:** GMAC\_SAT4

**Address:** 0x400340A4

**Access:** Read-write

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

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 4**

The most significant bits of the destination address, that is bits 47:32.

#### 44.7.27 Type ID Match 1 Register

**Name:** GMAC\_TIDM1

**Access:** Read-write

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

- **TID: Type ID Match 1**

For use in comparisons with received frames type ID/length frames.

#### 44.7.28 Type ID Match 2 Register

**Name:** GMAC\_TIDM2

**Access:** Read-write

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

- **TID: Type ID Match 2**

For use in comparisons with received frames type ID/length frames.

#### 44.7.29 Type ID Match 3 Register

**Name:** GMAC\_TIDM3

**Access:** Read-write

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

- **TID: Type ID Match 3**

For use in comparisons with received frames type ID/length frames.

#### 44.7.30 Type ID Match 4 Register

**Name:** GMAC\_TIDM4

**Access:** Read-write

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

- **TID: Type ID Match 4**

For use in comparisons with received frames type ID/length frames.

#### 44.7.31 IPG Stretch Register

**Name:** GMAC\_IPGS

**Address:** 0x400340BC

**Access:** Read-write

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

- **FL: Frame Length**

Bits 7:0 are multiplied with the previously transmitted frame length (including preamble). Bits 15:8 +1 divide the frame length. If the resulting number is greater than 96 and bit 28 is set in the Network Configuration Register then the resulting number is used for the transmit inter-packet-gap. 1 is added to bits 15:8 to prevent a divide by zero. See [Section 44.5.4 "MAC Transmit Block"](#).

#### 44.7.32 Stacked VLAN Register

**Name:** GMAC\_SVLAN

**Address:** 0x400340C0

**Access:** Read-write

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

- **VLAN\_TYPE: User Defined VLAN\_TYPE Field**

User defined VLAN\_TYPE field. When Stacked VLAN is enabled, the first VLAN tag in a received frame will only be accepted if the VLAN type field is equal to this user defined VLAN\_TYPE, OR equal to the standard VLAN type (0x8100). Note that the second VLAN tag of a Stacked VLAN packet will only be matched correctly if its VLAN\_TYPE field equals 0x8100.

- **ESVLAN: Enable Stacked VLAN Processing Mode**

#### 44.7.33 Transmit PFC Pause Register

**Name:** GMAC\_TPFCP

**Address:** 0x400340C4

**Access:** Read-write

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

- **PEV: Priority Enable Vector**

If bit 17 of the Network Control Register is written with a one then the priority enable vector of the PFC priority based pause frame will be set equal to the value stored in this register [7:0].

- **PQ: Pause Quantum**

If bit 17 of the Network Control Register is written with a one then for each entry equal to zero in the Transmit PFC Pause Register[15:8], the PFC pause frame's pause quantum field associated with that entry will be taken from the Transmit Pause Quantum Register. For each entry equal to one in the Transmit PFC Pause Register [15:8], the pause quantum associated with that entry will be zero.

#### 44.7.34 Specific Address 1 Mask Bottom [31:0] Register

**Name:** GMAC\_SAMB1

**Address:** 0x400340C8

**Access:** Read-write

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

- **ADDR: Specific Address 1 Mask**

Setting a bit to one masks the corresponding bit in the Specific Address 1 Register.

#### 44.7.35 Specific Address Mask 1 Top [47:32] Register

**Name:** GMAC\_SAMT1

**Address:** 0x400340CC

**Access:** Read-write

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

- **ADDR: Specific Address 1 Mask**

Setting a bit to one masks the corresponding bit in the Specific Address 1 Register.

#### 44.7.36 Octets Transmitted [31:0] Register

**Name:** GMAC\_OTLO

**Address:** 0x40034100

**Access:** Read-only

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

When reading the Octets Transmitted and Octets Received Registers, bits 31:0 should be read prior to bits 47:32 to ensure reliable operation.

- **TXO: Transmitted Octets**

Transmitted octets in frame without errors [31:0]. The number of octets transmitted in valid frames of any type. This counter is 48-bits, and is read through two registers. This count does not include octets from automatically generated pause frames.

#### 44.7.37 Octets Transmitted [47:32] Register

**Name:** GMAC\_OTH1

**Address:** 0x40034104

**Access:** Read-only

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

When reading the Octets Transmitted and Octets Received Registers, bits 31:0 should be read prior to bits 47:32 to ensure reliable operation.

- **TXO: Transmitted Octets**

Transmitted octets in frame without errors [47:32]. The number of octets transmitted in valid frames of any type. This counter is 48-bits, and is read through two registers. This count does not include octets from automatically generated pause frames.

#### 44.7.38 Frames Transmitted Register

**Name:** GMAC\_FT

**Address:** 0x40034108

**Access:** Read-only

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

- **FTX: Frames Transmitted without Error**

Frames transmitted without error. This register counts the number of frames successfully transmitted, i.e., no under run and not too many retries. Excludes pause frames.

#### 44.7.39 Broadcast Frames Transmitted Register

**Name:** GMAC\_BCFT

**Address:** 0x4003410C

**Access:** Read-only

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

- **BFTX: Broadcast Frames Transmitted without Error**

Broadcast frames transmitted without error. This register counts the number of broadcast frames successfully transmitted without error, i.e., no under run and not too many retries. Excludes pause frames.

#### 44.7.40 Multicast Frames Transmitted Register

**Name:** GMAC\_MFT

**Address:** 0x40034110

**Access:** Read-only

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

- **MFTX: Multicast Frames Transmitted without Error**

This register counts the number of multicast frames successfully transmitted without error, i.e., no under run and not too many retries. Excludes pause frames.

#### 44.7.41 Pause Frames Transmitted Register

**Name:** GMAC\_PFT

**Address:** 0x40034114

**Access:** Read-only

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

- **PFTX: Pause Frames Transmitted Register**

This register counts the number of pause frames transmitted. Only pause frames triggered by the register interface or through the external pause pins are counted as pause frames. Pause frames received through FIFO are counted in the frames transmitted counter.

#### 44.7.42 64 Byte Frames Transmitted Register

**Name:** GMAC\_BFT64

**Address:** 0x40034118

**Access:** Read-only

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

- **NFTX: 64 Byte Frames Transmitted without Error**

This register counts the number of 64 byte frames successfully transmitted without error, i.e., no under run and not too many retries. Excludes pause frames.

#### 44.7.43 65 to 127 Byte Frames Transmitted Register

**Name:** GMAC\_TBFT127

**Address:** 0x4003411C

**Access:** Read-only

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

- **NFTX: 65 to 127 Byte Frames Transmitted without Error**

This register counts the number of 65 to 127 byte frames successfully transmitted without error, i.e., no under run and not too many retries. Excludes pause frames.

#### 44.7.44 128 to 255 Byte Frames Transmitted Register

**Name:** GMAC\_TBFT255

**Address:** 0x40034120

**Access:** Read-only

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

- **NFTX: 128 to 255 Byte Frames Transmitted without Error**

This register counts the number of 128 to 255 byte frames successfully transmitted without error, i.e., no under run and not too many retries.

#### 44.7.45 256 to 511 Byte Frames Transmitted Register

**Name:** GMAC\_TBFT511

**Address:** 0x40034124

**Access:** Read-only

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

- **NFTX: 256 to 511 Byte Frames Transmitted without Error**

This register counts the number of 256 to 511 byte frames successfully transmitted without error, i.e., no under run and not too many retries.

#### 44.7.46 512 to 1023 Byte Frames Transmitted Register

**Name:** GMAC\_TBFT1023

**Address:** 0x40034128

**Access:** Read-only

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

- **NFTX: 512 to 1023 Byte Frames Transmitted without Error**

This register counts the number of 512 to 1023 byte frames successfully transmitted without error, i.e., no under run and not too many retries.

#### 44.7.47 1024 to 1518 Byte Frames Transmitted Register

**Name:** GMAC\_TBFT1518

**Address:** 0x4003412C

**Access:** Read-only

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

- **NFTX: 1024 to 1518 Byte Frames Transmitted without Error**

This register counts the number of 1024 to 1518 byte frames successfully transmitted without error, i.e., no under run and not too many retries.

#### 44.7.48 Greater Than 1518 Byte Frames Transmitted Register

**Name:** GMAC\_GTBFT1518

**Address:** 0x40034130

**Access:** Read-only

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

- **NFTX: Greater than 1518 Byte Frames Transmitted without Error**

This register counts the number of 1518 or above byte frames successfully transmitted without error i.e., no under run and not too many retries.

#### 44.7.49 Transmit Under Runs Register

**Name:** GMAC\_TUR

**Address:** 0x40034134

**Access:** Read-only

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

- **TXUNR: Transmit Under Runs**

This register counts the number of frames not transmitted due to a transmit under run. If this register is incremented then no other statistics register is incremented.

#### 44.7.50 Single Collision Frames Register

**Name:** GMAC\_SCF

**Address:** 0x40034138

**Access:** Read-only

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

- **SCOL: Single Collision**

This register counts the number of frames experiencing a single collision before being successfully transmitted i.e., no under run.

#### 44.7.51 Multiple Collision Frames Register

**Name:** GMAC\_MCF

**Address:** 0x4003413C

**Access:** Read-only

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

- **MCOL: Multiple Collision**

This register counts the number of frames experiencing between two and fifteen collisions prior to being successfully transmitted, i.e., no under run and not too many retries.

#### 44.7.52 Excessive Collisions Register

**Name:** GMAC\_EC

**Address:** 0x40034140

**Access:** Read-only

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

- **XCOL: Excessive Collisions**

This register counts the number of frames that failed to be transmitted because they experienced 16 collisions.

#### 44.7.53 Late Collisions Register

**Name:** GMAC\_LC

**Address:** 0x40034144

**Access:** Read-only

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

- **LCOL: Late Collisions**

This register counts the number of late collisions occurring after the slot time (512 bits) has expired. In 10/100 mode, late collisions are counted twice i.e., both as a collision and a late collision.

#### 44.7.54 Deferred Transmission Frames Register

**Name:** GMAC\_DTF

**Address:** 0x40034148

**Access:** Read-only

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

- **DEFT: Deferred Transmission**

This register counts the number of frames experiencing deferral due to carrier sense being active on their first attempt at transmission. Frames involved in any collision are not counted nor are frames that experienced a transmit under run.

#### 44.7.55 Carrier Sense Errors Register

**Name:** GMAC\_CSE

**Address:** 0x4003414C

**Access:** Read-only

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

- **CSR: Carrier Sense Error**

This register counts the number of frames transmitted where carrier sense was not seen during transmission or where carrier sense was deasserted after being asserted in a transmit frame without collision (no under run). Only incremented in half duplex mode. The only effect of a carrier sense error is to increment this register. The behavior of the other Statistics Registers is unaffected by the detection of a carrier sense error.

#### 44.7.56 Octets Received [31:0] Register

**Name:** GMAC\_ORLO

**Address:** 0x40034150

**Access:** Read-only

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

When reading the Octets Transmitted and Octets Received Registers, bits [31:0] should be read prior to bits [47:32] to ensure reliable operation.

- **RXO: Received Octets**

Received octets in frame without errors [31:0]. The number of octets received in valid frames of any type. This counter is 48-bits and is read through two registers. This count does not include octets from pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.57 Octets Received [47:32] Register

**Name:** GMAC\_ORHI

**Address:** 0x40034154

**Access:** Read-only

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

When reading the Octets Transmitted and Octets Received Registers, bits 31:0 should be read prior to bits 47:32 to ensure reliable operation.

- **RXO: Received Octets**

Received octets in frame without errors [47:32]. The number of octets received in valid frames of any type. This counter is 48-bits and is read through two registers. This count does not include octets from pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.58 Frames Received Register

**Name:** GMAC\_FR

**Address:** 0x40034158

**Access:** Read-only

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

- **FRX: Frames Received without Error**

Frames received without error. This register counts the number of frames successfully received. Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.59 Broadcast Frames Received Register

**Name:** GMAC\_BCFR

**Address:** 0x4003415C

**Access:** Read-only

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

- **BFRX: Broadcast Frames Received without Error**

Broadcast frames received without error. This register counts the number of broadcast frames successfully received. Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.60 Multicast Frames Received Register

**Name:** GMAC\_MFR

**Address:** 0x40034160

**Access:** Read-only

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

- **MFRX: Multicast Frames Received without Error**

This register counts the number of multicast frames successfully received without error. Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.61 Pause Frames Received Register

**Name:** GMAC\_PFR

**Address:** 0x40034164

**Access:** Read-only

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

- **PFRX: Pause Frames Received Register**

This register counts the number of pause frames received without error.

#### 44.7.62 64 Byte Frames Received Register

**Name:** GMAC\_BFR64

**Address:** 0x40034168

**Access:** Read-only

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

- **NFRX: 64 Byte Frames Received without Error**

This register counts the number of 64 byte frames successfully received without error. Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.63 65 to 127 Byte Frames Received Register

**Name:** GMAC\_TBFR127

**Address:** 0x4003416C

**Access:** Read-only

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

- **NFRX: 65 to 127 Byte Frames Received without Error**

This register counts the number of 65 to 127 byte frames successfully received without error. Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.64 128 to 255 Byte Frames Received Register

**Name:** GMAC\_TBFR255

**Address:** 0x40034170

**Access:** Read-only

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

- **NFRX: 128 to 255 Byte Frames Received without Error**

This register counts the number of 128 to 255 byte frames successfully received without error. Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.65 256 to 511 Byte Frames Received Register

**Name:** GMAC\_TBFR511

**Address:** 0x40034174

**Access:** Read-only

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

- **NFRX: 256 to 511 Byte Frames Received without Error**

This register counts the number of 256 to 511 byte frames successfully received without error. Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.66 512 to 1023 Byte Frames Received Register

**Name:** GMAC\_TBFR1023

**Address:** 0x40034178

**Access:** Read-only

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

- **NFRX: 512 to 1023 Byte Frames Received without Error**

This register counts the number of 512 to 1023 byte frames successfully received without error. Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory.

#### 44.7.67 1024 to 1518 Byte Frames Received Register

**Name:** GMAC\_TBFR1518

**Address:** 0x4003417C

**Access:** Read-only

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

- **NFRX: 1024 to 1518 Byte Frames Received without Error**

This register counts the number of 1024 to 1518 byte frames successfully received without error, i.e., no under run and not too many retries.

#### 44.7.68 1519 to Maximum Byte Frames Received Register

**Name:** GMAC\_TMXBFR

**Address:** 0x40034180

**Access:** Read-only

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

- **NFRX: 1519 to Maximum Byte Frames Received without Error**

This register counts the number of 1519 byte or above frames successfully received without error. Maximum frame size is determined by the Network Configuration Register bit 8 (1536 maximum frame size) or bit 3 (jumbo frame size). Excludes pause frames, and is only incremented if the frame is successfully filtered and copied to memory. See: [Section 44.7.2 "Network Configuration Register"](#).

#### 44.7.69 Undersized Frames Received Register

**Name:** GMAC\_UFR

**Address:** 0x40034184

**Access:** Read-only

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

- **UFRX: Undersize Frames Received**

This register counts the number of frames received less than 64 bytes in length (10/100 mode, full duplex) that do not have either a CRC error or an alignment error.

#### 44.7.70 Oversized Frames Received Register

**Name:** GMAC\_OFR

**Address:** 0x40034188

**Access:** Read-only

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

- **OFRX: Oversized Frames Received**

This register counts the number of frames received exceeding 1518 bytes (1536 bytes if bit 8 is set in the Network Configuration Register) in length but do not have either a CRC error, an alignment error nor a receive symbol error. See [Section 44.7.2 "Network Configuration Register"](#).

#### 44.7.71 Jabbers Received Register

**Name:** GMAC\_JR  
**Address:** 0x4003418C  
**Access:** Read-only

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

- **JRX: Jabbers Received**

The register counts the number of frames received exceeding 1518 bytes in length (1536 if bit 8 is set in Network Configuration Register) and have either a CRC error, an alignment error or a receive symbol error. See: [Section 44.7.2 "Network Configuration Register"](#).

#### 44.7.72 Frame Check Sequence Errors Register

**Name:** GMAC\_FCSE  
**Address:** 0x40034190  
**Access:** Read-only

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

- **FCKR: Frame Check Sequence Errors**

The register counts frames that are an integral number of bytes, have bad CRC and are between 64 and 1518 bytes in length (1536 if bit 8 is set in Network Configuration Register). This register is also incremented if a symbol error is detected and the frame is of valid length and has an integral number of bytes.

This register is incremented for a frame with bad FCS, regardless of whether it is copied to memory due to ignore FCS mode being enabled in bit 26 of the Network Configuration Register. See: [Section 44.7.2 "Network Configuration Register"](#).

#### 44.7.73 Length Field Frame Errors Register

**Name:** GMAC\_LFFE

**Address:** 0x40034194

**Access:** Read-only

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

- **LFER: Length Field Frame Errors**

This register counts the number of frames received that have a measured length shorter than that extracted from the length field (bytes 13 and 14). This condition is only counted if the value of the length field is less than 0x0600, the frame is not of excessive length and checking is enabled through bit 16 of the Network Configuration Register. See: [Section 44.7.2 "Network Configuration Register"](#).

#### 44.7.74 Receive Symbol Errors Register

**Name:** GMAC\_RSE

**Address:** 0x40034198

**Access:** Read-only

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

- **RXSE: Receive Symbol Errors**

This register counts the number of frames that had GRXER asserted during reception. For 10/100 mode symbol errors are counted regardless of frame length checks. Receive symbol errors will also be counted as an FCS or alignment error if the frame is between 64 and 1518 bytes (1536 bytes if bit 8 is set in the Network Configuration Register). If the frame is larger it will be recorded as a jabber error. See [Section 44.7.2 "Network Configuration Register"](#).

#### 44.7.75 Alignment Errors Register

**Name:** GMAC\_AE  
**Address:** 0x4003419C  
**Access:** Read-only

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

- **AER: Alignment Errors**

This register counts the frames that are not an integral number of bytes long and have bad CRC when their length is truncated to an integral number of bytes and are between 64 and 1518 bytes in length (1536 if bit 8 is set in Network Configuration Register). This register is also incremented if a symbol error is detected and the frame is of valid length and does not have an integral number of bytes. See: [Section 44.7.2 "Network Configuration Register"](#).

#### 44.7.76 Receive Resource Errors Register

**Name:** GMAC\_RRE  
**Address:** 0x400341A0  
**Access:** Read-only

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

- **RXRER: Receive Resource Errors**

This register counts frames that are not an integral number of bytes long and have bad CRC when their length is truncated to an integral number of bytes and are between 64 and 1518 bytes in length (1536 if bit 8 is set in Network Configuration Register). This register is also incremented if a symbol error is detected and the frame is of valid length and does not have an integral number of bytes. See: [Section 44.7.2 "Network Configuration Register"](#).

#### 44.7.77 Receive Overruns Register

**Name:** GMAC\_ROE

**Address:** 0x400341A4

**Access:** Read-only

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

- **RXOVR: Receive Overruns**

This register counts the number of frames that are address recognized but were not copied to memory due to a receive overrun.

#### 44.7.78 IP Header Checksum Errors Register

**Name:** GMAC\_IHCE

**Address:** 0x400341A8

**Access:** Read-only

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
HCKER							

- **HCKER: IP Header Checksum Errors**

This register counts the number of frames discarded due to an incorrect IP header checksum, but are between 64 and 1518 bytes (1536 bytes if bit 8 is set in the Network Configuration Register) and do not have a CRC error, an alignment error, nor a symbol error.

#### 44.7.79 TCP Checksum Errors Register

**Name:** GMAC\_TCE

**Address:** 0x400341AC

**Access:** Read-only

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
TCKER							

- **TCKER: TCP Checksum Errors**

This register counts the number of frames discarded due to an incorrect TCP checksum, but are between 64 and 1518 bytes (1536 bytes if bit 8 is set in the Network Configuration Register) and do not have a CRC error, an alignment error, nor a symbol error.

#### 44.7.80 UDP Checksum Errors Register

**Name:** GMAC\_UCE

**Address:** 0x400341B0

**Access:** Read-only

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
UCKER							

- **UCKER: UDP Checksum Errors**

This register counts the number of frames discarded due to an incorrect UDP checksum, but are between 64 and 1518 bytes (1536 bytes if bit 8 is set in the Network Configuration Register) and do not have a CRC error, an alignment error, nor a symbol error.

#### 44.7.81 1588 Timer Sync Strobe Seconds Register

**Name:** GMAC\_TSSS

**Address:** 0x400341C8

**Access:** Read-write

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

- **VTS: Value of Timer Seconds Register Capture**

The value of the Timer Seconds Register is captured.

#### 44.7.82 1588 Timer Sync Strobe Nanoseconds Register

**Name:** GMAC\_TSSN

**Address:** 0x400341CC

**Access:** Read-write

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

- **VTN: Value Timer Nanoseconds Register Capture**

The value of the Timer Nanoseconds Register is captured.

#### 44.7.83 1588 Timer Seconds Register

**Name:** GMAC\_TS  
**Address:** 0x400341D0  
**Access:** Read-write

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

- **TCS: Timer Count in Seconds**

This register is writable. It increments by one when the 1588 nanoseconds counter counts to one second. It may also be incremented when the Timer Adjust Register is written.

#### 44.7.84 1588 Timer Nanoseconds Register

**Name:** GMAC\_TN  
**Address:** 0x400341D4  
**Access:** Read-write

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

- **TNS: Timer Count in Nanoseconds**

This register is writable. It can also be adjusted by writes to the 1588 Timer Adjust Register. It increments by the value of the 1588 Timer Increment Register each clock cycle.

#### 44.7.85 1588 Timer Adjust Register

**Name:** GMAC\_TA

**Address:** 0x400341D8

**Access:** Write-only

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

- **ITDT: Increment/Decrement**

The number of nanoseconds to increment or decrement the 1588 Timer Nanoseconds Register. If necessary, the 1588 Seconds Register will be incremented or decremented.

- **ADJ: Adjust 1588 Timer**

Write as one to subtract from the 1588 timer. Write as zero to add to it.

#### 44.7.86 1588 Timer Increment Register

**Name:** GMAC\_TI  
**Address:** 0x400341DC  
**Access:** Read-write

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

- **CNS: Count Nanoseconds**

A count of nanoseconds by which the 1588 Timer Nanoseconds Register will be incremented each clock cycle.

- **ACNS: Alternative Count Nanoseconds**

Alternative count of nanoseconds by which the 1588 Timer Nanoseconds Register will be incremented each clock cycle.

- **NIT: Number of Increments**

The number of increments after which the alternative increment is used.

#### 44.7.87 PTP Event Frame Transmitted Seconds Register

**Name:** GMAC\_EFTS

**Address:** 0x400341E0

**Access:** Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP transmit peer event crosses the MII interface. An interrupt is issued when the register is updated.

#### 44.7.88 PTP Event Frame Transmitted Nanoseconds Register

**Name:** GMAC\_EFTN

**Address:** 0x400341E4

**Access:** Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Nanoseconds Register holds when the SFD of a PTP transmit peer event crosses the MII interface. An interrupt is issued when the register is updated.

#### 44.7.89 PTP Event Frame Received Seconds Register

**Name:** GMAC\_EFRS

**Address:** 0x400341E8

**Access:** Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP receive primary event crosses the MII interface. An interrupt is issued when the register is updated.

#### 44.7.90 PTP Event Frame Received Nanoseconds Register

**Name:** GMAC\_EFRN

**Address:** 0x400341EC

**Access:** Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Nanoseconds Register holds when the SFD of a PTP receive primary event crosses the MII interface. An interrupt is issued when the register is updated.

#### 44.7.91 PTP Peer Event Frame Transmitted Seconds Register

**Name:** GMAC\_PEFTS

**Address:** 0x400341F0

**Access:** Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP transmit peer event crosses the MII interface. An interrupt is issued when the register is updated.

#### 44.7.92 PTP Peer Event Frame Transmitted Nanoseconds Register

**Name:** GMAC\_PEFTN

**Address:** 0x400341F4

**Access:** Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Nanoseconds Register holds when the SFD of a PTP transmit peer event crosses the MII interface. An interrupt is issued when the register is updated.

#### 44.7.93 PTP Peer Event Frame Received Seconds Register

**Name:** GMAC\_PEFRS

**Address:** 0x400341F8

**Access:** Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP receive primary event crosses the MII interface. An interrupt is issued when the register is updated.

#### 44.7.94 PTP Peer Event Frame Received Nanoseconds Register

**Name:** GMAC\_PEFRN

**Address:** 0x400341FC

**Access:** Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Nanoseconds Register holds when the SFD of a PTP receive primary event crosses the MII interface. An interrupt is issued when the register is updated.

## 45. Digital-to-Analog Converter Controller (DACC)

### 45.1 Description

The Digital-to-Analog Converter Controller (DACC) offers up to 2 analog outputs, making it possible for the digital-to-analog conversion to drive up to 2 independent analog lines.

The DACC supports 12-bit resolution. Data to be converted are sent in a common register for all channels. External triggers or free running mode are configurable.

The DACC integrates a Sleep Mode and connects with a PDC channel. These features reduce both power consumption and processor intervention.

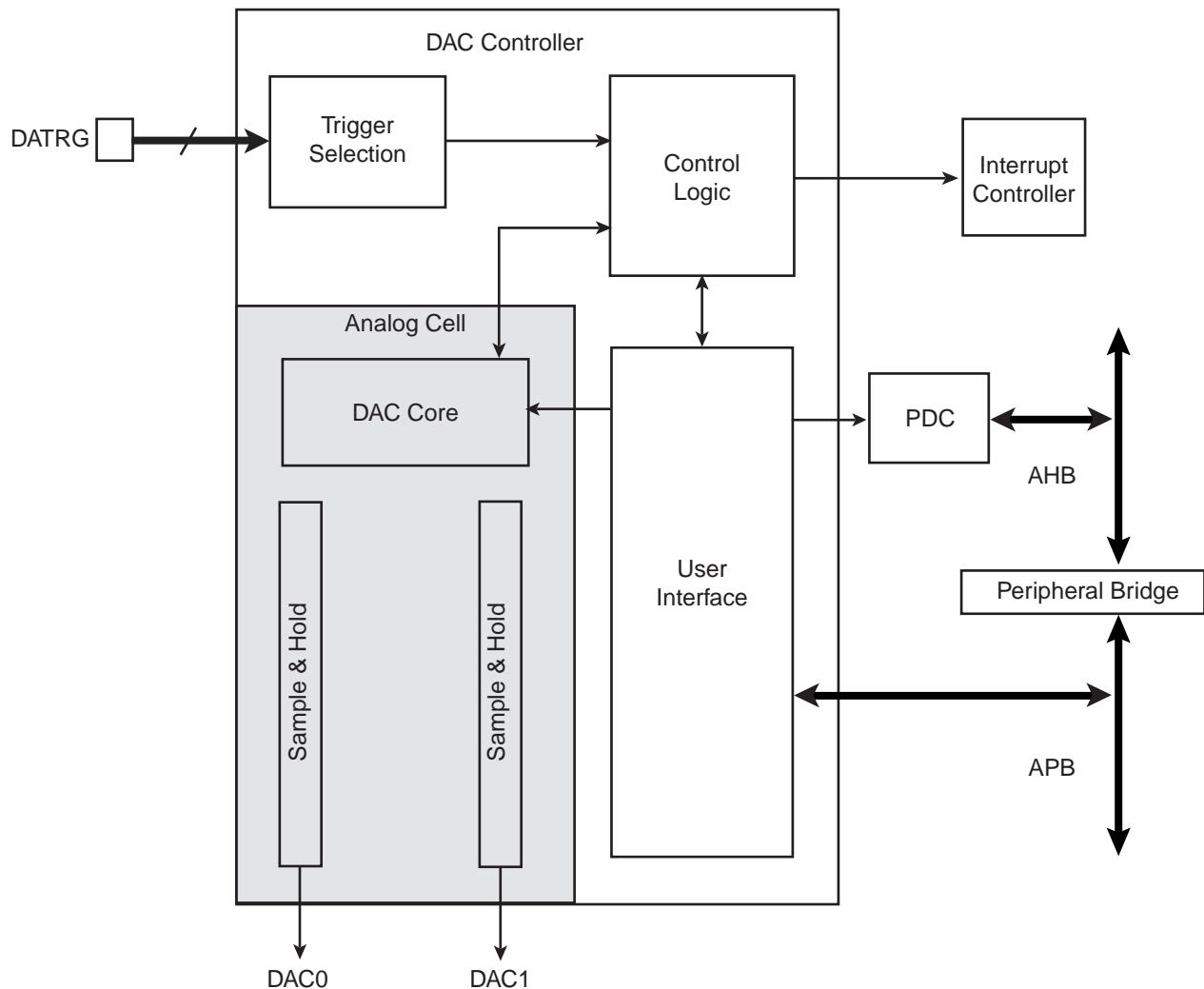
The user can configure DACC timings, such as Startup Time and Refresh Period.

### 45.2 Embedded Characteristics

- Up to Two Independent Analog Outputs
- 12-bit Resolution
- Individual Enable and Disable of Each Analog Channel
- Hardware Trigger
  - External Trigger Pins
- PDC Support
- Possibility of DACC Timings and Current Configuration
- Sleep Mode
  - Automatic Wake-up on Trigger and Back-to-Sleep Mode after Conversions of all Enabled Channels
- Internal FIFO

### 45.3 Block Diagram

Figure 45-1. Digital-to-Analog Converter Controller Block Diagram



## 45.4 Signal Description

Table 45-1. DACC Pin Description

Pin Name	Description
DAC0 - DAC1	Analog output channels
DATRG	External triggers

## 45.5 Product Dependencies

### 45.5.1 Power Management

The programmer must first enable the DAC Controller Clock in the Power Management Controller (PMC) before using the DACC.

The DACC becomes active as soon as a conversion is requested and at least one channel is enabled. The DACC is automatically deactivated when no channels are enabled.

For power saving options see [Section 45.6.6 "Sleep Mode"](#).

### 45.5.2 Interrupt Sources

The DACC interrupt line is connected on one of the internal sources of the interrupt controller. Using the DACC interrupt requires the interrupt controller to be programmed first.

Table 45-2. Peripheral IDs

Instance	ID
DACC	32

### 45.5.3 Conversion Performances

For performance and electrical characteristics of the DACC, see the product DC Characteristics section.

## 45.6 Functional Description

### 45.6.1 Digital-to-Analog Conversion

The DACC uses the master clock (MCK) divided by either two or four to perform conversions. This clock is named DACC Clock. If MCK clock frequency is above 100 MHz the CLKDIV bit must be written at 1 in DACC Mode register (DACC\_MR). Once a conversion starts the DACC takes 25 clock periods to provide the analog result on the selected analog output.

### 45.6.2 Conversion Results

When a conversion is completed, the resulting analog value is available at the selected DACC channel output and the EOC bit in the [DACC Interrupt Status Register](#), is set.

Reading the DACC\_ISR register clears the EOC bit.

### 45.6.3 Conversion Triggers

In free running mode, conversion starts as soon as at least one channel is enabled and data is written in the [DACC Conversion Data Register](#), then 25 DACC Clock periods later, the converted data is available at the corresponding analog output as stated above.

In external trigger mode, the conversion waits for a rising edge on the selected trigger to begin.

**Warning:** Disabling the external trigger mode automatically sets the DACC in free running mode.

### 45.6.4 Conversion FIFO

A 4 half-word FIFO is used to handle the data to be converted.

As long as the TXRDY flag in the [DACC Interrupt Status Register](#) is active the DAC Controller is ready to accept conversion requests by writing data into [DACC Conversion Data Register](#). Data which cannot be converted immediately are stored in the DACC FIFO.

When the FIFO is full or the DACC is not ready to accept conversion requests, the TXRDY flag is inactive.

The WORD field of the [DACC Mode Register](#) allows the user to switch between half-word and word transfer for writing into the FIFO.

In half-word transfer mode only the 16 LSB of DACC\_CDR data are taken into account, DACC\_CDR[15:0] is stored into the FIFO.

DACC\_CDR[11:0] field is used as data and the DACC\_CDR[15:12] bits are used for channel selection if the TAG field is set in DACC\_MR register.

In word transfer mode each time the DACC\_CDR register is written 2 data items are stored in the FIFO. The first data item sampled for conversion is DACC\_CDR[15:0] and the second DACC\_CDR[31:16].

Fields DACC\_CDR[15:12] and DACC\_CDR[31:28] are used for channel selection if the TAG field is set in DACC\_MR register.

**Warning:** Writing in the DACC\_CDR register while TXRDY flag is inactive will corrupt FIFO data.

### 45.6.5 Channel Selection

There are two means by which to select the channel to perform data conversion.

- By default, to select the channel where to convert the data, is to use the USER\_SEL field of the [DACC Mode Register](#). Data requests will merely be converted to the channel selected with the USER\_SEL field.
- A more flexible option to select the channel for the data to be converted to is to use the tag mode, setting the TAG field of the [DACC Mode Register](#) to 1. In this mode the 2 bits, DACC\_CDR[13:12] which are otherwise unused, are employed to select the channel in the same way as with the USER\_SEL field. Finally, if the WORD field is set, the 2 bits, DACC\_CDR[13:12] are used for channel selection of the first data and the 2 bits, DACC\_CDR[29:28] for channel selection of the second data.

#### 45.6.6 Sleep Mode

The DACC Sleep Mode maximizes power saving by automatically deactivating the DACC when it is not being used for conversions.

When a start conversion request occurs, the DACC is automatically activated. As the analog cell requires a start-up time, the logic waits during this time and starts the conversion on the selected channel. When all conversion requests are complete, the DACC is deactivated until the next request for conversion.

A fast wake-up mode is available in the [DACC Mode Register](#) as a compromise between power saving strategy and responsiveness. Setting the FASTW bit to 1 enables the fast wake-up mode. In fast wake-up mode the DACC is not fully deactivated while no conversion is requested, thereby providing less power saving but faster wake-up (4 times faster).

#### 45.6.7 DACC Timings

The DACC startup time must be defined by the user in the STARTUP field of the [DACC Mode Register](#).

This startup time differs depending of the use of the fast wake-up mode along with sleep mode, in this case the user must set the STARTUP time corresponding to the fast wake up and not the standard startup time.

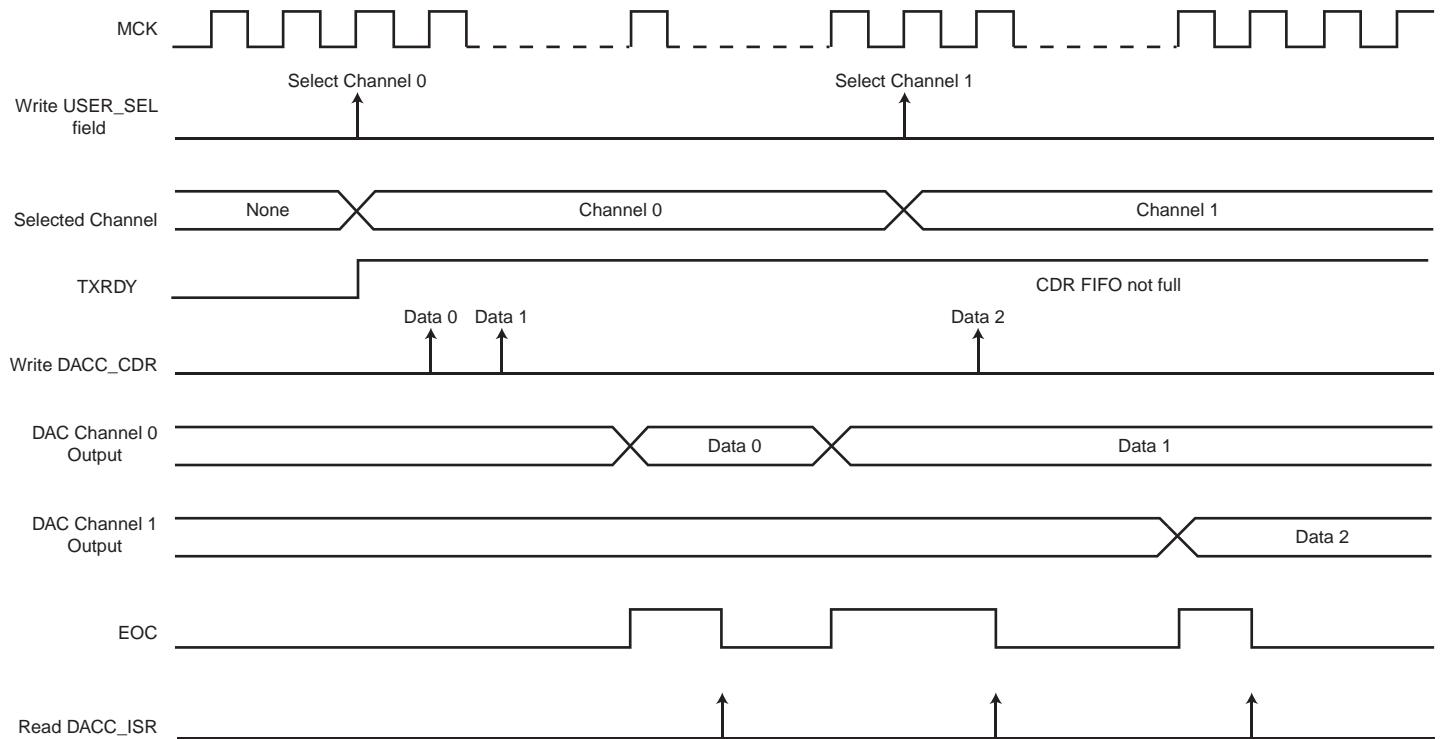
A max speed mode is available by setting the MAXS bit to 1 in the DACC\_MR register. Using this mode, the DAC Controller no longer waits to sample the end of cycle signal coming from the DACC block to start the next conversion and uses an internal counter instead. This mode gains 2 DACC Clock periods between each consecutive conversion.

**Warning:** Using this mode, the EOC interrupt of the DACC\_IER register should not be used as it is 2 DACC Clock periods late.

After 20 µs the analog voltage resulting from the converted data will start decreasing, therefore it is necessary to refresh the channel on a regular basis to prevent this voltage loss. This is the purpose of the REFRESH field in the DACC Mode Register where the user will define the period for the analog channels to be refreshed.

**Warning:** A REFRESH PERIOD field set to 0 will disable the refresh function of the DACC channels.

**Figure 45-2. Conversion Sequence**



#### 45.6.8 Write Protection Registers

In order to provide security to the DACC, a write protection system has been implemented.

The write protection mode prevents the writing of certain registers. When this mode is enabled and one of the protected registers is written, an error is generated in the [DACC Write Protect Status Register](#) and the register write request is canceled. When a write protection error occurs, the WPROTERR flag is set and the address of the corresponding canceled register write is available in the WPROTADDR field of the [DACC Write Protect Status Register](#).

Due to the nature of the write protection feature, enabling and disabling the write protection mode requires the use of a security code. Thus when enabling or disabling the write protection mode the WPKEY field of the [DACC Write Protect Mode Register](#) must be filled with the “DAC” ASCII code (corresponding to 0x444143) otherwise the register write is canceled.

The protected registers are:

- [DACC Mode Register](#)
- [DACC Channel Enable Register](#)
- [DACC Channel Disable Register](#)
- [DACC Analog Current Register](#)

## 45.7 Digital-to-Analog Converter Controller (DACC) User Interface

Table 45-3. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	DACC_CR	Write-only	–
0x04	Mode Register	DACC_MR	Read-write	0x00000000
0x08	Reserved	–	–	–
0x0C	Reserved	–	–	–
0x10	Channel Enable Register	DACC_CHER	Write-only	–
0x14	Channel Disable Register	DACC_CHDR	Write-only	–
0x18	Channel Status Register	DACC_CHSR	Read-only	0x00000000
0x1C	Reserved	–	–	–
0x20	Conversion Data Register	DACC_CDR	Write-only	0x00000000
0x24	Interrupt Enable Register	DACC_IER	Write-only	–
0x28	Interrupt Disable Register	DACC_IDR	Write-only	–
0x2C	Interrupt Mask Register	DACC_IMR	Read-only	0x00000000
0x30	Interrupt Status Register	DACC_ISR	Read-only	0x00000000
0x94	Analog Current Register	DACC_ACR	Read-write	0x00000000
0xE4	Write Protect Mode register	DACC_WPMR	Read-write	0x00000000
0xE8	Write Protect Status register	DACC_WPSR	Read-only	0x00000000
...	...	...	...	...
0xEC - 0xFC	Reserved	–	–	–

#### 45.7.1 DACC Control Register

**Name:** DACC\_CR

**Address:** 0x400B8000

**Access:** Write-only

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
—	—	—	—	—	—	—	SWRST

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the DACC simulating a hardware reset.

### 45.7.2 DACC Mode Register

**Name:** DACC\_MR

**Address:** 0x400B8004

**Access:** Read-write

31	30	29	28	27	26	25	24
—	—	STARTUP					
23	22	21	20	19	18	17	16
—	CLKDIV	MAXS	TAG	—	—	USER_SEL	
15	14	13	12	11	10	9	8
REFRESH							
7	6	5	4	3	2	1	0
—	FASTWKUP	SLEEP	WORD	TRGSEL			TRGEN

This register can only be written if the WPEN bit is cleared in [DACC Write Protect Mode Register](#).

- **TRGEN: Trigger Enable**

Value	Name	Description
0	DIS	External trigger mode disabled. DACC in free running mode.
1	EN	External trigger mode enabled.

- **TRGSEL: Trigger Selection**

TRGSEL			Selected TRGSEL
0	0	0	External trigger
0	0	1	TIO Output of the Timer Counter Channel 0
0	1	0	TIO Output of the Timer Counter Channel 1
0	1	1	TIO Output of the Timer Counter Channel 2
1	0	0	PWM Event Line 0
1	0	1	PWM Event Line 1
1	1	0	Reserved
1	1	1	Reserved

- **WORD: Word Transfer**

Value	Name	Description
0	HALF	Half-Word transfer
1	WORD	Word Transfer

- **SLEEP: Sleep Mode**

SLEEP	Selected Mode
0	Normal Mode
1	Sleep Mode

0 = Normal Mode: The DAC Core and reference voltage circuitry are kept ON between conversions.

After reset, the DAC is in normal mode but with the voltage reference and the DAC core off. For the first conversion, a startup time must be defined in the STARTUP field. Note that in this mode, STARTUP time is only required once, at start up.

1 = Sleep Mode: The DAC Core and reference voltage circuitry are OFF between conversions.

- **FASTWKUP: Fast Wake up Mode**

FASTWKUP	Selected Mode
0	Normal Sleep Mode
1	Fast Wake up Sleep Mode

0 = Normal Sleep Mode: The sleep mode is defined by the SLEEP bit.

1 = Fast Wake Up Sleep Mode: The voltage reference is ON between conversions and DAC Core is OFF.

- **REFRESH: Refresh Period**

Refresh Period =  $1024 * \text{REFRESH/DACC Clock}$

- **USER\_SEL: User Channel Selection**

Value	Name	Description
0	CHANNEL0	Channel 0
1	CHANNEL1	Channel 1

- **TAG: Tag Selection Mode**

Value	Name	Description
0	DIS	Tag selection mode disabled. Using USER_SEL to select the channel for the conversion.
1	EN	Tag selection mode enabled

- **MAXS: Max Speed Mode**

Value	Name	Description
0	NORMAL	Normal Mode
1	MAXIMUM	Max Speed Mode enabled

- **CLKDIV: Clock Divider**

Value	Name	Description
0	DIV_2	The DAC clock is MCK divided by 2
1	DIV_4	The DAC clock is MCK divided by 4 (to be used when MCK frequency is above 100 MHz)

- **STARTUP:** Startup Time Selection

<b>Value</b>	<b>Name</b>	<b>Description</b>
0	0	0 periods of DACClock
1	8	8 periods of DACClock
2	16	16 periods of DACClock
3	24	24 periods of DACClock
4	64	64 periods of DACClock
5	80	80 periods of DACClock
6	96	96 periods of DACClock
7	112	112 periods of DACClock
8	512	512 periods of DACClock
9	576	576 periods of DACClock
10	640	640 periods of DACClock
11	704	704 periods of DACClock
12	768	768 periods of DACClock
13	832	832 periods of DACClock
14	896	896 periods of DACClock
15	960	960 periods of DACClock
16	1024	1024 periods of DACClock
17	1088	1088 periods of DACClock
18	1152	1152 periods of DACClock
19	1216	1216 periods of DACClock
20	1280	1280 periods of DACClock
21	1344	1344 periods of DACClock
22	1408	1408 periods of DACClock
23	1472	1472 periods of DACClock
24	1536	1536 periods of DACClock
25	1600	1600 periods of DACClock
26	1664	1664 periods of DACClock
27	1728	1728 periods of DACClock
28	1792	1792 periods of DACClock
29	1856	1856 periods of DACClock
30	1920	1920 periods of DACClock
31	1984	1984 periods of DACClock

<b>Value</b>	<b>Name</b>	<b>Description</b>
32	2048	2048 periods of DACClock
33	2112	2112 periods of DACClock
34	2176	2176 periods of DACClock
35	2240	2240 periods of DACClock
36	2304	2304 periods of DACClock
37	2368	2368 periods of DACClock
38	2432	2432 periods of DACClock
39	2496	2496 periods of DACClock
40	2560	2560 periods of DACClock
41	2624	2624 periods of DACClock
42	2688	2688 periods of DACClock
43	2752	2752 periods of DACClock
44	2816	2816 periods of DACClock
45	2880	2880 periods of DACClock
46	2944	2944 periods of DACClock
47	3008	3008 periods of DACClock
48	3072	3072 periods of DACClock
49	3136	3136 periods of DACClock
50	3200	3200 periods of DACClock
51	3264	3264 periods of DACClock
52	3328	3328 periods of DACClock
53	3392	3392 periods of DACClock
54	3456	3456 periods of DACClock
55	3520	3520 periods of DACClock
56	3584	3584 periods of DACClock
57	3648	3648 periods of DACClock
58	3712	3712 periods of DACClock
59	3776	3776 periods of DACClock
60	3840	3840 periods of DACClock
61	3904	3904 periods of DACClock
62	3968	3968 periods of DACClock
63	4032	4032 periods of DACClock

Note: Please refer to the product DAC electrical characteristics section for Startup Time value.

#### 45.7.3 DACC Channel Enable Register

**Name:** DACC\_CHER

**Address:** 0x400B8010

**Access:** Write-only

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
—	—	—	—	—	—	CH1	CH0

This register can only be written if the WPEN bit is cleared in [DACC Write Protect Mode Register](#).

- **CHx: Channel x Enable**

0 = No effect.

1 = Enables the corresponding channel.

#### 45.7.4 DACC Channel Disable Register

**Name:** DACC\_CHDR

**Address:** 0x400B8014

**Access:** Write-only

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
—	—	—	—	—	—	CH1	CH0

This register can only be written if the WPEN bit is cleared in [DACC Write Protect Mode Register](#).

- **CHx: Channel x Disable**

0 = No effect.

1 = Disables the corresponding channel.

**Warning:** If the corresponding channel is disabled during a conversion or if it is disabled then re-enabled during a conversion, its associated analog value and its corresponding EOC flags in DACC\_ISR are unpredictable.

#### 45.7.5 DACC Channel Status Register

**Name:** DACC\_CHSR

**Address:** 0x400B8018

**Access:** Read-only

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
-	-	-	-	-	-	CH1	CH0

- **CHx: Channel x Status**

0 = The corresponding channel is disabled.

1 = The corresponding channel is enabled.

#### 45.7.6 DACC Conversion Data Register

**Name:** DACC\_CDR

**Address:** 0x400B8020

**Access:** Write-only

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

- **DATA: Data to Convert**

When the WORD bit in DACC\_MR register is cleared, only DATA[15:0] is used else DATA[31:0] is used to write 2 data to be converted.

#### 45.7.7 DACC Interrupt Enable Register

**Name:** DACC\_IER

**Address:** 0x400B8024

**Access:** Write-only

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
—	—	—	—	TXBUFE	ENDTX	EOC	TXRDY

- **TXRDY:** Transmit Ready Interrupt Enable
- **EOC:** End of Conversion Interrupt Enable
- **ENDTX:** End of Transmit Buffer Interrupt Enable
- **TXBUFE:** Transmit Buffer Empty Interrupt Enable

#### 45.7.8 DACC Interrupt Disable Register

**Name:** DACC\_IDR

**Address:** 0x400B8028

**Access:** Write-only

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
-	-	-	-	TXBUFE	ENDTX	EOC	TXRDY

- **TXRDY:** Transmit Ready Interrupt Disable.
- **EOC:** End of Conversion Interrupt Disable
- **ENDTX:** End of Transmit Buffer Interrupt Disable
- **TXBUFE:** Transmit Buffer Empty Interrupt Disable

#### 45.7.9 DACC Interrupt Mask Register

**Name:** DACC\_IMR

**Address:** 0x400B802C

**Access:** Read-only

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
—	—	—	—	TXBUFE	ENDTX	EOC	TXRDY

- **TXRDY:** Transmit Ready Interrupt Mask
- **EOC:** End of Conversion Interrupt Mask
- **ENDTX:** End of Transmit Buffer Interrupt Mask
- **TXBUFE:** Transmit Buffer Empty Interrupt Mask

#### 45.7.10 DACC Interrupt Status Register

**Name:** DACC\_ISR

**Address:** 0x400B8030

**Access:** Read-only

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
—	—	—	—	TXBUFE	ENDTX	EOC	TXRDY

- **TXRDY: Transmit Ready Interrupt Flag**

0 = DACC is not ready to accept new conversion requests.

1 = DACC is ready to accept new conversion requests.

- **EOC: End of Conversion Interrupt Flag**

0 = No conversion has been performed since the last DACC\_ISR read.

1 = At least one conversion has been performed since the last DACC\_ISR read.

- **ENDTX: End of DMA Interrupt Flag**

0 = The Transmit Counter Register has not reached 0 since the last write in DACC\_TCR or DACC\_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in DACC\_TCR or DACC\_TNCR

- **TXBUFE: Transmit Buffer Empty**

0 = The Transmit Counter Register has not reached 0 since the last write in DACC\_TCR or DACC\_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in DACC\_TCR or DACC\_TNCR.

#### 45.7.11 DACC Analog Current Register

**Name:** DACC\_ACR

**Address:** 0x400B8094

**Access:** Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	IBCTLDACCORE	
7	6	5	4	3	2	1	0
–	–	–	–	IBCTLCH1		IBCTLCH0	

This register can only be written if the WPEN bit is cleared in [DACC Write Protect Mode Register](#).

- **IBCTLCHx: Analog Output Current Control**

Allows to adapt the slew rate of the analog output. (See the product electrical characteristics for further details.)

- **IBCTLDACCORE: Bias Current Control for DAC Core**

Allows to adapt performance versus power consumption. (See the product electrical characteristics for further details.)

#### 45.7.12 DACC Write Protect Mode Register

**Name:** DACC\_WPMR

**Address:** 0x400B80E4

**Access:** Read-write

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

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x444143 ("DAC" in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x444143 ("DAC" in ASCII).

The protected registers are:

[DACC Mode Register](#)

[DACC Channel Enable Register](#)

[DACC Channel Disable Register](#)

[DACC Analog Current Register](#)

- **WPKEY: Write Protect KEY**

This security code is needed to set/reset the WPROT bit value (see [Section 45.6.8 "Write Protection Registers"](#) for details).

Must be filled with "DAC" ASCII code.

#### 45.7.13 DACC Write Protect Status Register

**Name:** DACC\_WPSR

**Address:** 0x400B80E8

**Access:** Read-only

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

- **WPROTADDR: Write protection error address**

Indicates the address of the register write request which generated the error.

- **WPROTERR: Write protection error**

Indicates a write protection error.

## 46. SAM4E Electrical Characteristics

### 46.1 Absolute Maximum Ratings

Table 46-1. Absolute Maximum Ratings\*

Operating Temperature (Industrial).....	-40°C to + 105°C
Storage Temperature.....	-60°C to + 150°C
Voltage on Input Pins with Respect to Ground.....	-0.3V to + 4.0V
Maximum Operating Voltage (VDDCORE).....	1.32V
Maximum Operating Voltage (VDDIO).....	4.0V
Total DC Output Current on all I/O lines	
144-ball LFBGA.....	150 mA
144-lead LQFP.....	150 mA
100-ball TFBGA.....	150 mA
100-lead LQFP.....	150 mA

\*NOTICE:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. **Exposure to absolute maximum rating conditions for extended periods may affect device reliability.**

## 46.2 DC Characteristics

The following characteristics are applicable to the operating temperature range:  $T_A = -40^\circ\text{C}$  to  $105^\circ\text{C}$ , unless otherwise specified.

**Table 46-2. DC Characteristics**

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_{DDCORE}$	DC Supply Core		1.08	1.20	1.32	V
$V_{VDDIO}$	DC Supply I/Os	(2) (3)	1.62	3.3	3.6	V
$V_{VDDPLL}$	PLL A and Main Oscillator Supply		1.08	—	1.32	V
$V_{IL}$	Input Low-level Voltage	PA0-PA31, PB0-PB14, PC0-PC31, PD0-PD31, PE0-PE5 NRST	-0.3	—	[0.8V:0.3 x $V_{VDDIO}$ ]	V
$V_{IH}$	Input High-level Voltage	PA0-PA31, PB0-PB14, PC0-PC31, PD0-PD31, PE0-PE5 NRST	MIN[2.0 V:0.7 x $V_{VDDIO}$ ]	—	$V_{VDDIO} +0.3V$	V
$V_{OH}$	Output High-level Voltage	PA0-PA31, PB0-PB9, PB12-PB14, PC0-PC31, PD0-PD31, PE0-PE5 $I_{OL} = 4.0\text{mA}$	$V_{VDDIO} - 0.4V$	—	—	V
		VDDIO [3.0V : 3.60V] PB10-PB11	$V_{VDDIO} - 0.15V$	—	—	
$V_{OL}$	Output Low-level Voltage	PA0-PA31, PB0-PB9, PB12-PB14, PC0-PC31, PD0-PD31, PE0-PE5 $I_{OH} = 4.0\text{mA}$	—	—	0.4	V
		VDDIO [3.0V : 3.60V] PB10-PB11	—	—	0.15	
$V_{Hys}$	Hysteresis Voltage	PA0-PA31, PB0-PB9, PB12-PB14, PC0-PC31 (Hysteresis mode enabled)	150	—	—	mV
$I_o$	$I_{OH}$ (or $I_{SOURCE}$ )	VDDIO [1.65V : 3.60V]; $V_{OH} = V_{VDDIO} - 0.4$ - PA14 (SPCK), PA29(MCCK) pins - PA[5:8], PA[12-13], PA[26-28], PA[30-31], PB[8:9], PB[14], PD[0:1], PD[3:17] pins - PA [0-3] - Other pins <sup>(1)</sup> VDDIO [3.0V : 3.60V] - PB[10-11]	—	—	-4 -4 -2 -2 -30	mA
		VDDIO [1.65V : 3.60V]; $V_{OH} = V_{VDDIO} - 0.4$ - NRST	—	—	-2	

Table 46-2. DC Characteristics (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Units	
I <sub>O</sub>	I <sub>OL</sub> (or I <sub>SINK</sub> )	VDDIO [1.65V : 3.60V] ; V <sub>OL</sub> = 0.4V - PA14 (SPCK), PA29(MCCK) pins - PA[5:8], PA[12-13], PA[26-28], PA[30-31], PB[8:9], PB[14], PD[0:1],PD[3:17] pins - PA [0-3] - Other pins <sup>(1)</sup> VDDIO [3.0V : 3.60V] - PB[10-11]	—	—	4 4 2 2	30	mA
		VDDIO [1.65V : 3.60V] ; V <sub>OL</sub> = 0.4V - NRST	—	—	2		
I <sub>IL</sub>	Input Low	Pull_up OFF	-1	—	1	uA	
		Pull_up ON	10	—	50		
I <sub>IH</sub>	Input High	Pull-down OFF	-1	—	1	uA	
		Pull-down ON	10	—	50		
R <sub>PULLUP</sub>	Pull-up Resistor	PA0-PA31, PB0-PB14, PC0-PC31, PD0-PD31, PE0-PE5 NRST	70	100	130	kΩ	
R <sub>PULLDOWN</sub>	Pull-down Resistor	PA0-PA31, PB0-PB14, PC0-PC3,PD0-PD31, PE0-PE51 NRST	70	100	130	kΩ	
R <sub>ODT</sub>	On-die Series Termination Resistor	PA4-PA31, PB0-PB9, PB12-PB14,PC0- PC31,PD0-PD31, PE0-PE5 PA0-PA3	—	36 18	—	Ω	

Notes: 1. PA[4], PA[9-11], PA[15-25], PB[0-7], PB[12-13], PC[0-31], PD[2], PD[18-31], PE[0-5].  
 2. At power-up, VDDIO needs to reach 0.6V before VDDIN reaches 1.0V.  
 3. VDDIO voltage needs to be equal or below to (VDDIN voltage +0.5V).

**Table 46-3. 1.2V Voltage Regulator Characteristics**

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_{VDDIN}$	DC Input Voltage Range	(4) (5)	1.6	3.3	3.6	V
$V_{VDDOUT}$	DC Output Voltage	Normal Mode Standby Mode	— —	1.2 0	—	V
$V_{ACCURACY}$	Output Voltage Accuracy	$I_{Load} = 0.8 \text{ mA to } 80 \text{ mA (after trimming)}$	-4	—	4	%
$I_{LOAD}$	Maximum DC Output Current	$V_{VDDIN} > 1.8\text{V}$ $V_{VDDIN} \leq 1.8\text{V}$	— —	— —	120 70	mA
$I_{LOAD-START}$	Maximum Peak Current during startup	(3)	—	—	400	mA
$D_{DROPOUT}$	Dropout Voltage	$V_{VDDIN} = 1.6\text{V}, I_{Load} = 70\text{mA}$	—	400	—	mV
$V_{LINE}$	Line Regulation	$V_{VDDIN}$ from 2.7V to 3.6V; $I_{Load}$ MAX	—	10	30	mV
$V_{LINE-TR}$	Transient Line regulation	$V_{VDDIN}$ from 2.7V to 3.6V; $tr = tf = 5\mu\text{s}$ ; $I_{Load}$ Max $CD_{OUT} = 4.7\mu\text{F}$	—	50	150	mV
$V_{LOAD}$	Load Regulation	$V_{VDDIN} \geq 1.8\text{V};$ $I_{Load} = 10\% \text{ to } 90\% \text{ MAX}$	—	25	60	mV
$V_{LOAD-TR}$	Transient Load Regulation	$V_{VDDIN} \geq 1.8\text{V};$ $I_{Load} = 10\% \text{ to } 90\% \text{ MAX}$ $tr = tf = 5 \mu\text{s}$ $CD_{OUT} = 4.7 \mu\text{F}$	—	45	210	mV
$I_Q$	Quiescent Current	Normal Mode;	—	5.5	—	μA
		@ $I_{Load} = 0 \text{ mA}$ @ $I_{Load} = 120 \text{ mA}$	—	350	—	μA
$CD_{IN}$	Input Decoupling Capacitor	Cf. External Capacitor Requirements (1)	—	4.7	—	μF
	Output Decoupling Capacitor	Cf. External Capacitor Requirements (2)	1.85	2.2	5.9	μF
$T_{ON}$	Turn on Time	$CD_{OUT} = 2.2\mu\text{F}, V_{VDDOUT}$ reaches 1.2V (+/- 3%)	—	300	—	μs
$T_{OFF}$	Turn off Time	$CD_{OUT} = 2.2\mu\text{F}; V_{VDDIN} \geq 1.8\text{V}$	—	—	9.5	ms

Notes: 1. A 4.7  $\mu\text{F}$  or higher ceramic capacitor must be connected between  $VDDIN$  and the closest GND pin of the device.

This large decoupling capacitor is mandatory to reduce startup current, improving transient response and noise rejection.

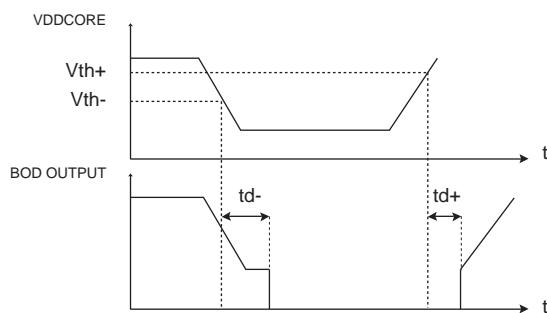
2. To ensure stability, an external 2.2  $\mu\text{F}$  output capacitor,  $CD_{OUT}$  must be connected between the  $VDDOUT$  and the closest GND pin of the device. The ESR (Equivalent Series Resistance) of the capacitor must be in the range 0.1Ω to 10Ω. Solid tantalum and multilayer ceramic capacitors are all suitable as output capacitor. A 100 nF bypass capacitor between  $VDDOUT$  and the closest GND pin of the device helps decreasing output noise and improves the load transient response.
3. Defined as the current needed to charge external bypass/decoupling capacitor network.
4. At power-up  $VDDIO$  needs to reach 0.6V before  $VDDIN$  reaches 1.0V
5.  $VDDIO$  voltage needs to be equal or below to ( $VDDIN$  voltage +0.5V)

**Table 46-4. Core Power Supply Brownout Detector Characteristics**

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_{TH-}$	Supply Falling Threshold <sup>(1)</sup>		0.98	1.0	1.04	V
$V_{HYST}$	Hysteresis		—	—	110	mV
$V_{TH+}$	Supply Rising Threshold		0.8	1.0	1.08	V
$I_{DDON}$ $I_{DDOFF}$	Current Consumption on VDDCORE	Brownout Detector enabled	—	—	24	$\mu$ A
		Brownout Detector disabled	—	—	2	uA
$I_{DD33ON}$ $I_{DD33OFF}$	Current Consumption on VDDIO	Brownout Detector enabled	—	—	24	$\mu$ A
		Brownout Detector disabled	—	—	2	uA
$T_d-$	$V_{TH-}$ detection propagation time	$VDDCORE = V_{TH+}$ to $(V_{TH-} - 100mV)$	—	200	300	ns
$T_{START}$	Startup Time	From disabled state to enabled state	—	—	300	$\mu$ s

Note: 1. The product is guaranteed to be functional at  $V_{TH-}$ .

**Figure 46-1. Core Brownout Output Waveform**



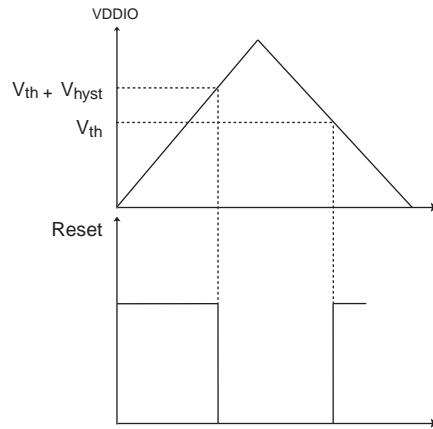
**Table 46-5. VDDIO Supply Monitor**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Typ</b>	<b>Max</b>	<b>Units</b>
$V_{TH}$	Supply Monitor Threshold	16 selectable steps	1.6	—	3.4	V
$T_{ACCURACY}$	Threshold Level Accuracy	[-40/+105°C]	-2.5	—	+2.5	%
$V_{HYST}$	Hysteresis		—	20	30	mV
$I_{DDON}$	Current Consumption	enabled		23	40	$\mu A$
$I_{DDOFF}$		disabled	—	0.02	2	
$T_{START}$	Startup Time	From disabled state to enabled state	—	—	300	$\mu s$

**Table 46-6. Threshold Selection**

<b>Digital Code</b>	<b>Threshold min (V)</b>	<b>Threshold typ (V)</b>	<b>Threshold max (V)</b>
0000	1.56	1.6	1.64
0001	1.68	1.72	1.76
0010	1.79	1.84	1.89
0011	1.91	1.96	2.01
0100	2.03	2.08	2.13
0101	2.15	2.2	2.23
0110	2.26	2.32	2.38
0111	2.38	2.44	2.50
1000	2.50	2.56	2.62
1001	2.61	2.68	2.75
1010	2.73	2.8	2.87
1011	2.85	2.92	2.99
1100	2.96	3.04	3.12
1101	3.08	3.16	3.24
1110	3.20	3.28	3.36
1111	3.32	3.4	3.49

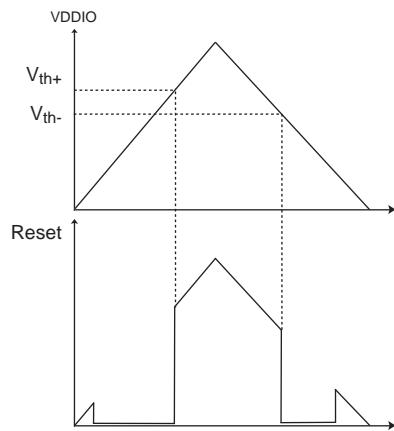
**Figure 46-2. VDDIO Supply Monitor**



**Table 46-7. Zero-Power-On Reset Characteristics**

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_{th+}$	Threshold voltage rising	At Startup	1.45	1.53	1.59	V
$V_{th-}$	Threshold voltage falling		1.35	1.45	1.55	V
Tres	Reset Time-out Period		100	340	580	μs

**Figure 46-3. Zero-Power-On Reset Characteristics**



**Table 46-8. DC Flash Characteristics**

Symbol	Parameter	Conditions	Typ	Max	Units
$I_{cc}$	Active current	Random 128-bit Read: Maximum Read Frequency onto VDDCORE = 1.2V @ 25°C	16	25	mA
		Random 64-bit Read: Maximum Read Frequency onto VDDCORE = 1.2V @ 25°C	10	18	mA
		Program onto VDDCORE = 1.2V @ 25°C	3	5	mA

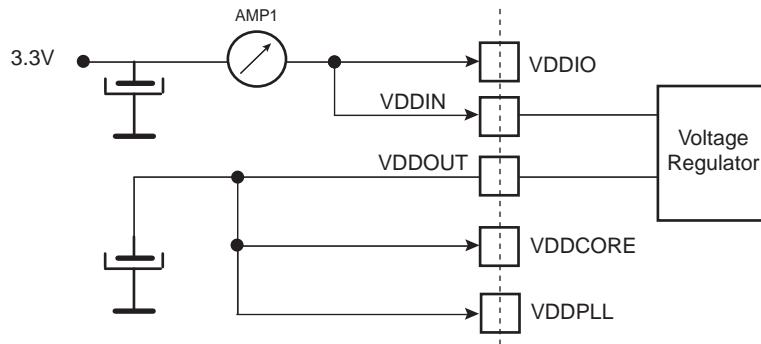
## 46.3 Power Consumption

- Power consumption of the device according to the different Low Power Mode Capabilities (Backup, Wait, Sleep and Active Mode).
- Power consumption on power supply in different modes: Backup, Wait, Sleep and Active.
- Power consumption by peripheral: calculated as the difference in current measurement after having enabled then disabled the corresponding clock.
- All power consumption values are based on characterization. Note that these values are not covered by test limits in production.

### 46.3.1 Backup Mode Current Consumption

The backup mode configuration and measurements are defined as follow.

**Figure 46-4. Measurement Setup**



#### 46.3.1.1 Configuration A: Embedded Slow Clock RC Oscillator Enabled

- Supply Monitor on VDDIO is disabled
- RTT used
- BOD disabled
- One WKUPx enabled
- Current measurement on AMP1 (see [Figure 46-4](#))

#### 46.3.1.2 Configuration B: 32768 kHz Crystal Oscillator Enabled

- Supply Monitor on VDDIO is disabled
- RTT used
- BOD disabled
- One WKUPx enabled
- Current measurement on AMP1 (see [Figure 46-4](#))

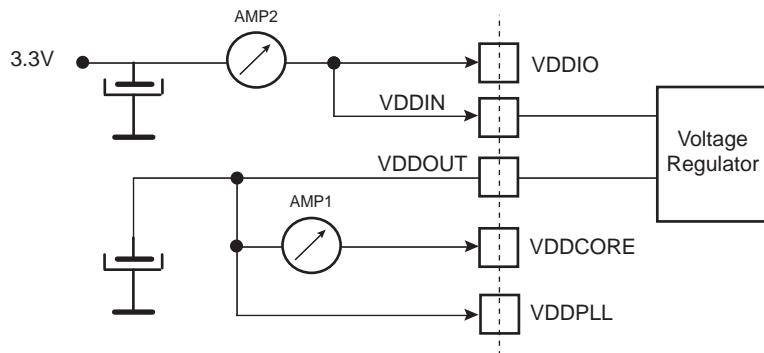
**Table 46-9. Power Consumption for Backup Mode Configuration A and B**

BACKUP Total Consumption	Typical value @25 degrees		Maximum Value @85 degrees	Maximum Value @105 degrees	Unit
Conditions	(AMP1) Configuration A	(AMP1) Configuration B	(AMP1) Configuration A	(AMP1) Configuration A	Unit
VDDIO = 3.6V	2.0	1.9	7.2	15.4	
VDDIO = 3.3V	1.7	1.6	6.9	12.0	
VDDIO = 3.0V	1.5	1.5	6.2	11.2	µA
VDDIO = 2.5V	1.3	1.2	5.5	9.8	
VDDIO = 1.8V	1	0.9	4.6	8.3	

#### 46.3.2 Sleep and Wait Mode Current Consumption

The Wait Mode and Sleep Mode configuration and measurements are defined below.

Figure 46-5. Measurement Setup for Sleep Mode

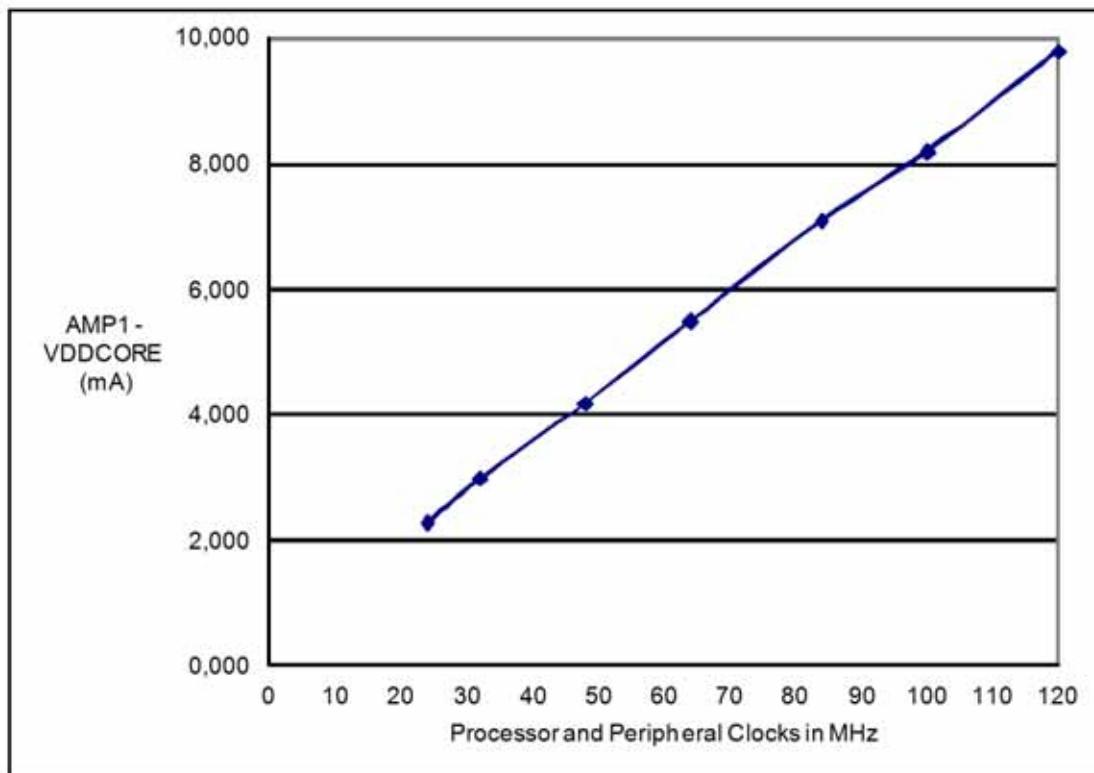


##### 46.3.2.1 Sleep Mode

- Core Clock OFF
- VDDIO = VDDIN = 3.3V
- Master Clock (MCK) running at various frequencies with PLLA or the fast RC oscillator
- Fast start-up through WKUP0-15 pins
- Current measurement as shown in figure [Figure 46-5](#)
- All peripheral clocks deactivated
- Temperature = 25°C

[Table 46-10](#) gives current consumption in typical conditions.

Figure 46-6. Current Consumption in Sleep Mode (AMP1) versus Master Clock Ranges (Condition from [Table 46-10](#))



**Table 46-10. Sleep Mode Current Consumption versus Master Clock (MCK) Variation with PLLA**

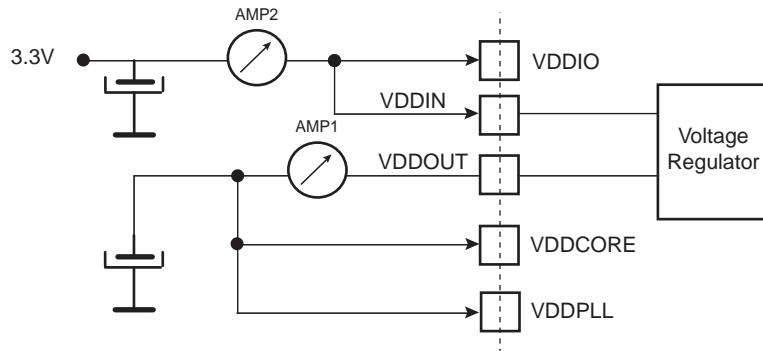
Core Clock/MCK (MHz)	VDDCORE Consumption (AMP1)	Total Consumption (AMP2)	Unit
120	9.8	11.4	mA
100	8.2	9.5	mA
84	7.1	9.4	mA
64	5.5	7.2	mA
48	4.2	5.5	mA
32	3.0	4.7	mA
24	2.3	3.5	mA

**Table 46-11. Sleep Mode Current Consumption versus Master Clock (MCK) Variation with FAST RC**

Core Clock/MCK (MHz)	VDDCORE Consumption (AMP1)	Total Consumption (AMP2)	Unit
12	1.11	1.14	mA
8	0.77	0.8	mA
4	0.45	0.48	mA
2	0.3	0.33	mA
1	0.22	0.25	mA
0.5	0.18	0.21	mA
0.25	0.16	0.19	mA

#### 46.3.2.2 Wait Mode

**Figure 46-7. Measurement Setup for Wait Mode**



- VDDIO = VDDIN = 3.6V
- Core Clock and Master Clock stopped
- Current measurement as shown in the above figure
- All peripheral clocks deactivated
- BOD disabled
- RTT enabled

Table 46-12 gives current consumption in typical conditions.

Note: 1. Value from characterization, not tested in production.

**Table 46-12. Typical Current Consumption in Wait Mode**

Wait Mode Consumption	Typical Value @25°		Maximum Value @85°	Maximum Value @105°	Unit
Conditions	VDDOUT Consumption (AMP1)	Total Consumption (AMP2)	Total Consumption (AMP2)	Total Consumption (AMP2)	Unit
See Figure 46-7 on page 1318 There is no activity on the I/Os of the device. With the Flash in Standby Mode	43	56	550	1200	µA
See Figure 46-7 on page 1318 There is no activity on the I/Os of the device. With the Flash in Deep Power Down Mode	39	47	496	980	

#### 46.3.3 Active Mode Power Consumption

The Active Mode configuration and measurements are defined as follows:

- VDDIO = VDDIN = 3.3V
- VDDCORE = 1.2V (Internal Voltage regulator used)
- TA = 25°C
- Application running from Flash Memory with 128-bit access mode
- All peripheral clocks are deactivated.
- Master Clock (MCK) running at various frequencies with PLLA or the fast RC oscillator.
- Current measurement on AMP1 (VDDCORE) and total current on AMP2

**Figure 46-8. Active Mode Measurement Setup**

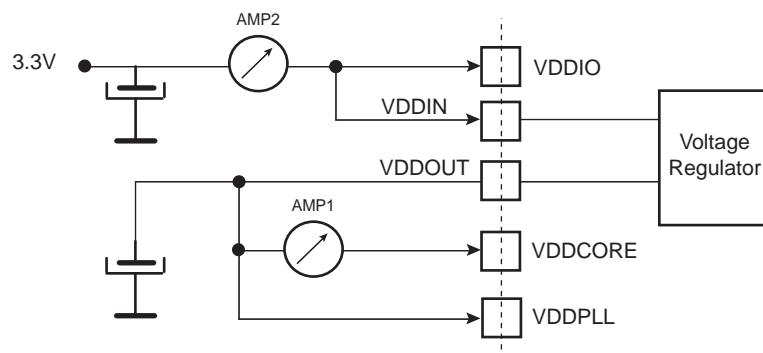


Table 46-13 on page 1320 and Figure 46-14 on page 1321 give Active Mode Current Consumption in typical conditions.

- VDDCORE at 1.2V
- Temperature = 25°C

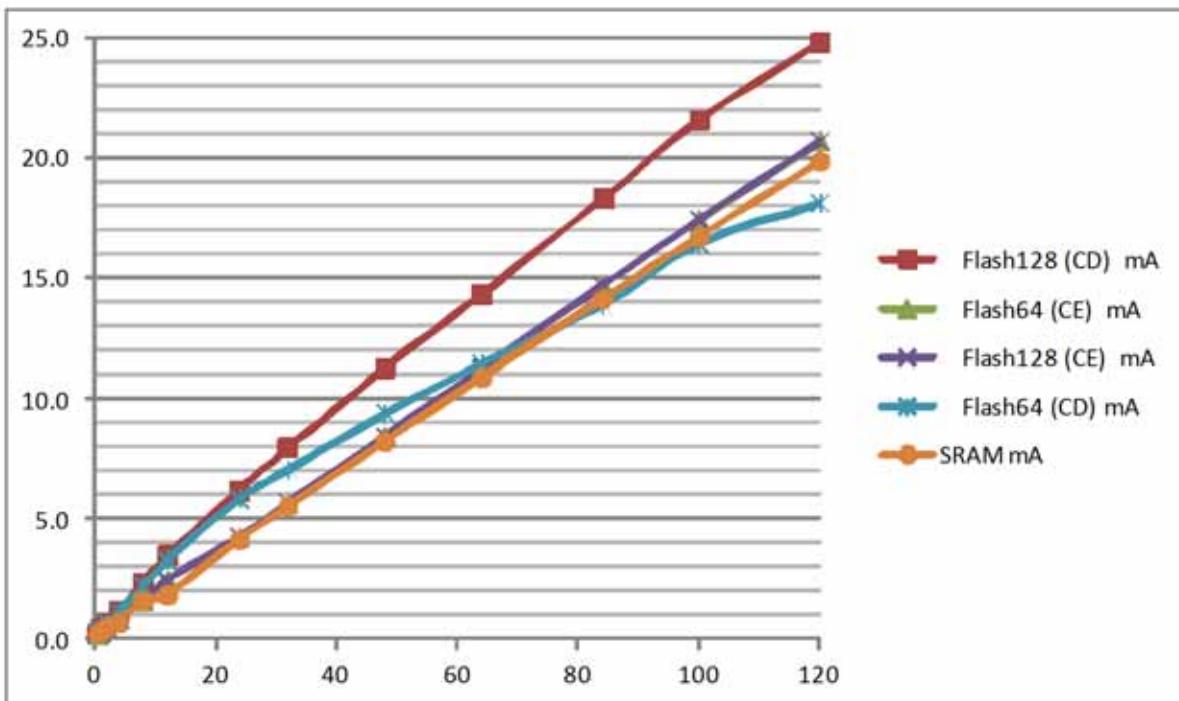
#### 46.3.3.1 SAM4E Active Power Consumption

Table 46-13. Active Power Consumption with VDDCORE @ 1.2V Running from Embedded Memory (IDDCORE- AMP1)

Core Clock (MHz)	CoreMark					Unit	
	Cache Enable (CE)		Cache Disable (CD)		SRAM		
	128-bit Flash access <sup>(1)</sup>	64-bit Flash access <sup>(1)</sup>	128-bit Flash access <sup>(1)</sup>	64-bit Flash access <sup>(1)</sup>			
120	21.1	21.0	25.5	19.0	17.9	mA	
100	18.1	18.1	22.5	17.2	15.0		
84	15.5	15.5	20.0	16.1	12.86		
64	11.9	11.9	16.4	13.6	9.9		
48	9.0	9.0	12.7	11.7	7.5		
32	6.2	6.2	9.1	8.9	5.2		
24	4.6	4.6	7	6.8	3.9		
12	2.5	2.4	4.1	3.8	2.2		
8	1.9	1.8	2.9	2.8	1.6		
4	1.2	1.1	1.7	1.7	1.0		
2	0.81	0.79	1	1	0.75		
1	0.46	0.46	0.6	0.6	0.44		
0.5	0.38	0.38	0.47	0.44	0.36		

Note: 1. Flash Wait State (FWS) in EEFC\_FMR is adjusted versus Core Frequency.

Figure 46-9. Active Power Consumption with VDDCORE @ 1.2V



- VDDCORE at 1.2V
- Temperature = 25°C

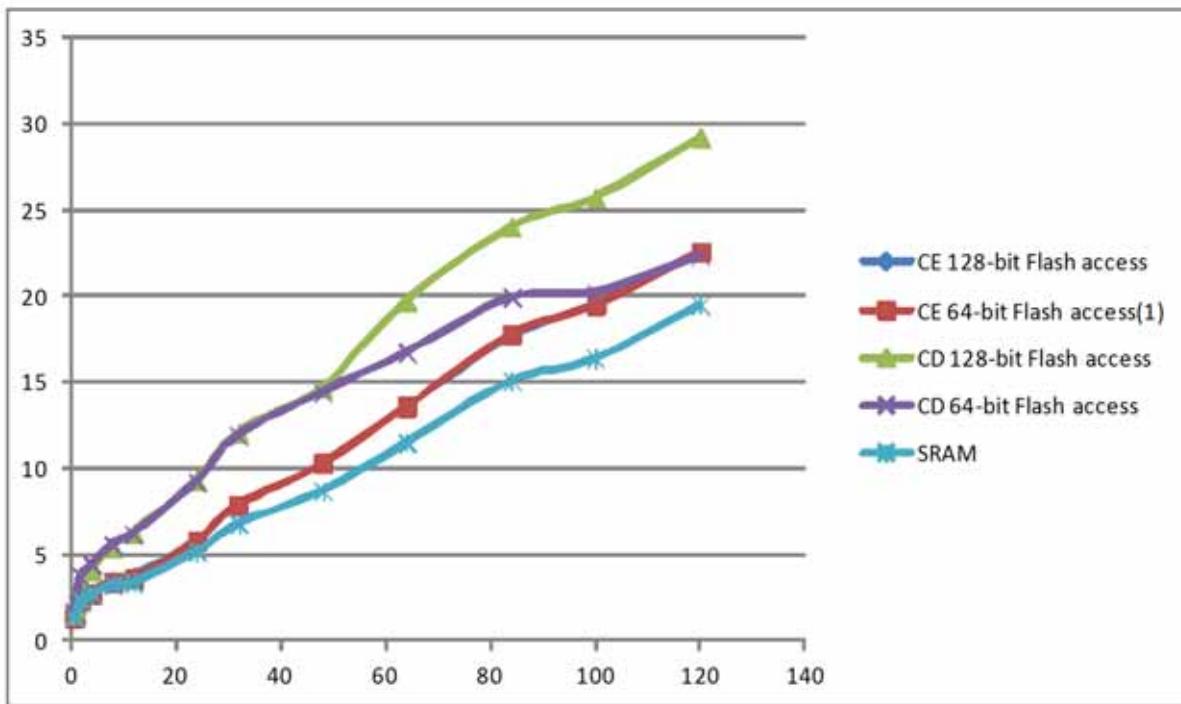
#### 46.3.3.2 SAM4E Active Total Power Consumption

Table 46-14. Active Total Power Consumption with VDDCORE @ 1.2V running from Embedded Memory (IDDIO + IDDIN - AMP2)

Core Clock (MHz)	CoreMark					Unit	
	Cache Enable (CE)		Cache Disable (CD)		SRAM		
	128-bit Flash Access <sup>(1)</sup>	64-bit Flash Access <sup>(1)</sup>	128-bit Flash Access <sup>(1)</sup>	64-bit Flash Access <sup>(1)</sup>			
120	22.6	22.6	29.2	22.3	19.5	mA	
100	19.5	19.5	25.7	20.2	16.4		
84	17.7	17.8	24.0	19.9	15.1		
64	13.6	13.6	19.7	16.7	11.5		
48	10.3	10.3	14.7	14.4	8.7		
32	7.9	7.9	12.1	11.9	6.8		
24	5.8	5.8	9.3	9.2	5.2		
12	3.8	3.6	6.3	6.2	3.4		
8	3.5	3.4	5.5	5.6	3.3		
4	2.8	2.7	4.1	4.4	2.7		
2	2.5	2.4	3.6	3.7	2.4		
1	1.5	1.5	1.9	2.1	1.5		
0.5	1.4	1.4	1.6	1.7	1.4		

Note: 1. Flash Wait State (FWS) in EEFC\_FMR adjusted versus Core Frequency

Figure 46-10. Active Total Power Consumption with VDDCORE @ 1.2V



#### 46.3.4 Peripheral Power Consumption in Active Mode

Table 46-15. Power Consumption on  $V_{DDCORE}$ <sup>(1)</sup>

Peripheral	Consumption (Typ)	Unit
PIO Controller A (PIOA)	5.23	
PIO Controller B (PIOB)	1.44	
PIO Controller C (PIOC)	4.02	
PIO Controller D (PIOD)	3.17	
PIO Controller E (PIOE)	0.86	
UART	4.50	
USART	6.5	
PWM	11.00	
TWI	4.70	
SPI	4.42	
Timer Counter (TCx)	3.7	µA/MHz
AFEC	5.15	
DACC	3.0	
ACC	0.28	
HSMCI	6.43	
CAN	6.5	
SMC	2.77	
UDP	5.11	
EMAC	44.2	
AES	2.39	
DMAC	7.21	

Note: 1. Note: VDDIO = 3.3V,  $V_{DDCORE}$  = 1.08V,  $T_A$  = 25°C

## 46.4 Oscillator Characteristics

### 46.4.1 32 kHz RC Oscillator Characteristics

Table 46-16. 32 kHz RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
	RC Oscillator Frequency		20	32	44	kHz
	Frequency Supply Dependency		-3	—	3	%/V
	Frequency Temperature Dependency	Over temperature range (-40°C/+105°C) versus 25°C	-7	—	7	%
Duty	Duty Cycle		45	50	55	%
T <sub>ON</sub>	Startup Time		—	—	100	μs
I <sub>DDON</sub>	Current Consumption	After Startup Time Temp. Range = -40°C to +105°C Typical Consumption at 2.2V supply and Temp = 25°C	—	540	860	nA

### 46.4.2 4/8/12 MHz RC Oscillators Characteristics

Table 46-17. 4/8/12 MHz RC Oscillators Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
F <sub>Range</sub>	RC Oscillator Frequency Range	(1)	4	—	12	MHz
ACC <sub>4</sub>	4 MHz Total Accuracy	-40°C < Temp < +105°C 4 MHz output selected (1)(2)	—	—	±30	%
ACC <sub>8</sub>	8 MHz Total Accuracy	-40°C < Temp < +105°C 8 MHz output selected (1)(2)	—	—	±30	%
		-40°C < Temp < +105°C 8 MHz output selected (1)(3)	—	—	±5	
ACC <sub>12</sub>	12 MHz Total Accuracy	-40°C < Temp < +105°C 12 MHz output selected (1)(2)	—	—	±30	%
		-40°C < Temp < +105°C 12 MHz output selected (1)(3)	—	—	±5	
	Frequency deviation versus trimming code	8 MHz 12 MHz	—	47 64	—	kHz/trimming code
Duty	Duty Cycle		45	50	55	%
T <sub>ON</sub>	Startup Time		—	—	10	μs
I <sub>DDON</sub>	Active Current Consumption <sup>(2)</sup>	4 MHz	50	75		
		8 MHz	65	95		
		12 MHz	82	118		μA

Notes: 1. Frequency range can be configured in the Supply Controller Registers

- 2. Not trimmed from factory
- 3. After Trimming from factory

The 4/8/12 MHz Fast RC oscillator is calibrated in production. This calibration can be read through the Get CALIB Bit command (see the EEFC section) and the frequency can be trimmed by software through the PMC.

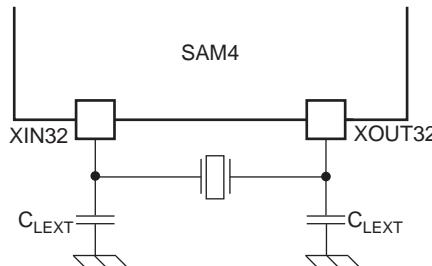
#### 46.4.3 32.768 kHz Crystal Oscillator Characteristics

Table 46-18. 32.768 kHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
$F_{req}$	Operating Frequency	Normal mode with crystal		—	—	32.768	kHz
	Supply Ripple Voltage (on VDDIO)	Rms value, 10 KHz to 10 MHz		—	—	30	mV
	Duty Cycle			40	50	60	%
	Startup Time	$R_s < 50 \text{ k}\Omega$ <sup>(1)</sup>	$C_{\text{crystal}} = 12.5 \text{ pF}$ $C_{\text{crystal}} = 6 \text{ pF}$	—	—	900 300 1200 500	ms
		$R_s < 100 \text{ k}\Omega$ <sup>(1)</sup>	$C_{\text{crystal}} = 12.5 \text{ pF}$ $C_{\text{crystal}} = 6 \text{ pF}$	—	—	1200 500	
Iddon	Current consumption	$R_s < 50 \text{ k}\Omega$ <sup>(1)</sup>	$C_{\text{crystal}} = 12.5 \text{ pF}$ $C_{\text{crystal}} = 6 \text{ pF}$	—	550 380 820 530	1150 980 1600 1350	nA
$P_{ON}$	Drive level			—	—	0.1	$\mu\text{W}$
$R_f$	Internal resistor	Between XIN32 and XOUT32		—	10		$\text{M}\Omega$
$C_{\text{LEXT}}$	Maximum external capacitor on XIN32 and XOUT32			—	—	20	pF
$C_{\text{para}}$	Internal Parasitic Capacitance			0.6	0.7	0.8	pF

Note: 1.  $R_S$  is the series resistor.

Figure 46-11. 32.768 kHz Crystal Oscillator Schematics



$$C_{\text{LEXT}} = 2x(C_{\text{CRYSTAL}} - C_{\text{para}} - C_{\text{PCB}})$$

where:

$C_{\text{PCB}}$  is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM4 pin.

#### 46.4.4 32.768 kHz Crystal Characteristics

Table 46-19. Crystal Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor ( $R_S$ )	Crystal @ 32.768 kHz	—	50	100	kΩ
$C_M$	Motional capacitance	Crystal @ 32.768 kHz	0.6	—	3	fF
$C_{SHUNT}$	Shunt capacitance	Crystal @ 32.768 kHz	0.6	—	2	pF

#### 46.4.5 3 to 20 MHz Crystal Oscillator Characteristics

Table 46-20. 3 to 20 MHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$F_{req}$	Operating Frequency	Normal mode with crystal	3	16	20	MHz
	Supply Ripple Voltage (on VDDPLL)	RMS value, 10 kHz to 10 MHz	—	—	30	mV
	Duty Cycle		40	50	60	%
$T_{ON}$	Startup Time	3 MHz, $C_{SHUNT} = 3 \text{ pF}$ 8 MHz, $C_{SHUNT} = 7 \text{ pF}$ 16 MHz, $C_{SHUNT} = 7 \text{ pF}$ with $C_m = 8 \text{ fF}$ 16 MHz, $C_{SHUNT} = 7 \text{ pF}$ with $C_m = 1.6 \text{ fF}$ 20 MHz, $C_{SHUNT} = 7 \text{ pF}$	—	—	14.5 4 1.4 2.5 1	ms
$I_{DD\_ON}$	Current consumption (on VDDIO)	3 MHz <sup>(2)</sup> 8 MHz <sup>(3)</sup> 16 MHz <sup>(4)</sup> 20 MHz <sup>(5)</sup>	—	230 300 390 450	350 400 470 560	μA
$P_{ON}$	Drive level	3 MHz 8 MHz 16 MHz, 20 MHz	—	—	15 30 50	μW
$R_f$	Internal resistor	Between XIN and XOUT	—	0.5	—	MΩ
$C_{LEXT}$	Maximum external capacitor on XIN and XOUT		12.5	—	17.5	pF
$C_L$	Internal Equivalent Load Capacitance	Integrated Load Capacitance (XIN and XOUT in series)	7.5	9.5	10.5	pF

Notes:

1.  $R_S$  is the series resistor

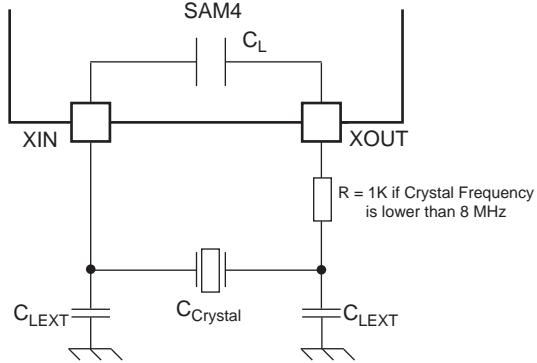
2.  $R_s = 100\text{-}200 \Omega$ ;  $C_s = 2.0\text{-}2.5 \text{ pF}$ ;  $C_m = 2\text{-}1.5 \text{ fF}$ (typ, worst case) using 1 kΩ serial resistor on XOUT.

3.  $R_s = 50\text{-}100 \Omega$ ;  $C_s = 2.0\text{-}2.5 \text{ pF}$ ;  $C_m = 4\text{-}3 \text{ fF}$ (typ, worst case).

4.  $R_s = 25\text{-}50 \Omega$ ;  $C_s = 2.5\text{-}3.0 \text{ pF}$ ;  $C_m = 7\text{-}5 \text{ fF}$  (typ, worst case).

5.  $R_s = 20\text{-}50 \Omega$ ;  $C_s = 3.2\text{-}4.0 \text{ pF}$ ;  $C_m = 10\text{-}8 \text{ fF}$ (typ, worst case).

Figure 46-12. 3 to 20 MHz Crystal Oscillator Schematics



$$C_{\text{LECT}} = 2x(C_{\text{CRYSTAL}} - C_{\text{L}} - C_{\text{PCB}})$$

where:

$C_{\text{PCB}}$  is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM4 pin.

#### 46.4.6 3 to 20 MHz Crystal Characteristics

Table 46-21. Crystal Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor (Rs)	Fundamental @ 3 MHz Fundamental @ 8 MHz Fundamental @ 12 MHz Fundamental @ 16 MHz Fundamental @ 20 MHz	—	—	200 100 80 80 50	Ω
$C_M$	Motional capacitance	—	—	—	8	fF
$C_{\text{SHUNT}}$	Shunt capacitance	—	—	—	7	pF

#### 46.4.7 3 to 20 MHz XIN Clock Input Characteristics in Bypass Mode

Table 46-22. XIN Clock Electrical Characteristics (In Bypass Mode)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$1/(t_{\text{CPXIN}})$	XIN Clock Frequency	(1)	—	—	50	MHz
$t_{\text{CPXIN}}$	XIN Clock Period	(1)	20	—	—	ns
$t_{\text{CHXIN}}$	XIN Clock High Half-period	(1)	8	—	—	ns
$t_{\text{CLXIN}}$	XIN Clock Low Half-period	(1)	8	—	—	ns
$t_{\text{CLCH}}$	Rise Time	(1)	2.2	—	—	ns
$t_{\text{CHCL}}$	Fall Time	(1)	2.2	—	—	ns
$V_{\text{XIN\_IL}}$	$V_{\text{XIN}}$ Input Low-level Voltage	(1)	-0.3	—	[0.8V:0. 3 x $V_{\text{VDDIO}}$ ]	V
$V_{\text{XIN\_IH}}$	$V_{\text{XIN}}$ Input High-level Voltage	(1)	[2.0V:0.7 x $V_{\text{VDDIO}}$ ]	—	$V_{\text{VDDIO}}$ +0.3V	V

Note: 1. These characteristics apply only when the 3-20 MHz XTAL Oscillator is in bypass mode.

## 46.4.8 Crystal Oscillator Design Considerations Information

### 46.4.8.1 Choosing a Crystal

When choosing a crystal for the 32768 Hz Slow Clock Oscillator or for the 3-20 MHz Oscillator, several parameters must be taken into account. Important parameters between crystal and SAM4S specifications are as follows:

- Load Capacitance
  - $C_{\text{crystal}}$  is the equivalent capacitor value the oscillator must “show” to the crystal in order to oscillate at the target frequency. The crystal must be chosen according to the internal load capacitance ( $C_L$ ) of the on-chip oscillator. Having a mismatch for the load capacitance will result in a frequency drift.
- Drive Level
  - Crystal drive level  $\geq$  Oscillator Drive Level. Having a crystal drive level number lower than the oscillator specification may damage the crystal.
- Equivalent Series Resistor (ESR)
  - Crystal ESR  $\leq$  Oscillator ESR Max. Having a crystal with ESR value higher than the oscillator may cause the oscillator to not start.
- Shunt Capacitance
  - Max. crystal Shunt capacitance  $\leq$  Oscillator Shunt Capacitance ( $C_{\text{SHUNT}}$ ). Having a crystal with ESR value higher than the oscillator may cause the oscillator to not start.

### 46.4.8.2 Printed Circuit Board (PCB)

SAM4E Oscillators are low power oscillators requiring particular attention when designing PCB systems.

## 46.5 PLLA Characteristics

**Table 46-23. Supply Voltage Phase Lock Loop Characteristics**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
VDDPLL	Supply Voltage Range		1.08	1.2	1.32	V
	Allowable Voltage Ripple	RMS Value 10 kHz to 10 MHz RMS Value > 10 MHz	—	—	20 10	mV

**Table 46-24. PLLA Characteristics**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$F_{\text{IN}}$	Input Frequency		3	—	32	MHz
$F_{\text{OUT}}$	Output Frequency		80	—	240	MHz
$I_{\text{PLL}}$	Current Consumption	Active mode @ 80 MHz @1.2V Active mode @ 96 MHz @1.2V Active mode @ 160 MHz @1.2V Active Mode @ 240 MHz @1.2V	—	0.94 1.2 2.1 3.34	1.2 1.5 2.5 4	mA
$T_{\text{START}}$	Settling Time		—	60	150	$\mu$ s

## 46.6 USB Transceiver Characteristics

### 46.6.1 Typical Connection

For typical connection please refer to the USB Device Section.

### 46.6.2 Electrical Characteristics

**Table 46-25. Electrical Parameters**

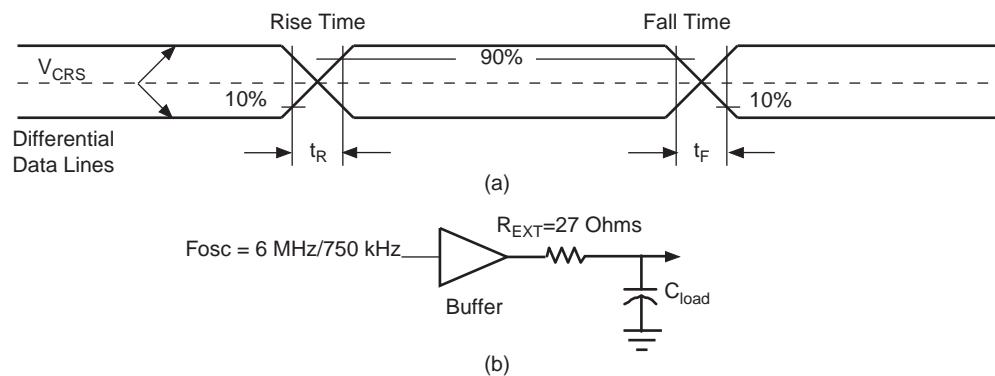
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
<b>Input Levels</b>						
$V_{IL}$	Low Level		—	—	0.8	V
$V_{IH}$	High Level		2.0	—	—	V
$V_{DI}$	Differential Input Sensitivity	$ (D+) - (D-) $	0.2	—	—	V
$V_{CM}$	Differential Input Common Mode Range		0.8	—	2.5	V
$C_{IN}$	Transceiver capacitance	Capacitance to ground on each line	—	—	9.18	pF
I	Hi-Z State Data Line Leakage	$0V < V_{IN} < 3.3V$	-10	—	+10	$\mu A$
$R_{EXT}$	Recommended External USB Series Resistor	In series with each USB pin with $\pm 5\%$	—	27	—	$\Omega$
<b>Output Levels</b>						
$V_{OL}$	Low Level Output	Measured with $R_L$ of 1.425 k $\Omega$ tied to 3.6V	0.0	—	0.3	V
$V_{OH}$	High Level Output	Measured with $R_L$ of 14.25 k $\Omega$ tied to GND	2.8	—	3.6	V
$V_{CRS}$	Output Signal Crossover Voltage	Measure conditions described in <a href="#">Figure 46-13 “USB Data Signal Rise and Fall Times”</a>	1.3	—	2.0	V
<b>Consumption</b>						
$I_{VDDIO}$	Current Consumption	Transceiver enabled in input mode DDP = 1 and DDM = 0	—	105	200	$\mu A$
$I_{VDDCORE}$	Current Consumption		—	80	150	$\mu A$
<b>Pull-up Resistor</b>						
$R_{PUI}$	Bus Pull-up Resistor on Upstream Port (idle bus)		0.900	—	1.575	k $\Omega$
$R_{PUA}$	Bus Pull-up Resistor on Upstream Port (upstream port receiving)		1.425	—	3.090	k $\Omega$

### 46.6.3 Switching Characteristics

Table 46-26. In Full Speed

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$t_{FR}$	Transition Rise Time	$C_{LOAD} = 50 \text{ pF}$	4	—	20	ns
$t_{FE}$	Transition Fall Time	$C_{LOAD} = 50 \text{ pF}$	4	—	20	ns
$t_{FRFM}$	Rise/Fall time Matching		90	—	111.11	%

Figure 46-13. USB Data Signal Rise and Fall Times



## 46.7 12-bit AFE (Analog Front End) Characteristics

Figure 46-14. 12-bit AFE (Analog Front End) Diagram

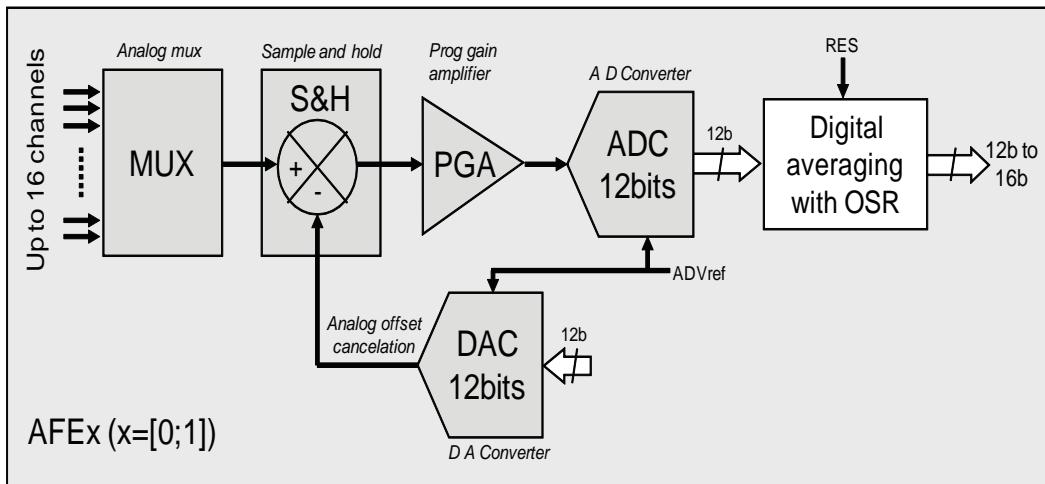


Table 46-27. Analog Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_{VDDIN}$	ADC Analog Supply	12-bit resolution	2.4	3.0	3.6	V
	Max. Voltage Ripple	RMS value, 10 kHz to 20 MHz	—	—	20	mV
$I_{VDDIN}$	Current Consumption	Sleep Mode (Clock OFF) Fast Wake-up Mode (Standby CLK = 20 MHz, IBCTL= 01) Normal Mode (IBCTL= 01)	—	4 1.8 3.8	8 3 6	$\mu$ A mA mA

Note: Use IBCTL = 00 for Sampling Frequency below 500 kHz and IBCTL = 01 between 500 kHz and 1 MHz.

Table 46-28. Channel Conversion Time and ADC Clock

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$f_{ADC}$	ADC Clock Frequency		1	—	20	MHz
$t_{CP\_ADC}$	ADC Clock Period		50	—	1000	ns
$f_S$	Sampling Frequency		—	—	1	MHz
$t_{START-UP}$	ADC Startup time	From OFF Mode to Normal Mode: - Voltage Reference OFF - Analog Circuitry OFF  From Standby Mode to Normal Mode: - Voltage Reference ON - Analog Circuitry OFF	—	16 4	32 8	$\mu$ s
$t_{TRACKTIM}$	Track and Hold Time	See <a href="#">Section 46.7.3.1 "Track and Hold Time versus Source Output Impedance"</a> for more details	160	—	—	ns
$t_{CONV}$	Conversion Time		—	20	—	$T_{CP\_ADC}$
$t_{cal}$	Calibration Time		—	—	306	$T_{CP\_ADC}$
$t_{SETTLE}$	Settling Time	Settling time to change offset and gain	100	—	—	ns

**Table 46-29. External Voltage Reference Input**

Parameter	Conditions	Min	Typ	Max	Units
ADVREF Input Voltage Range, 12-bit	$2.4V < V_{VDDIN} < 3.6V$	2.4	3	VDDIN	V
ADVREF Current (ADVref + DAC Current)		—	1	1.5	mA
ADVREF Input DC Impedance		2.4	3	10	kΩ

#### 46.7.1 ADC Resolution

##### 46.7.1.1 Conditions @ 25 degrees with Gain =1

- $f_{ADC} = 20$  MHz;  $f_{ADC} = 2$  MHz for INL and DNL static measurement only
- $f_S = 1$  MHz, ADC Sampling Frequency in Free Run Mode
- ADVref = 3V
- Signal Amplitude: ADVref/2, Signal Frequency < 100 Hz
- OSR: Number of Averaged Samples
- VDDIN = 2.4V

**Table 46-30. ADC Resolution following Digital Averaging**

Parameter Averaging Resolution RES (AFEC_EMR)	OSR (Over Sampling Ratio)	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (Bits)	FS (KSpS)
<b>Single Ended Mode</b>								
RES = 0	1	12	+/-1	+/-0.5	63.5	-83	10.2	1000
RES = 2	4	13	+/-1	+/-1	68.3	-84.5	11	250
RES = 3	16	14	+4 / -2	+3.2 / -1	73	-85.7	11.8	62.5
RES = 4	64	15	+6 / -3.5	—	76.8	-85.8	12.4	15.6
RES = 5	256	16	+15 / -8	—	82.7	-86.2	13.2	3.9
<b>Differential Mode</b>								
RES = 0	1	12	+/-1	+/-0.5	64	-83	10.3	1000
RES = 2	4	13	+/-1	+/-1	69.2	-83.7	11.2	250
RES = 3	16	14	+3 / -1.5	+3 / -1	74.8	-84.5	12.1	62.5
RES = 4	64	15	+6 / -3.5	—	80.2	-84.5	12.8	15.6
RES = 5	256	16	+10 / -7	—	83.9	-84.9	13.2	3.9

#### 46.7.1.2 Conditions @ 25 degrees with Gain =4

- $f_{ADC} = 20$  MHz
- $f_S = 1$  MHz, ADC Sampling Frequency in Free Run Mode
- ADVref = 3V
- Signal Amplitude: ADVref/2, Signal Frequency < 100 Hz
- OSR: Number of Averaged Samples

Table 46-31. ADC Resolution following Digital Averaging

Parameter Averaging Resolution RES (AFEC_EMR)	OSR (Over Sampling Ratio)	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (Bits)	FS (KSPS)
<b>Single Ended Mode</b>								
RES = 0	1	12	+/-1	+/-0.5	59	-81	9.5	1000
RES = 2	4	13	+1.7 / -1.3	+1.6 / -1	63.1	-82.9	10.2	250
RES = 3	16	14	+1.7 / -2.5	+2 / -1	67	-83.6	10.8	62.5
RES = 4	64	15	+/-8	—	70.3	-84.5	11.4	15.6
RES = 5	256	16	+/-12	—	74.8	-85.1	12.1	3.9
<b>Differential Mode</b>								
RES = 0	1	12	+/-1	+/-0.5	62	-84.5	10	1000
RES = 2	4	13	+/-1	+/-1	67.7	-85.7	10.9	25
RES = 3	16	14	+4.1 / -1.6	+3.4 / -1	73.6	-86.8	11.9	6.25
RES = 4	64	15	+/-3.5	—	78.7	-86.8	12.7	1.56
RES = 5	256	16	+/-7.5	—	82.1	-86.8	13.1	0.39

#### 46.7.2 Static Performance Characteristics

In the tables that follow, the LSB is relative to analog scale:

- Single Ended (ex: ADVREF=3.0V),
  - Gain = 1, LSB =  $(3.0V / 4096) = 732 \mu V$
  - Gain = 2, LSB =  $(1.5V / 4096) = 366 \mu V$
  - Gain = 4, LSB =  $(750 mV / 4096) = 183 \mu V$
- Differential (ex: ADVREF=3.0V),
  - Gain = 0.5, LSB =  $(6.0V / 4096) = 1465 \mu V$
  - Gain = 1, LSB =  $(3.0V / 4096) = 732 \mu V$
  - Gain = 2, LSB =  $(1.5V / 4096) = 366 \mu V$

**Table 46-32. INL, DNL, 12-bit mode, VDDIN 2.4V to 3.6V Supply Voltage Conditions for All Gains**

Parameter	Conditions	Min	Typ	Max	Units
Resolution		12			bit
Integral Non-linearity (INL)	$f_{ADC} = 2 \text{ MHz}$ ; differential mode or single mode, Gain = xx	-2	+/-1	2	LSB
Differential Non-linearity (DNL)	$f_{ADC} = 2 \text{ MHz}$ ; differential mode or single mode, Gain = xx	-1	+/-0.5	1	LSB

**Table 46-33. Gain and Error Offset, 12-bit Mode, VDDIN 2.4V to 3.6V Supply Voltage Conditions**

Parameter	Conditions	Min	Typ	Max	Units
Offset Error (Without DAC Compensation)	Differential mode, Gain = 0.5	-18	—	18	LSB
	Differential mode, Gain = 1	-35	—	35	
	Differential mode, Gain = 2	-60	—	60	
	Single Ended Gain = 1	-18	—	18	
	Single Ended Gain = 2	-35	—	35	
	Single Ended Gain = 4	-60	—	60	
Gain Error, after calibration	Differential mode, Gain = xx	-50	—	0	LSB
	Single Ended Gain = xx	-25	—	0	

### 46.7.3 Dynamic Performance Characteristics

Table 46-34. Dynamic Performance Characteristics in Single-ended and 12-bit Mode <sup>(1)</sup>

Parameter	Conditions	Min	Typ	Max	Units
Signal to Noise Ratio - SNR		56	64	72	dB
Total Harmonic Distortion - THD			-74	-66	dB
Signal to Noise and Distortion - SINAD		55	62	—	dB
Effective Number of Bits ENOB		9	10	—	bits

Note: 1. ADC Clock ( $f_{ADC}$ ) = 20 MHz,  $f_S = 1$  MHz, Fin = 127 kHz, IBCTL = 01, FFT using 1024 points or more, Frequency band = [1 kHz, 500 kHz] – Nyquist conditions fulfilled.

Table 46-35. Dynamic Performance Characteristics in Differential and 12-bit Mode <sup>(1)</sup>

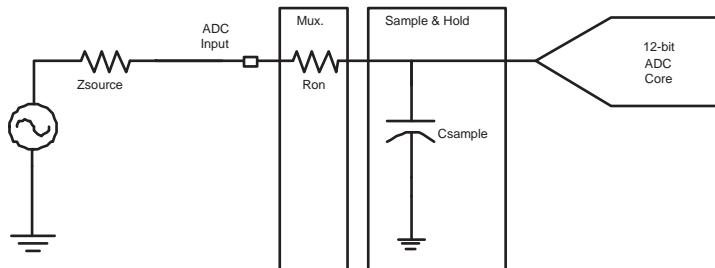
Parameter	Conditions	Min	Typ	Max	Units
Signal to Noise Ratio - SNR		60	64	74	dB
Total Harmonic Distortion - THD			-84	-76	dB
Signal to Noise and Distortion - SINAD		60	64	—	dB
Effective Number of Bits ENOB		9.5	10.5	—	bits

Note: 1. ADC Clock ( $f_{ADC}$ )= 20 MHz,  $f_S = 1$  MHz, Fin = 127 kHz, IBCTL = 01, FFT using 1024 points or more, Frequency band = [1 kHz, 500 kHz] – Nyquist conditions fulfilled.

#### 46.7.3.1 Track and Hold Time versus Source Output Impedance

Figure 46-15 shows a simplified acquisition path.

Figure 46-15. Simplified Acquisition Path



During the tracking phase the ADC needs to track the input signal during the tracking time shown below:

- 12-bit mode:  $t_{TRACK} = 0.054 \times Z_{SOURCE} + 205$

with  $t_{TRACK}$  expressed in ns and  $Z_{SOURCE}$  expressed in  $\Omega$ .

Two cases must be considered:

1. The calculated tracking time ( $t_{TRACK}$ ) is lower than  $15 t_{CP\_ADC}$

Set TRANSFER = 1 and TRACTIM = 0.

In this case, the allowed  $Z_{SOURCE}$  can be computed versus the ADC frequency with the hypothesis of:

$$t_{TRACK} = 15 \times t_{CP\_ADC}$$

where  $t_{CP\_ADC} = 1/f_{ADC}$ .

**Table 46-36. Source Impedance Values**

$f_{ADC}$ = ADC Clock (MHz)	$Z_{SOURCE}$ (kΩ) for 12 bits
20.00	10
16.00	14
10.67	22
8.00	31
6.40	40
5.33	48
4.57	57
4.00	66
3.56	74
3.20	83
2.91	92
2.67	100
2.46	109
2.29	118
2.13	126
2.00	135
1.00	274

2. The calculated tracking time ( $t_{TRACK}$ ) is higher than  $15 t_{CP\_ADC}$

Set TRANSFER = 1 and TRACTIM = 0.

In this case, a timer will trigger the ADC in order to set the correct sampling rate according to the Track time.

The maximum possible sampling frequency will be defined by  $t_{TRACK}$  in nano seconds, computed by the previous formula but with minus  $15 \times t_{CP\_ADC}$  and plus TRANSFER time:

- 12-bit mode:  $1/f_S = t_{TRACK} - 15 \times t_{CP\_ADC} + 5 t_{CP\_ADC}$

Note:  $C_{SAMPLE}$  and  $R_{ON}$  are taken into account in the formulas.

**Table 46-37. Analog Inputs**

Parameter	Min	Typ	Max	Units
Input Voltage Range	0	—	$V_{ADVREF}$	
Input Leakage Current	—	—	$\pm 0.5$	$\mu A$
Input Capacitance	—	—	8	pF

Note: 1. Input Voltage range can be up to VDDIN without destruction or over-consumption.

If  $V_{DDIO} < V_{ADVREF}$  max input voltage is VDDIO.

#### 46.7.3.2 AFE DAC Offset Compensation

**Table 46-38. AFE DAC Offset Compensation**

Parameter	Conditions	Min	Typ	Max	Units
Resolution - N		—	12	—	bits
INL	Range [32 to 4063]	-4	—	+4	LSB
DNL		-2	—	+2	LSB
LSB relative to Vrefin Scale	$LSB = V_{refin} / 2^{12}$	—	732	—	uV

## 46.8 12-bit DAC Characteristics

Table 46-39. Analog Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_{VDDIN}$	Analog Supply		2.4	3.0	3.6	V
	Max. Voltage Ripple	RMS value, 10 kHz to 20 MHz	—	—	20	mV
$I_{VDDIN}$	Current Consumption	Sleep Mode (Clock OFF)	—	—	3	$\mu$ A
		Fast Wake-up (Standby Mode, Clock on)	—	2	3	mA
		Normal Mode with 1 Output On (IBCTLADCORE = 01, IBCTLCHx = 10)	—	4.3	5.6	mA
		Normal Mode with 2 Outputs On (IBCTLADCORE = 01, IBCTLCHx = 10)	—	5	6.5	mA

Table 46-40. Channel Conversion Time and DAC Clock

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$F_{DAC}$	Clock Frequency		1	—	50	MHz
$T_{CP\_DAC}$	Clock Period		—	—	20	ns
$f_s$	Sampling Frequency		—	—	2	MHz
$T_{START-UP}$	Startup time	From Sleep Mode to Normal Mode: - Voltage Reference OFF - DAC Core OFF	20	30	40	μs
		From Fast Wake-Up to Normal Mode: - Voltage Reference ON - DAC Core OFF	2.5	3.75	5	
$T_{CONV}$	Conversion Time		—	—	25	$T_{CP\_DAC}$

External voltage reference for DAC is ADVREF. See the ADC voltage reference characteristics [Table 46-29 on page 1331](#).

Table 46-41. Static Performance Characteristics

Parameter	Conditions	Min	Typ	Max	Units
Resolution		—	12	—	bit
Integral Non-linearity (INL)	$2.4V < V_{VDDIN} < 2.7V$	-6	—	+6	LSB
	$2.7V < V_{VDDIN} < 3.6V$	-2.5	$\pm 1$	+2.5	
Differential Non-linearity (DNL)	$2.4V < V_{VDDIN} < 2.7V$	-2.5	$\pm 1$	+2.5	LSB
	$2.7V < V_{VDDIN} < 3.6V$				
Offset Error		-32	$\pm 8$	32	LSB
Gain Error		-32	$\pm 2$	32	LSB

Note: DAC Clock ( $f_{DAC}$ ) = 5 MHz,  $f_s$  = 200 kHz, IBCTL = 01.

**Table 46-42. Dynamic Performance Characteristics**

Parameter	Conditions	Min	Typ	Max	Units
Signal to Noise Ratio - SNR	2.4V < V <sub>VDDIN</sub> < 2.7V	47	58	70	dB
	2.7V < V <sub>VDDIN</sub> < 3.6V	56	61	74	
Total Harmonic Distortion - THD	2.4V < V <sub>VDDIN</sub> < 2.7V	—	-72	-60	dB
	2.7V < V <sub>VDDIN</sub> < 3.6V	—	-76	-68	
Signal to Noise and Distortion - SINAD	2.4V < V <sub>VDDIN</sub> < 2.7V	47	58	—	dB
	2.7V < V <sub>VDDIN</sub> < 3.6V	56	61	—	
Effective Number of Bits ENOB	2.4V < V <sub>VDDIN</sub> < 2.7V	7.5	9	12	bits
	2.7V < V <sub>VDDIN</sub> < 3.6V	9	10	12	

Note: DAC Clock ( $f_{DAC}$ ) = 50 MHz,  $f_S = 2$  MHz, Fin = 241 kHz, IBCTL = 01, FFT using 1024 points or more, Frequency band = [10 kHz, 1 MHz] – Nyquist conditions fulfilled.

**Table 46-43. Analog Outputs**

Parameter	Conditions	Min	Typ	Max	Units
Voltage Range		(1/6) x V <sub>ADVREF</sub>		(5/6) x V <sub>ADVREF</sub>	V
Slew Rate	Channel Output Current versus Slew Rate (IBCTL for DAC0 or DAC1, noted IBCTLCHx) R <sub>LOAD</sub> = 10 kΩ/0 pF < C <sub>LOAD</sub> < 50 pF IBCTLCHx = 00	—	2.7	—	V/μs
	IBCTLCHx = 01	—	5.3	—	
	IBCTLCHx = 10	—	8	—	
	IBCTLCHx = 11	—	10.7	—	
	No resistive load IBCTLCHx = 00	—	0.23	—	
Output Channel Current Consumption	IBCTLCHx = 01	—	0.45	—	mA
	IBCTLCHx = 10	—	0.67	—	
	IBCTLCHx = 11	—	0.89	—	
Settling Time	R <sub>LOAD</sub> = 10 kΩ/0 pF < C <sub>LOAD</sub> < 50 pF,	—	—	0.5	μs
R <sub>LOAD</sub>	Output Load Resistor	10	—	—	kΩ
C <sub>LOAD</sub>	Output Load Capacitor	—	30	50	pF

## 46.9 Analog Comparator Characteristics

Table 46-44. Analog Comparator Characteristics

Parameter	Conditions	Min	Typ	Max	Units
Voltage Range	The Analog Comparator is supplied by VDDIN	1.62	3.3	3.6	V
Input Voltage Range		GND + 0.2	—	VDDIN - 0.2	V
Input Offset Voltage		—	—	20	mV
Current Consumption	On VDDIN	—	—	—	—
	Low Power Option (ISEL = 0)	—	—	25	µA
	High Speed Option (ISEL = 1)			170	
Hysteresis	HYST = 0x01 or 0x10	—	15	50	mV
	HYST = 0x11		30	90	
Settling Time	Given for overdrive > 100 mV	—	v	—	—
	Low Power Option	—	v	1	µs
	High Speed Option			0.1	

## 46.10 Temperature Sensor

The temperature sensor is connected to channel 15 of the ADC.

The temperature sensor provides an output voltage ( $V_T$ ) that is proportional to absolute temperature (PTAT). The  $V_T$  output voltage linearly varies with a temperature slope  $dV_T/dT = 4.7 \text{ mV/}^\circ\text{C}$ .

The  $V_T$  voltage equals 1.44V at  $27^\circ\text{C}$ , with a  $\pm 60 \text{ mV}$  accuracy. The  $V_T$  slope versus temperature  $dV_T/dT = 4.7 \text{ mV/}^\circ\text{C}$  only shows a  $\pm 7\%$  slight variation over process, mismatch and supply voltage.

The user needs to calibrate it (offset calibration) at ambient temperature in order to get rid of the  $V_T$  spread at ambient temperature ( $\pm 15\%$ ).

Table 46-45. Temperature Sensor Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_T$	Output Voltage	$T^\circ = 27^\circ\text{ C}$ <sup>(1)</sup>	—	1.44		V
$\Delta V_T$	Output Voltage Accuracy	$T^\circ = 27^\circ\text{ C}$ <sup>(1)</sup>	-60	—	+60	mV
$dV_T/dT$	Temperature Sensitivity (Slope Voltage versus Temperature)	<sup>(1)</sup>	—	4.7		$\text{mV/}^\circ\text{C}$
	Slope Accuracy	Over temperature range [-40°C / +105°C] <sup>(1)</sup>	-7	—	+7	%
	Temperature Accuracy <sup>(2)</sup>	After offset calibration over temperature range [-40°C / +105°C]	-6	—	+6	$^\circ\text{C}$
		After offset calibration over temperature range [0°C / +80°C]	-5	—	+5	$^\circ\text{C}$
$T_{\text{START-UP}}$	Startup Time	After TSON = 1 <sup>(1)</sup>	—	5	10	µs
$I_{\text{VDDCORE}}$	Current Consumption	<sup>(1)</sup>	50	70	80	µA

Notes: 1. The value of TS only (the value do not take into account the ADC offset/gain/errors).

2. The temperature accuracy is taking account of the ADC offset error, gain error in single ended mode with Gain=1.

## 46.11 AC Characteristics

### 46.11.1 Master Clock Characteristics

Table 46-46. Master Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Units
1/(t <sub>CPMCK</sub> )	Master Clock Frequency	VDDCORE @ 1.20V	—	120	MHz
1/(t <sub>CPMCK</sub> )	Master Clock Frequency	VDDCORE @ 1.08V	—	100	MHz

### 46.11.2 I/O Characteristics

Criteria used to define the maximum frequency of the I/Os:

- Output duty cycle (40%-60%)
- Minimum output swing: 100 mV to VDDIO - 100 mV
- Maximum output swing: 100 mV to VDDIO - 100 mV
- Addition of rising and falling time inferior to 75% of the period

Table 46-47. I/O Characteristics

Symbol	Parameter	Conditions		Min	Max	Units
FreqMax1	Pin Group 1 <sup>(1)</sup> Maximum output frequency	10 pF	V <sub>DDIO</sub> = 1.62V	—	70	MHz
		30 pF	V <sub>DDIO</sub> = 1.62V	—	45	
PulseminH <sub>1</sub>	Pin Group 1 <sup>(1)</sup> High Level Pulse Width	10 pF	V <sub>DDIO</sub> = 1.62V	7.2	—	ns
		30 pF	V <sub>DDIO</sub> = 1.62V	11	—	
PulseminL <sub>1</sub>	Pin Group 1 <sup>(1)</sup> Low Level Pulse Width	10 pF	V <sub>DDIO</sub> = 1.62V	7.2	—	ns
		30 pF	V <sub>DDIO</sub> = 1.62V	11	—	
FreqMax2	Pin Group 2 <sup>(2)</sup> Maximum output frequency	10 pF	V <sub>DDIO</sub> = 1.62V	—	46	MHz
		25 pF	V <sub>DDIO</sub> = 1.62V	—	23	
PulseminH <sub>2</sub>	Pin Group 2 <sup>(2)</sup> High Level Pulse Width	10 pF	V <sub>DDIO</sub> = 1.62V	11	—	ns
		25 pF	V <sub>DDIO</sub> = 1.62V	21.8	—	
PulseminL <sub>2</sub>	Pin Group 2 <sup>(2)</sup> Low Level Pulse Width	10 pF	V <sub>DDIO</sub> = 1.62V	11	—	ns
		25 pF	V <sub>DDIO</sub> = 1.62V	21.8	—	
FreqMax3	Pin Group 3 <sup>(3)</sup> Maximum output frequency	10 pF	V <sub>DDIO</sub> = 1.62V	—	70	MHz
		25 pF	V <sub>DDIO</sub> = 1.62V	—	35	
PulseminH <sub>3</sub>	Pin Group 3 <sup>(3)</sup> High Level Pulse Width	10 pF	V <sub>DDIO</sub> = 1.62V	7.2	—	ns
		25 pF	V <sub>DDIO</sub> = 1.62V	14.2	—	
PulseminL <sub>3</sub>	Pin Group 3 <sup>(3)</sup> Low Level Pulse Width	10 pF	V <sub>DDIO</sub> = 1.62V	7.2	—	ns
		25 pF	V <sub>DDIO</sub> = 1.62V	14.2	—	
FreqMax4	Pin Group 4 <sup>(4)</sup> Maximum output frequency	10 pF	V <sub>DDIO</sub> = 1.62V	—	58	MHz
		25 pF	V <sub>DDIO</sub> = 1.62V	—	29	
PulseminH <sub>4</sub>	Pin Group 4 <sup>(4)</sup> High Level Pulse Width	10 pF	V <sub>DDIO</sub> = 1.62V	8.6	—	ns
		25 pF	V <sub>DDIO</sub> = 1.62V	17.2	—	
PulseminL <sub>4</sub>	Pin Group 4 <sup>(4)</sup> Low Level Pulse Width	10 pF	V <sub>DDIO</sub> = 1.62V	8.6	—	ns
		25 pF	V <sub>DDIO</sub> = 1.62V	17.2	—	
FreqMax5	Pin Group 5 <sup>(5)</sup> Maximum output frequency	25 pF	V <sub>DDIO</sub> = 1.62V	—	25	MHz

Notes: 1. Pin Group 1 = PA14, PA29

2. Pin Group 2 = PA[4], PA[9-11], PA[15-25], PB[0-7], PB[12-13], PC[0-31], PD[2], PD[18:31], PE[0-5]

3. Pin Group 3 = PA[5:8], PA[12-13], PA[26-28], PA[30-31], PB[8:9], PB[14], PD[0:1], PD[3:17]

4. Pin Group 4 = PA[0-3]

5. Pin Group 5 = PB[10-11]

#### 46.11.3 SPI Characteristics

Figure 46-16. SPI Master Mode with (CPOL= NCPHA = 0) or (CPOL= NCPHA= 1)

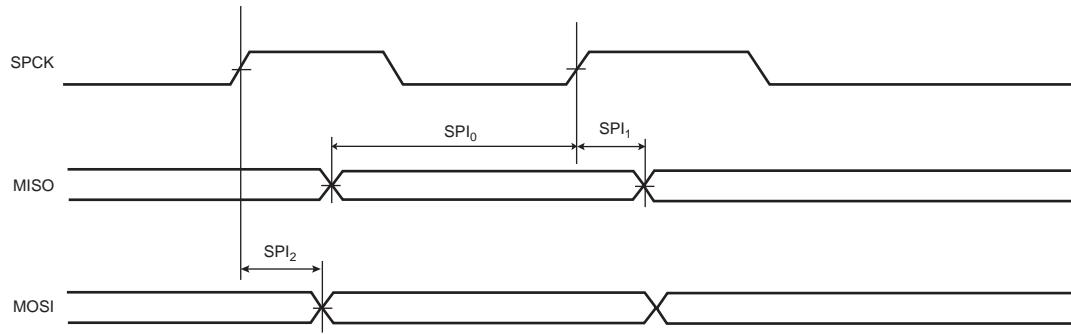


Figure 46-17. SPI Master Mode with (CPOL = 0 and NCPHA=1) or (CPOL=1 and NCPHA= 0)

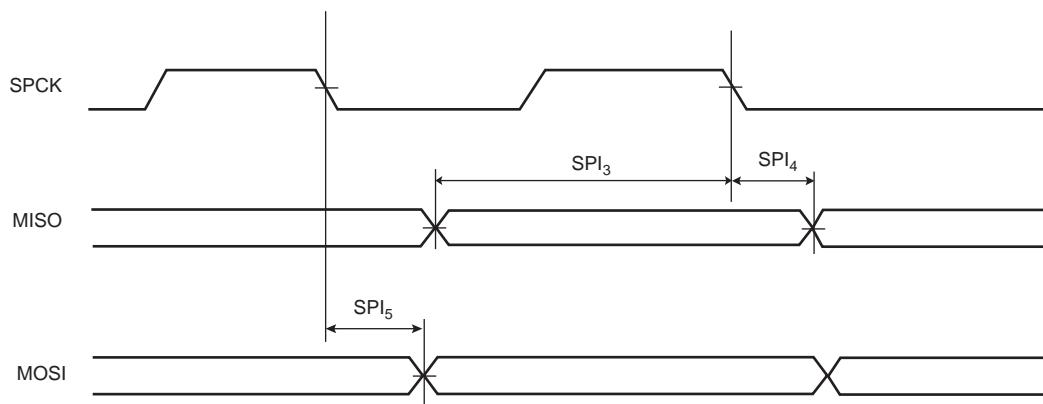


Figure 46-18. SPI Slave Mode with (CPOL=0 and NCPHA=1) or (CPOL=1 and NCPHA=0)

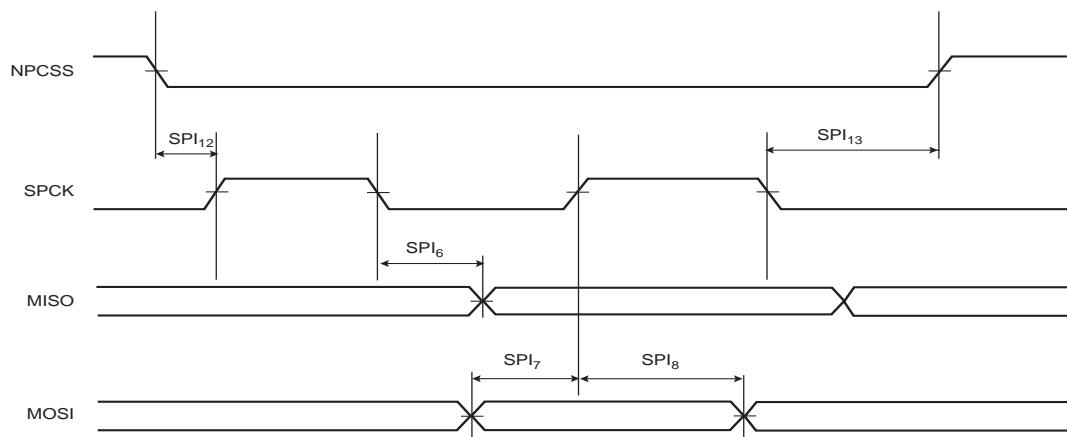
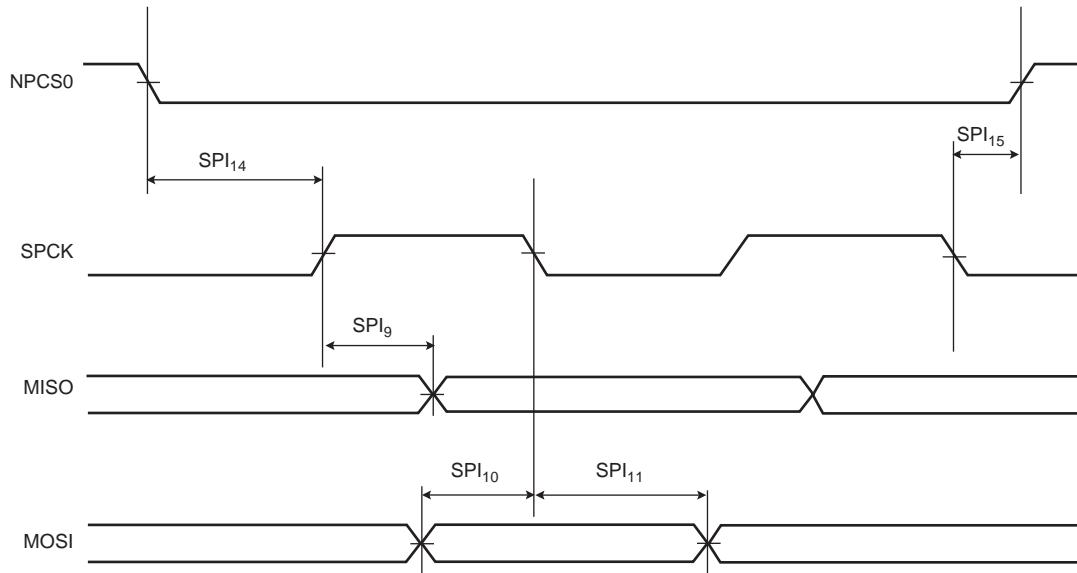


Figure 46-19. SPI Slave Mode with (CPOL = NCPHA = 0) or (CPOL= NCPHA= 1)



#### 46.11.3.1 Maximum SPI Frequency

The following formulas give maximum SPI frequency in Master read and write modes and in Slave read and write modes.

##### Master Write Mode

The SPI is only sending data to a slave device such as an LCD, for example. The limit is given by SPI<sub>2</sub> (or SPI<sub>5</sub>) timing. Since it gives a maximum frequency above the maximum pad speed (see [Section 46.11.2 “I/O Characteristics”](#)), the max SPI frequency is the one from the pad.

##### Master Read Mode

$$f_{SPCKMax} = \frac{1}{SPI_0(\text{or } SPI_3) + T_{valid}}$$

T<sub>VALID</sub> is the slave time response to output data after detecting an SPCK edge. For Atmel SPI DataFlash (AT45DB642D), T<sub>VALID</sub> (or T<sub>v</sub>) is 12 ns Max.

In the formula above, f<sub>SPCKMax</sub> = 33.0 MHz @ VDDIO = 3.3V.

##### Slave Read Mode

In slave mode, SPCK is the input clock for the SPI. The max SPCK frequency is given by setup and hold timings SPI<sub>7</sub>/SPI<sub>8</sub>(or SPI<sub>10</sub>/SPI<sub>11</sub>). Since this gives a frequency well above the pad limit, the limit in slave read mode is given by SPCK pad.

##### Slave Write Mode

$$f_{SPCKMax} = \frac{1}{2x(SPI_{6max}(\text{or } SPI_{9max}) + T_{setup})}$$

For 3.3V I/O domain and SPI<sub>6</sub>, f<sub>SPCKMax</sub> = 25 MHz. T<sub>SETUP</sub> is the setup time from the master before sampling data.

#### 46.11.3.2 SPI Timings

**Table 46-48. SPI Timings**

Symbol	Parameter	Conditions	Min	Max	Units
SPI <sub>0</sub>	MISO Setup time before SPCK rises (master)	3.3V domain <sup>(1)</sup>	11.0	—	ns
		1.8V domain <sup>(2)</sup>	12.5	—	ns
SPI <sub>1</sub>	MISO Hold time after SPCK rises (master)	3.3V domain <sup>(1)</sup>	0	—	ns
		1.8V domain <sup>(2)</sup>	0	—	ns
SPI <sub>2</sub>	SPCK rising to MOSI Delay (master)	3.3V domain <sup>(1)</sup>	-3.4	3.0	ns
		1.8V domain <sup>(2)</sup>	-3.2	1.9	ns
SPI <sub>3</sub>	MISO Setup time before SPCK falls (master)	3.3V domain <sup>(1)</sup>	18.0	—	ns
		1.8V domain <sup>(2)</sup>	19.8	—	ns
SPI <sub>4</sub>	MISO Hold time after SPCK falls (master)	3.3V domain <sup>(1)</sup>	0	—	ns
		1.8V domain <sup>(2)</sup>	0	—	ns
SPI <sub>5</sub>	SPCK falling to MOSI Delay (master)	3.3V domain <sup>(1)</sup>	-6.4	-1.9	ns
		1.8V domain <sup>(2)</sup>	-5.9	-2.6	ns
SPI <sub>6</sub>	SPCK falling to MISO Delay (slave)	3.3V domain <sup>(1)</sup>	3.6	11.9	ns
		1.8V domain <sup>(2)</sup>	4.2	13.9	ns
SPI <sub>7</sub>	MOSI Setup time before SPCK rises (slave)	3.3V domain <sup>(1)</sup>	0	—	ns
		1.8V domain <sup>(2)</sup>	0	—	ns
SPI <sub>8</sub>	MOSI Hold time after SPCK rises (slave)	3.3V domain <sup>(1)</sup>	2.8	—	ns
		1.8V domain <sup>(2)</sup>	2.3	—	ns
SPI <sub>9</sub>	SPCK rising to MISO Delay (slave)	3.3V domain <sup>(1)</sup>	3.8	12.1	ns
		1.8V domain <sup>(2)</sup>	4.2	13.6	ns
SPI <sub>10</sub>	MOSI Setup time before SPCK falls (slave)	3.3V domain <sup>(1)</sup>	0	—	ns
		1.8V domain <sup>(2)</sup>	0	—	ns
SPI <sub>11</sub>	MOSI Hold time after SPCK falls (slave)	3.3V domain <sup>(1)</sup>	2.4	—	ns
		1.8V domain <sup>(2)</sup>	2.5	—	ns
SPI <sub>12</sub>	NPSC setup to SPCK rising (slave)	3.3V domain <sup>(1)</sup>	3.2	—	ns
		1.8V domain <sup>(2)</sup>	3.3	—	ns
SPI <sub>13</sub>	NPSC hold after SPCK falling (slave)	3.3V domain <sup>(1)</sup>	0	—	ns
		1.8V domain <sup>(2)</sup>	0	—	ns
SPI <sub>14</sub>	NPSC setup to SPCK falling (slave)	3.3V domain <sup>(1)</sup>	3.8	—	ns
		1.8V domain <sup>(2)</sup>	3.3	—	ns
SPI <sub>15</sub>	NPSC hold after SPCK falling (slave)	3.3V domain <sup>(1)</sup>	0	—	ns
		1.8V domain <sup>(2)</sup>	0	—	ns

Notes: 1. 3.3V domain:  $V_{VDDIO}$  from 2.85V to 3.6V, maximum external capacitor = 40 pF.

2. 1.8V domain:  $V_{VDDIO}$  from 1.65V to 1.95V, maximum external capacitor = 20 pF.

Note that in SPI master mode, the SAM4E does not sample the data (MISO) on the opposite edge where the data clocks out (MOSI), but the same edge is used. See [Figure 46-16](#) and [Figure 46-17](#).

#### 46.11.4 HSMCI Timings

The High Speed MultiMedia Card Interface (HSMCI) supports the MultiMedia Card (MMC) Specification V4.3, the SD Memory Card Specification V2.0, the SDIO V2.0 specification and CE-ATA V1.1.

#### 46.11.5 SMC Timings

Timings are given in the following domains:

- 1.8V domain: VDDIO from 1.65V to 1.95V, maximum external capacitor = 30 pF
- 3.3V domain: VDDIO from 2.85V to 3.6V, maximum external capacitor = 50 pF.

Timings are given assuming a capacitance load on data, control and address pads:

In the tables that follow,  $t_{CPMCK}$  is MCK period.

##### 46.11.5.1 Read Timings

**Table 46-49. SMC Read Signals - NRD Controlled (READ\_MODE = 1)**

Symbol	Parameter	Min		Max		Units
	VDDIO Supply	1.8V <sup>(1)</sup>		3.3V <sup>(2)</sup>		1.8V <sup>(1)</sup> 3.3V <sup>(2)</sup>
<b>NO HOLD SETTINGS (nrd hold = 0)</b>						
SMC <sub>1</sub>	Data Setup before NRD High	20.6	18.6	—	—	ns
SMC <sub>2</sub>	Data Hold after NRD High	0	0	—	—	ns
<b>HOLD SETTINGS (nrd hold ≠ 0)</b>						
SMC <sub>3</sub>	Data Setup before NRD High	15.9	14.1	—	—	ns
SMC <sub>4</sub>	Data Hold after NRD High	0	0	—	—	ns
<b>HOLD or NO HOLD SETTINGS (nrd hold ≠ 0, nrd hold = 0)</b>						
SMC <sub>5</sub>	A0 - A22 Valid before NRD High	(nrd setup + nrd pulse) * $t_{CPMCK}$ - 6.8	(nrd setup + nrd pulse) * $t_{CPMCK}$ - 6.5	—	—	ns
SMC <sub>6</sub>	NCS low before NRD High	(nrd setup + nrd pulse - ncs rd setup) * $t_{CPMCK}$ - 4.6	(nrd setup + nrd pulse - ncs rd setup) * $t_{CPMCK}$ - 5.1	—	—	ns
SMC <sub>7</sub>	NRD Pulse Width	nrd pulse * $t_{CPMCK}$ - 7.5	nrd pulse * $t_{CPMCK}$ - 6.6	—	—	ns

**Table 46-50. SMC Read Signals - NCS Controlled (READ\_MODE= 0)**

Symbol	Parameter	Min		Max		Units
	VDDIO supply	1.8V <sup>(1)</sup>		3.3V <sup>(2)</sup>		1.8V <sup>(1)</sup> 3.3V <sup>(2)</sup>
<b>NO HOLD SETTINGS (ncs rd hold = 0)</b>						
SMC <sub>8</sub>	Data Setup before NCS High	21.8	19.4	—	—	ns
SMC <sub>9</sub>	Data Hold after NCS High	0	0	—	—	ns
<b>HOLD SETTINGS (ncs rd hold ≠ 0)</b>						
SMC <sub>10</sub>	Data Setup before NCS High	17.1	14.9	—	—	ns
SMC <sub>11</sub>	Data Hold after NCS High	0	0	—	—	ns

**Table 46-50. SMC Read Signals - NCS Controlled (READ\_MODE= 0)**

HOLD or NO HOLD SETTINGS (ncs rd hold ≠ 0, ncs rd hold = 0)						
SMC <sub>12</sub>	A0 - A22 valid before NCS High	(ncs rd setup + ncs rd pulse)* t <sub>CPMCK</sub> - 6.9	(ncs rd setup + ncs rd pulse)* t <sub>CPMCK</sub> - 6.6	—	—	ns
SMC <sub>13</sub>	NRD low before NCS High	(ncs rd setup + ncs rd pulse - nrd setup)* t <sub>CPMCK</sub> - 5.8	(ncs rd setup + ncs rd pulse - nrd setup)* t <sub>CPMCK</sub> - 5.5	—	—	ns
SMC <sub>14</sub>	NCS Pulse Width	ncs rd pulse length * t <sub>CPMCK</sub> - 7.6	ncs rd pulse length * t <sub>CPMCK</sub> - 6.7	—	—	ns

#### 46.11.5.2 Write Timings

**Table 46-51. SMC Write Signals - NWE Controlled (WRITE\_MODE = 1)**

Symbol	Parameter	Min		Max		Units
		1.8V <sup>(2)</sup>	3.3V <sup>(3)</sup>	1.8V <sup>(2)</sup>	3.3V <sup>(3)</sup>	
HOLD or NO HOLD SETTINGS (nwe hold ≠ 0, nwe hold = 0)						
SMC <sub>15</sub>	Data Out Valid before NWE High	nwe pulse * t <sub>CPMCK</sub> - 6.2	nwe pulse * t <sub>CPMCK</sub> - 5.9	—	—	ns
SMC <sub>16</sub>	NWE Pulse Width	nwe pulse * t <sub>CPMCK</sub> - 7.0	nwe pulse * t <sub>CPMCK</sub> - 6.1	—	—	ns
SMC <sub>17</sub>	A0 - A22 valid before NWE low	nwe setup * t <sub>CPMCK</sub> - 6.6	nwe setup * t <sub>CPMCK</sub> - 6.2	—	—	ns
SMC <sub>18</sub>	NCS low before NWE high	(nwe setup - ncs rd setup + nwe pulse) * t <sub>CPMCK</sub> - 5.9	(nwe setup - ncs rd setup + nwe pulse) * t <sub>CPMCK</sub> - 5.5	—	—	ns
HOLD SETTINGS (nwe hold ≠ 0)						
SMC <sub>19</sub>	NWE High to Data OUT, NBS0/A0 NBS1, NBS2/A1, NBS3, A2 - A25 change	nwe hold * t <sub>CPMCK</sub> - 9.4	nwe hold * t <sub>CPMCK</sub> - 7.6	—	—	ns
SMC <sub>20</sub>	NWE High to NCS Inactive <sup>(1)</sup>	(nwe hold - ncs wr hold) * t <sub>CPMCK</sub> - 6.0	(nwe hold - ncs wr hold) * t <sub>CPMCK</sub> - 5.6	—	—	ns
NO HOLD SETTINGS (nwe hold = 0)						
SMC <sub>21</sub>	NWE High to Data OUT, NBS0/A0 NBS1, NBS2/A1, NBS3, A2 - A25, NCS change <sup>(1)</sup>	3.3	3.2	—	—	ns

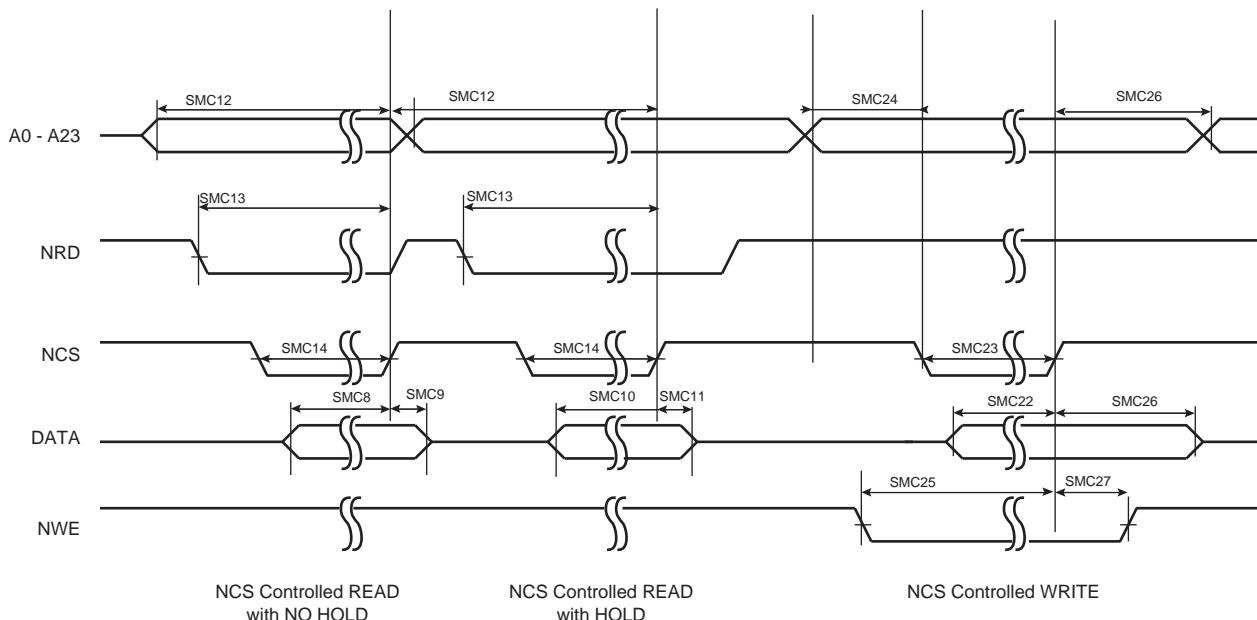
- Notes:
1. hold length = total cycle duration - setup duration - pulse duration. "hold length" is for "ncs wr hold length" or "NWE hold length".
  2. 1.8V domain: VDDIO from 1.65V to 1.95V, maximum external capacitor = 30 pF.
  3. 3.3V domain: VDDIO from 2.85V to 3.6V, maximum external capacitor = 50 pF.

**Table 46-52. SMC Write NCS Controlled (WRITE\_MODE = 0)**

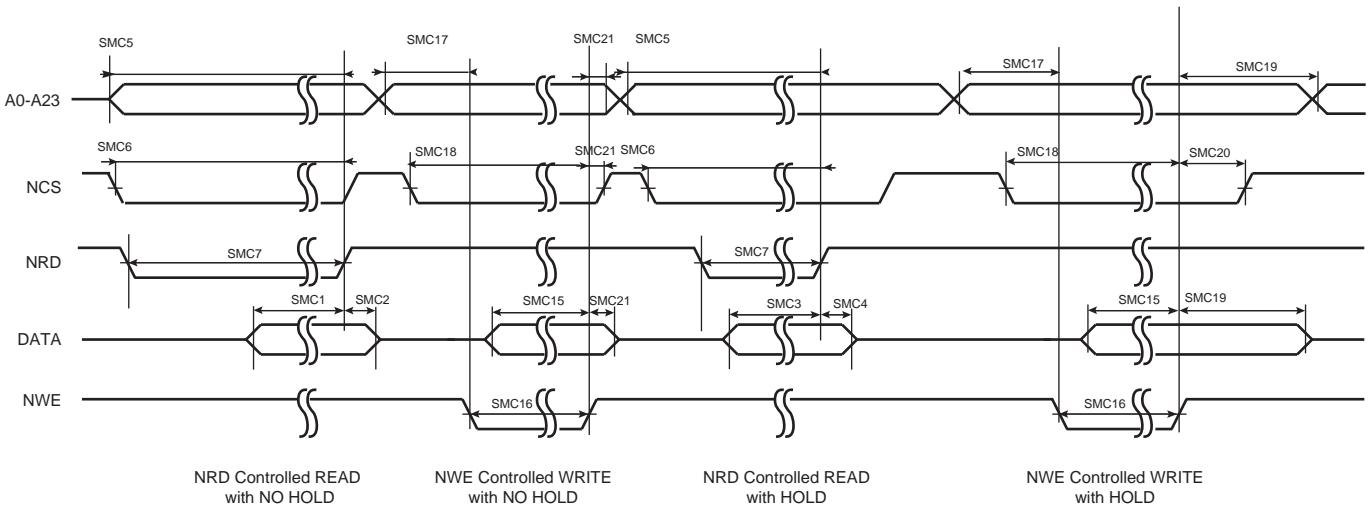
Symbol	Parameter	Min		Max		Units
		1.8V <sup>(1)</sup>	3.3V <sup>(2)</sup>	1.8V <sup>(1)</sup>	3.3V <sup>(2)</sup>	
SMC <sub>22</sub>	Data Out Valid before NCS High	ncs wr pulse * t <sub>CPMCK</sub> - 6.3	ncs wr pulse * t <sub>CPMCK</sub> - 6.0	—	—	ns
SMC <sub>23</sub>	NCS Pulse Width	ncs wr pulse * t <sub>CPMCK</sub> - 7.6	ncs wr pulse * t <sub>CPMCK</sub> - 6.7	—	—	ns
SMC <sub>24</sub>	A0 - A22 valid before NCS low	ncs wr setup * t <sub>CPMCK</sub> - 6.7	ncs wr setup * t <sub>CPMCK</sub> - 6.3	—	—	ns
SMC <sub>25</sub>	NWE low before NCS high	(ncs wr setup - nwe setup + ncs pulse) * t <sub>CPMCK</sub> - 5.6	(ncs wr setup - nwe setup + ncs pulse) * t <sub>CPMCK</sub> - 5.3	—	—	ns
SMC <sub>26</sub>	NCS High to Data Out,A0 - A25, change	ncs wr hold * t <sub>CPMCK</sub> - 10.6	ncs wr hold * t <sub>CPMCK</sub> - 9.0	—	—	ns
SMC <sub>27</sub>	NCS High to NWE Inactive	(ncs wr hold - nwe hold) * t <sub>CPMCK</sub> - 7.0	(ncs wr hold - nwe hold) * t <sub>CPMCK</sub> - 6.8	—	—	ns

Notes: 1. 1.8V domain: VDDIO from 1.65V to 1.95V, maximum external capacitor = 30 pF.  
 2. 3.3V domain: VDDIO from 2.85V to 3.6V, maximum external capacitor = 50 pF.

**Figure 46-20. SMC Timings - NCS Controlled Read and Write**



**Figure 46-21. SMC Timings - NRD Controlled Read and NWE Controlled Write**



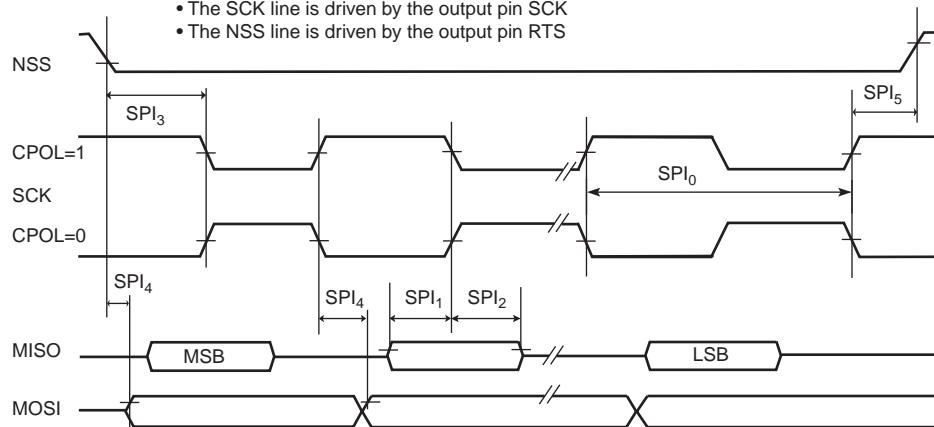
#### 46.11.6 USART in SPI Mode Timings

Timings are given in the following domains:

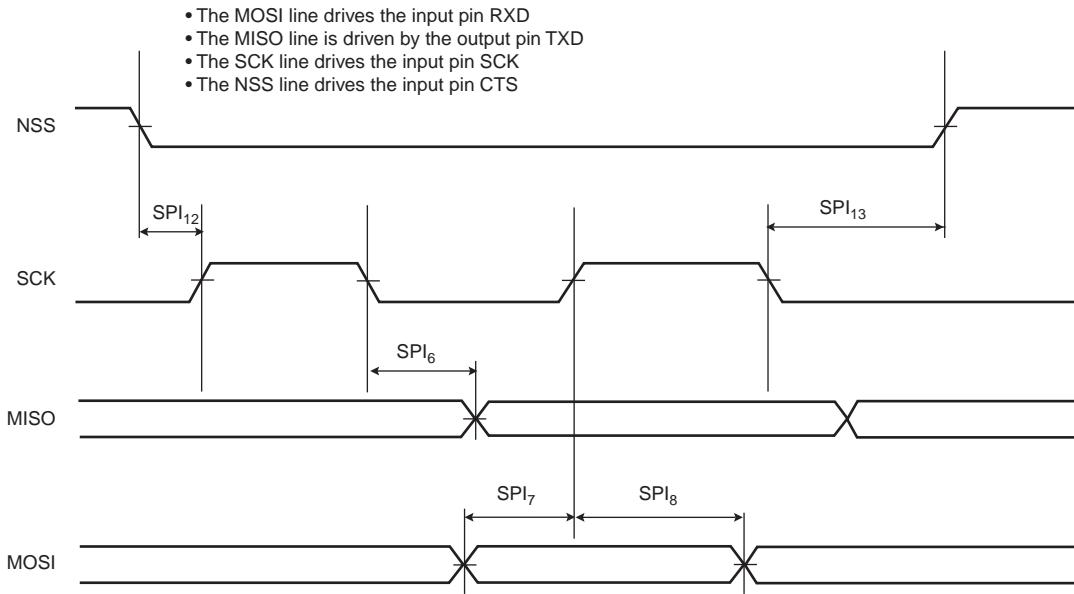
- 1.8V domain: VDDIO from 1.65V to 1.95V, maximum external capacitor = 20 pF
- 3.3V domain: VDDIO from 2.85V to 3.6V, maximum external capacitor = 40 pF.

**Figure 46-22. USART SPI Master Mode**

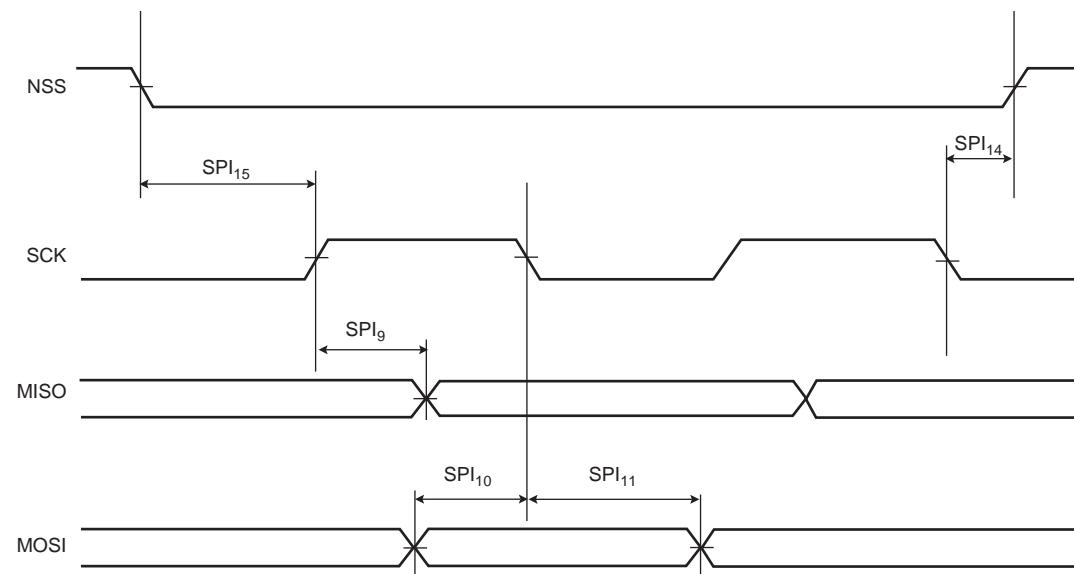
- The MOSI line is driven by the output pin TXD
- The MISO line drives the input pin RXD
- The SCK line is driven by the output pin SCK
- The NSS line is driven by the output pin RTS



**Figure 46-23. USART SPI Slave Mode (Mode 1 or 2)**



**Figure 46-24. USART SPI Slave Mode (Mode 0 or 3)**



#### 46.11.6.1 USART SPI Timings

**Table 46-53. USART SPI Timings**

Symbol	Parameter	Conditions	Min	Max	Units
<b>Master Mode</b>					
SPI <sub>0</sub>	SCK Period	1.8V domain 3.3V domain	MCK/6	—	ns
SPI <sub>1</sub>	Input Data Setup Time	1.8V domain 3.3V domain	0.5 * MCK + 3.3 0.5 * MCK + 3.7	—	ns
SPI <sub>2</sub>	Input Data Hold Time	1.8V domain 3.3V domain	1.5 * MCK + 0.8 1.5 * MCK + 1.1	—	ns
SPI <sub>3</sub>	Chip Select Active to Serial Clock	1.8V domain 3.3V domain	1.5 * SPCK - 1.4 1.5 * SPCK - 1.9	—	ns
SPI <sub>4</sub>	Output Data Setup Time	1.8V domain 3.3V domain	- 6.6 - 6.0	9.1 9.8	ns
SPI <sub>5</sub>	Serial Clock to Chip Select Inactive	1.8V domain 3.3V domain	1 * SPCK - 4.1 1 * SPCK - 4.6	—	ns
<b>Slave Mode</b>					
SPI <sub>6</sub>	SCK falling to MISO	1.8V domain 3.3V domain	3.9 3.2	17.1 15.1	ns
SPI <sub>7</sub>	MOSI Setup time before SCK rises	1.8V domain 3.3V domain	2 * MCK + 2.6 2 * MCK + 2.5	—	ns
SPI <sub>8</sub>	MOSI Hold time after SCK rises	1.8V domain 3.3V domain	1.3 1.8	—	ns
SPI <sub>9</sub>	SCK rising to MISO	1.8V domain 3.3V domain	3.9 3.3	17.2 15.8	ns
SPI <sub>10</sub>	MOSI Setup time before SCK falls	1.8V domain 3.3V domain	2 * MCK + 2.5 2 * MCK + 3.0	—	ns
SPI <sub>11</sub>	MOSI Hold time after SCK falls	1.8V domain 3.3V domain	1.4 1.3	—	ns
SPI <sub>12</sub>	NPCS0 setup to SCK rising	1.8V domain 3.3V domain	2,5 * MCK + 1.3 2,5 * MCK + 0.4	—	ns
SPI <sub>13</sub>	NPCS0 hold after SCK falling	1.8V domain 3.3V domain	1,5 * MCK + 1.7 1,5 * MCK + 1.0	—	ns
SPI <sub>14</sub>	NPCS0 setup to SCK falling	1.8V domain 3.3V domain	2,5 * MCK + 1.2 2,5 * MCK + 0.9	—	ns
SPI <sub>15</sub>	NPCS0 hold after SCK rising	1.8V domain 3.3V domain	1,5 * MCK +1.6 1,5 * MCK +1.5	—	ns

Notes: 1. 1.8V domain: VDDIO from 1.65V to 1.95V, maximum external capacitor = 20 pF

2. 3.3V domain: VDDIO from 2.85V to 3.6V, maximum external capacitor = 40 pF.

#### 46.11.7 Two-wire Serial Interface Characteristics

Table 50 describes the requirements for devices connected to the Two-wire Serial Bus. For timing symbols refer to [Figure 46-25](#).

**Table 46-54. Two-wire Serial Bus Requirements**

Symbol	Parameter	Condition	Min	Max	Units
$V_{IL}$	Input Low-voltage		-0.3	$0.3 V_{VDDIO}$	V
$V_{IH}$	Input High-voltage		$0.7 \times V_{VDDIO}$	$V_{CC} + 0.3$	V
$V_{HYS}$	Hysteresis of Schmitt Trigger Inputs		0.150	—	V
$V_{OL}$	Output Low-voltage	3 mA sink current	—	0.4	V
$t_R$	Rise Time for both TWD and TWCK		—	300	ns
$t_{OF}$	Output Fall Time from $V_{IHmin}$ to $V_{ILmax}$	$10 \text{ pF} < C_b < 400 \text{ pF}$ <a href="#">Figure 46-25</a>	$20 + 0.1C_b^{(1)(2)}$	250	ns
$C_i^{(1)}$	Capacitance for each I/O Pin		—	10	pF
$f_{TWCK}$	TWCK Clock Frequency		0	400	kHz
$R_p$	Value of Pull-up resistor	$f_{TWCK} \leq 100 \text{ kHz}$	$\frac{V_{VDDIO} - 0.4V}{3mA}$	$\frac{1000\text{ns}}{C_b}$	$\Omega$
		$f_{TWCK} > 100 \text{ kHz}$	$\frac{V_{VDDIO} - 0.4V}{3mA}$	$\frac{300\text{ns}}{C_b}$	$\Omega$
$t_{LOW}$	Low Period of the TWCK clock	$f_{TWCK} \leq 100 \text{ kHz}$	(3)	—	$\mu\text{s}$
		$f_{TWCK} > 100 \text{ kHz}$	(3)	—	$\mu\text{s}$
$t_{HIGH}$	High period of the TWCK clock	$f_{TWCK} \leq 100 \text{ kHz}$	(4)	—	$\mu\text{s}$
		$f_{TWCK} > 100 \text{ kHz}$	(4)	—	$\mu\text{s}$
$t_{HD;STA}$	Hold Time (repeated) START Condition	$f_{TWCK} \leq 100 \text{ kHz}$	$t_{HIGH}$	—	$\mu\text{s}$
		$f_{TWCK} > 100 \text{ kHz}$	$t_{HIGH}$	—	$\mu\text{s}$
$t_{SU;STA}$	Set-up time for a repeated START condition	$f_{TWCK} \leq 100 \text{ kHz}$	$t_{HIGH}$	—	$\mu\text{s}$
		$f_{TWCK} > 100 \text{ kHz}$	$t_{HIGH}$	—	$\mu\text{s}$
$t_{HD;DAT}$	Data hold time	$f_{TWCK} \leq 100 \text{ kHz}$	0	$3 \times T_{CP\_MCK}^{(5)}$	$\mu\text{s}$
		$f_{TWCK} > 100 \text{ kHz}$	0	$3 \times T_{CP\_MCK}^{(5)}$	$\mu\text{s}$
$t_{SU;DAT}$	Data setup time	$f_{TWCK} \leq 100 \text{ kHz}$	$t_{LOW\_3 \times T_{CP\_MCK}^{(5)}}$	—	ns
		$f_{TWCK} > 100 \text{ kHz}$	$t_{LOW\_3 \times T_{CP\_MCK}^{(5)}}$	—	ns
$t_{SU;STO}$	Setup time for STOP condition	$f_{TWCK} \leq 100 \text{ kHz}$	$t_{HIGH}$	—	$\mu\text{s}$
		$f_{TWCK} > 100 \text{ kHz}$	$t_{HIGH}$	—	$\mu\text{s}$
$t_{HD;STA}$	Hold Time (repeated) START Condition	$f_{TWCK} \leq 100 \text{ kHz}$	$t_{HIGH}$	—	$\mu\text{s}$
		$f_{TWCK} > 100 \text{ kHz}$	$t_{HIGH}$	—	$\mu\text{s}$

Notes: 1. Required only for  $f_{TWCK} > 100 \text{ kHz}$ .

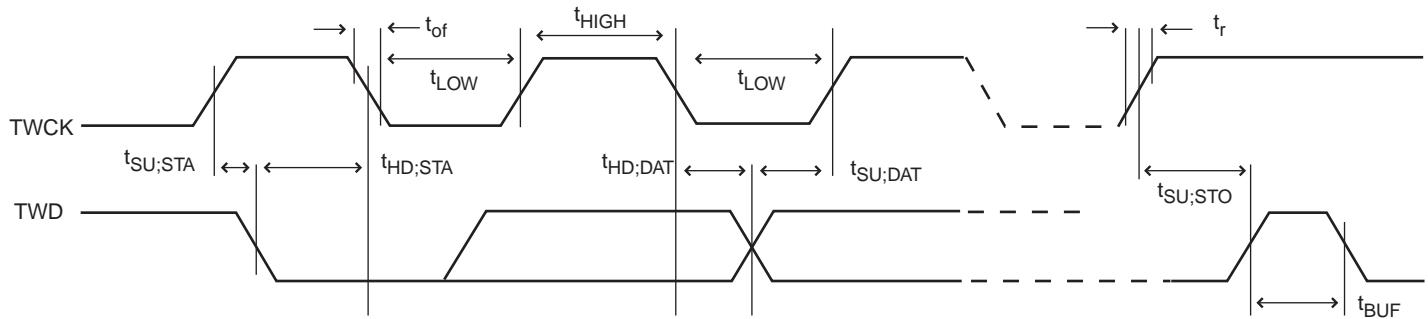
2.  $C_B$  = capacitance of one bus line in pF. Per I2C Standard,  $C_B$  Max = 400pF

3. The TWCK low Period is defined as follows:  $T_{low} = ((CLDIV \times 2^{CKDIV}) + 4) \times T_{MCK}$

4. The TWCK high period is defined as follows:  $T_{high} = ((CHDIV \times 2^{CKDIV}) + 4) \times T_{MCK}$

5.  $t_{CP\_MCK}$  = MCK Bus Period.

**Figure 46-25. Two-wire Serial Bus Timing**



#### 46.11.8 EMAC Characteristics

##### 46.11.8.1 Timing Conditions

**Table 46-55. Capacitance Load on data, clock pads**

	Corner		
	MAX	STH	MIN
3.3V	20 pf	20 pf	0 pf
1.8V	20 pf	20 pf	0 pf

These values may be product dependant and should be confirmed by the specification.

##### 46.11.8.2 Timing Constraints

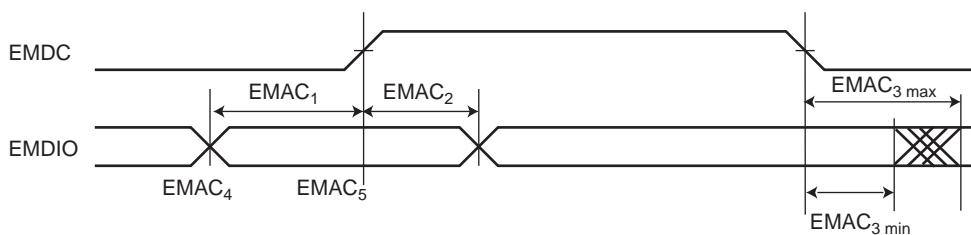
The Ethernet controller must be constrained so as to satisfy the timings of standard given in [Table 46-56](#) and [Table 46-57](#), in MAX and STH corners.

**Table 46-56. EMAC Signals Relative to GMDC**

Symbol	Parameter	Min (ns)	Max (ns)
EMAC <sub>1</sub>	Setup for GMDIO from GMDC rising	10 ns	—
EMAC <sub>2</sub>	Hold for GMDIO from GMDC rising	10 ns	—
EMAC <sub>3</sub>	GMDIO toggling from GMDC falling	0 ns <sup>(1)</sup>	10 ns <sup>(1)</sup>

Note: 1. For EMAC output signals, Min and Max access time are defined. The Min access time is the time between the EDMD falling edge and the signal change. The Max access timing is the time between the EDMD falling edge and the signal stabilizes. [Figure 46-26](#) illustrates Min and Max accesses for EMAC3.

**Figure 46-26. Min and Max Access Time of EMAC Output Signals**



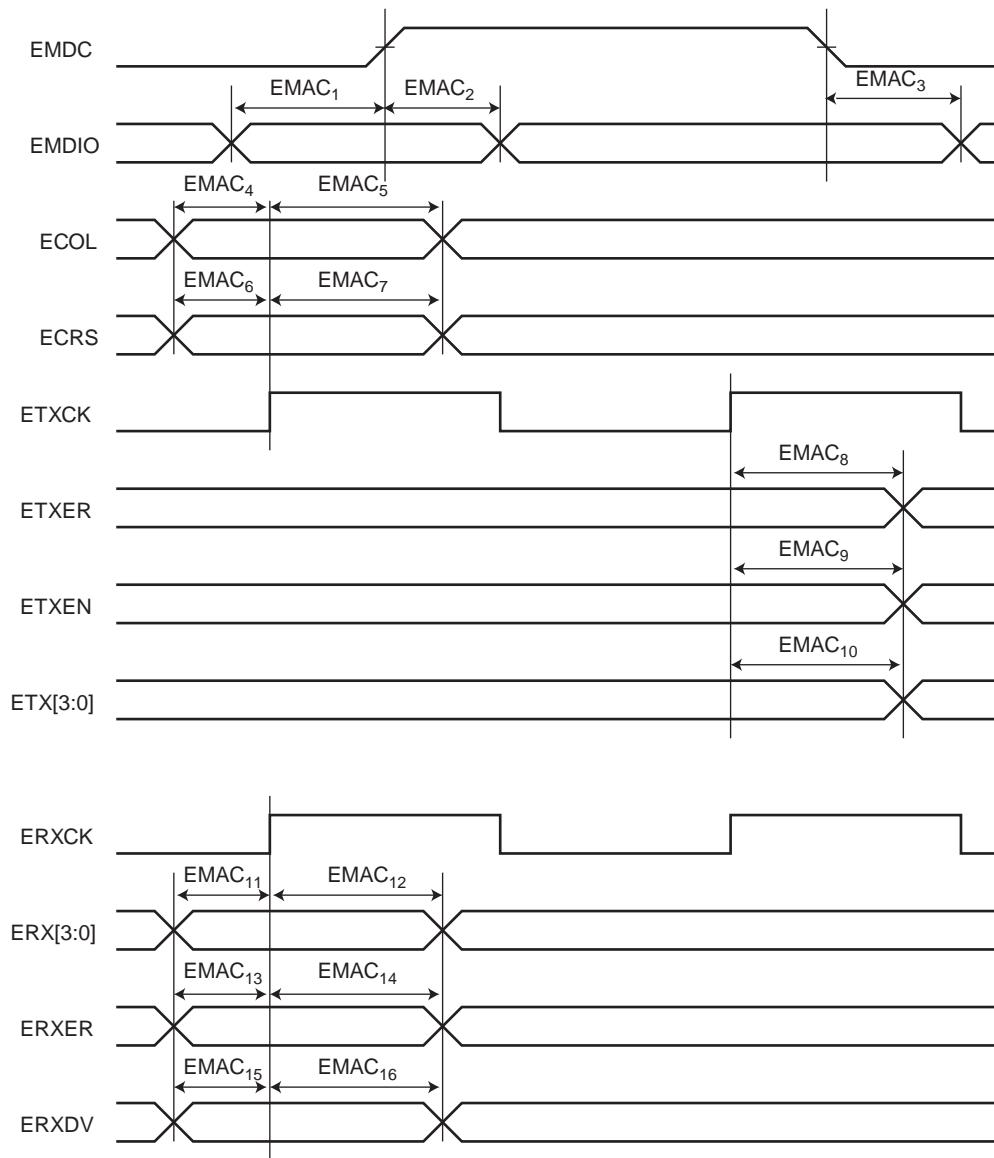
#### 46.11.8.3 MII Mode

**Table 46-57. EMAC MII Timings**

Symbol	Parameter	Min (ns)	Max (ns)
EMAC <sub>4</sub>	Setup for GCOL from GTXCK rising	10	—
EMAC <sub>5</sub>	Hold for GCOL from GTXCK rising	10	—
EMAC <sub>6</sub>	Setup for GCRS from GTXCK rising	10	—
EMAC <sub>7</sub>	Hold for GCRS from GTXCK rising	10	—
EMAC <sub>8</sub>	GTXER toggling from GTXCK rising	10 <sup>(1)</sup>	25 <sup>(1)</sup>
EMAC <sub>9</sub>	GTXEN toggling from GTXCK rising	10 <sup>(1)</sup>	25 <sup>(1)</sup>
EMAC <sub>10</sub>	GTX toggling from GTXCK rising	10 <sup>(1)</sup>	25 <sup>(1)</sup>
EMAC <sub>11</sub>	Setup for GRX from GRXCK	10	—
EMAC <sub>12</sub>	Hold for GRX from GRXCK	10	—
EMAC <sub>13</sub>	Setup for GRXER from GRXCK	10	—
EMAC <sub>14</sub>	Hold for GRXER from GRXCK	10	—
EMAC <sub>15</sub>	Setup for GRXDV from GRXCK	10	—
EMAC <sub>16</sub>	Hold for GRXDV from GRXCK	10	—

Note: 1. For EMAC output signals, Min and Max access time are defined. The Min access time is the time between the EDMC falling edge and the signal change. The Max access timing is the time between the EDMC falling edge and the signal stabilizes. [Figure 46-26](#) illustrates Min and Max accesses for EMAC3.

**Figure 46-27. EMAC MII Mode Signals**



#### 46.11.9 Embedded Flash Characteristics

The maximum operating frequency is given in [Table 46-58](#), [Table 46-59](#), [Table 46-60](#) and [Table 46-61](#), but it is limited by the Embedded Flash access time when the processor is fetching code out of it. These tables provide the device maximum operating frequency depending on the field FWS of the MC\_FMR register. This field defines the number of wait states required to access the Embedded Flash Memory.

**Table 46-58. Embedded Flash Wait State VDDCORE set at 1.08V and VDDIO 1.62V to 3.6V @105C**

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	17
1	2 cycles	34
2	3 cycles	51
3	4 cycles	69
4	5 cycles	86
5	6 cycles	100

**Table 46-59. Embedded Flash Wait State VDDCORE set at 1.08V and VDDIO 2.7V to 3.6V @105C**

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	20
1	2 cycles	41
2	3 cycles	62
3	4 cycles	83
4	5 cycles	104

**Table 46-60. Embedded Flash Wait State VDDCORE set at 1.2V and VDDIO 1.62V to 3.6V @ 105C**

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	17
1	2 cycles	35
2	3 cycles	53
3	4 cycles	71
4	5 cycles	88
5	6 cycles	106
6	7 cycles	124

**Table 46-61. Embedded Flash Wait State VDDCORE set at 1.20V and VDDIO 2.7V to 3.6V @ 105C**

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	21
1	2 cycles	43
2	3 cycles	64
3	4 cycles	86
4	5 cycles	107
5	6 cycles	129

**Table 46-62. AC Flash Characteristics**

Parameter	Conditions	Min	Typ	Max	Units
Program Cycle Time	Write page mode	—	1.5	3	ms
	Erase page mode	—	10	50	ms
	Erase block mode (by 4 Kbytes)	—	50	200	ms
	Erase sector mode	—	400	950	ms
Full Chip Erase	1 Mbytes 512 Kbytes	—	9 5.5	18 11	s
Data Retention	Not Powered or Powered	—	20	—	Years
Endurance	Write/Erase cycles per page, block or sector @ 105°C	10K	—	—	Cycles

## 47. SAM4E Mechanical Characteristics

The SAM4E series devices are available in TFBGA100, LFBGA144, LQFP100, and LQFP144 packages.

### 47.1 100-ball TFBGA Package Drawing

Figure 47-1. 100-ball TFBGA Package Drawing

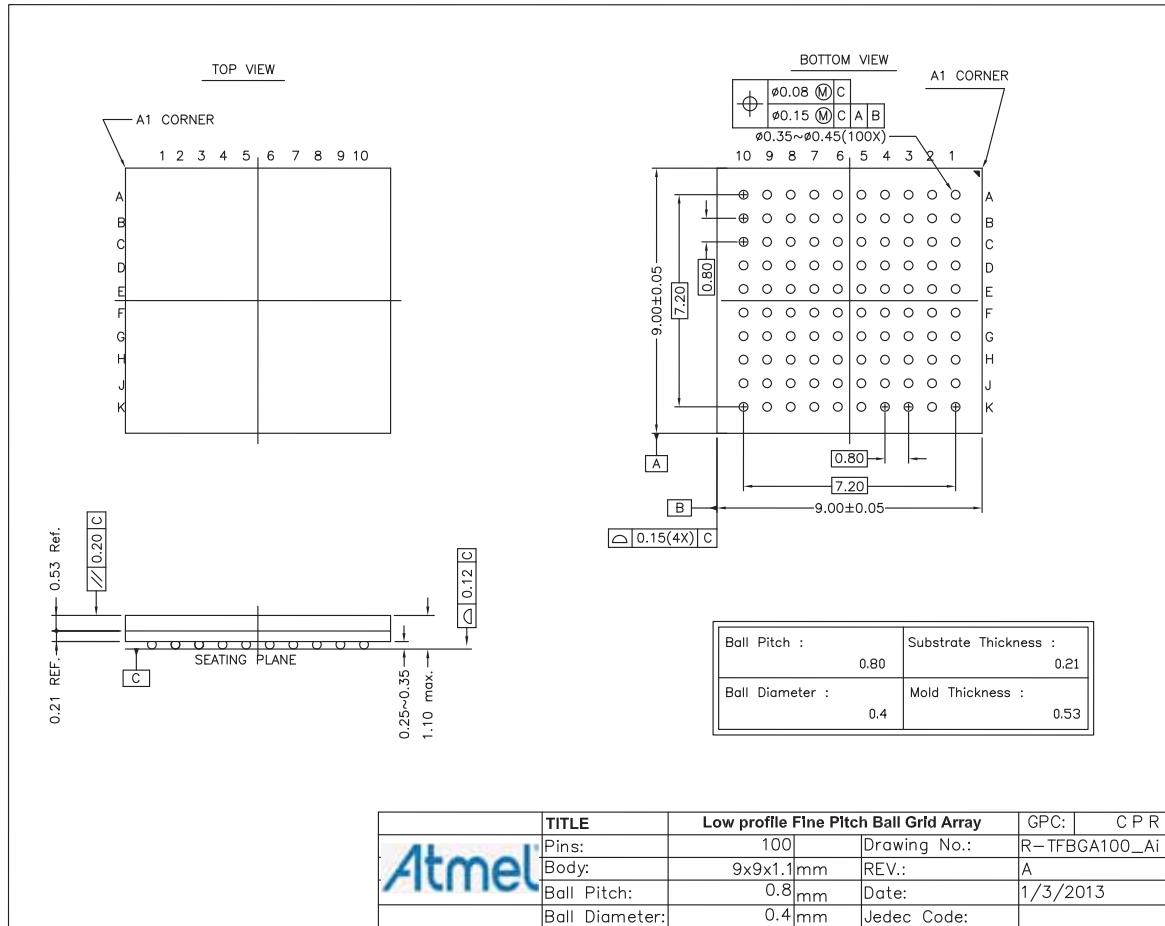


Table 47-1. Device and TFBGA Package Maximum Weight (Preliminary)

SAM4E	150	mg
-------	-----	----

Table 47-2. TFBGA Package Reference

JEDEC Drawing Reference	MO-275-DDAC-2
JESD97 Classification	e8

Table 47-3. TFBGA Package Characteristics

Moisture Sensitivity Level	3
----------------------------	---

This package respects the recommendations of the NEMI User Group.

## 47.2 144-ball LFBGA Package Drawing

Figure 47-2. 144-ball LFBGA Package Drawing

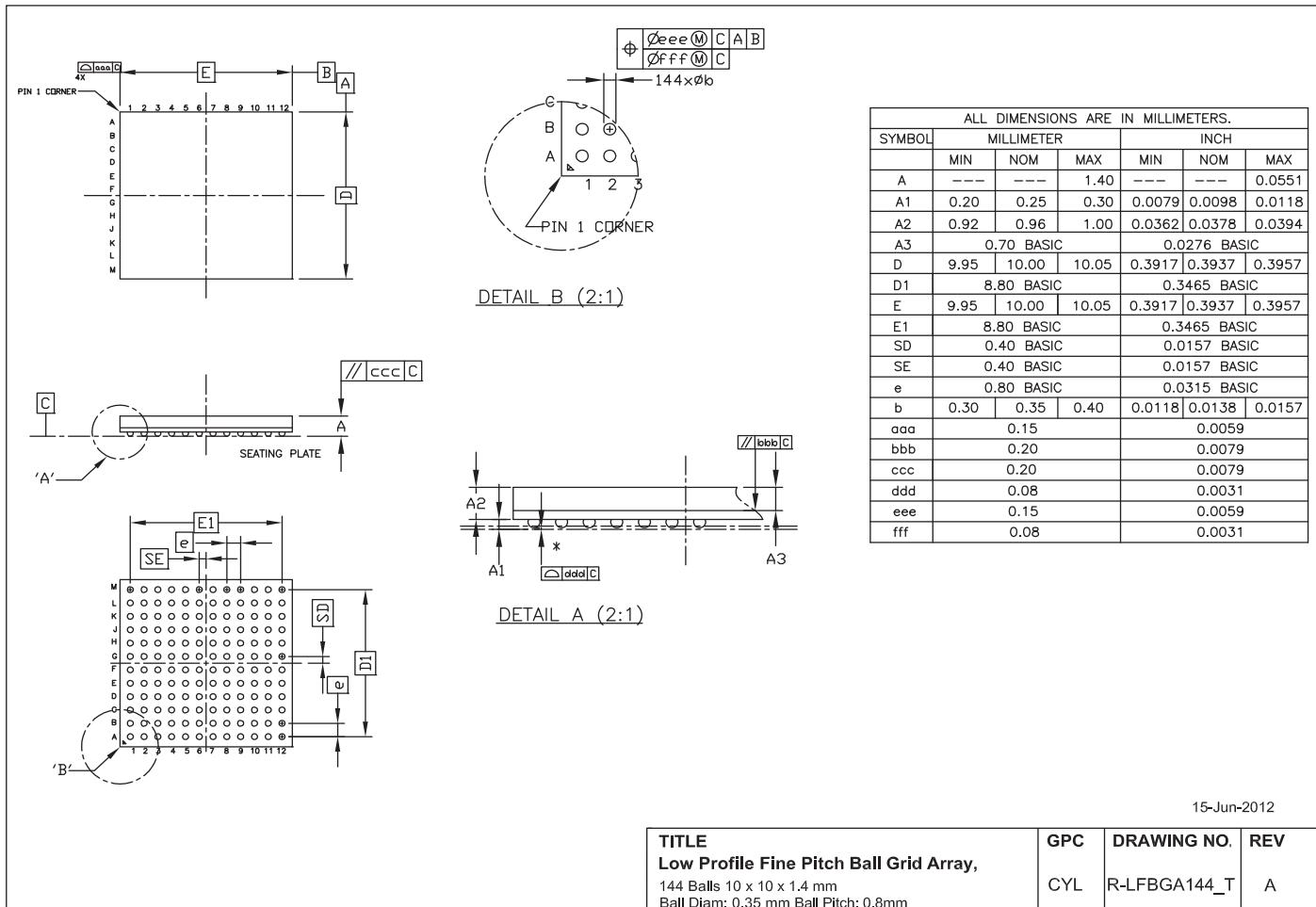


Table 47-4. Device and LFBGA Package Maximum Weight (Preliminary)

SAM4E	200	mg
-------	-----	----

Table 47-5. LFBGA Package Reference

JEDEC Drawing Reference	MS-275-EEAD-1
JESD97 Classification	e8

Table 47-6. LFBGA Package Characteristics

Moisture Sensitivity Level	3
----------------------------	---

This package respects the recommendations of the NEMI User Group.

### 47.3 100-lead LQFP Package Drawing

Figure 47-3. 100-lead LQFP Package Drawing

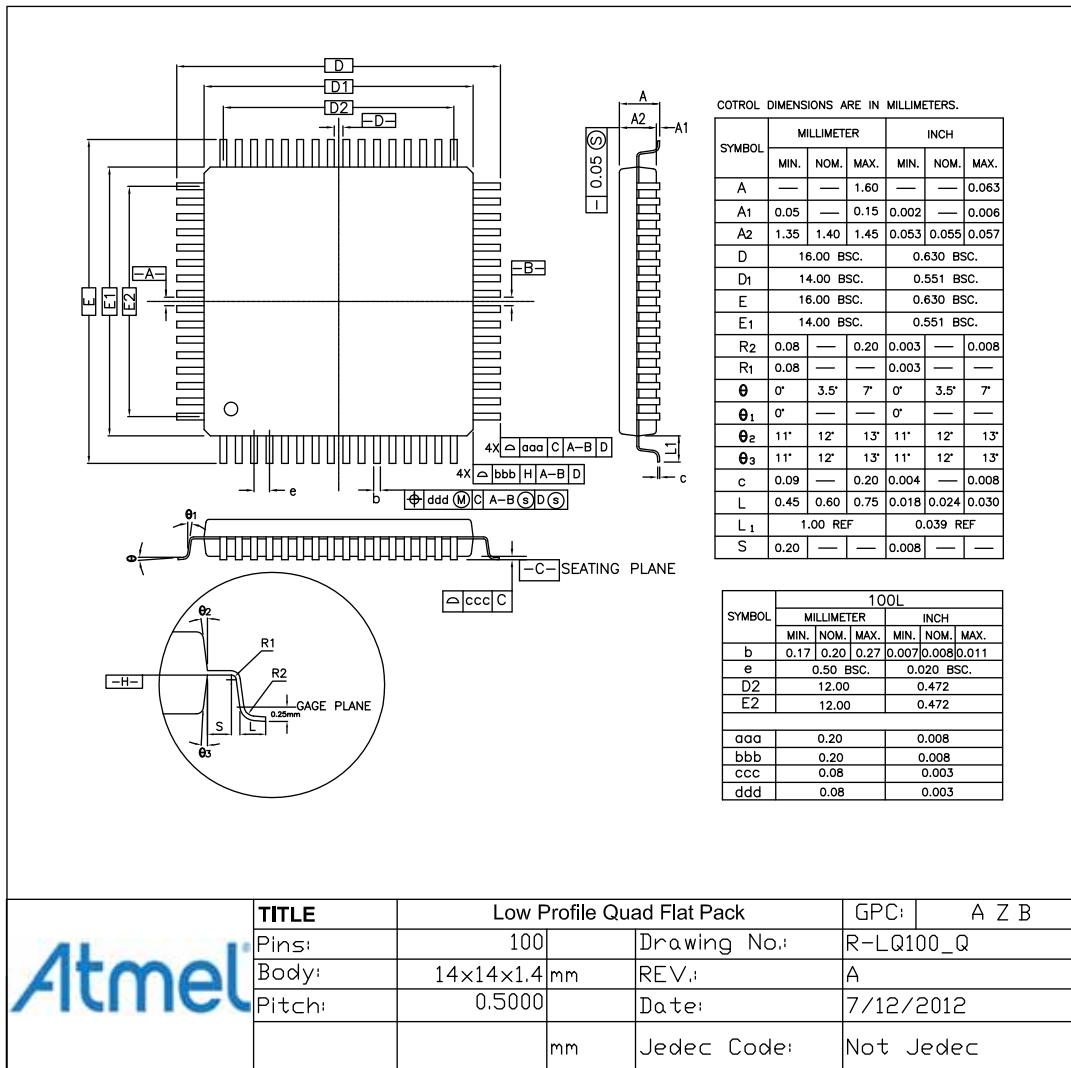


Table 47-7. Device and LQFP Package Maximum Weight (Preliminary)

SAM4E	740	mg
-------	-----	----

Table 47-8. LQFP Package Reference

JEDEC Drawing Reference	MS-026
JESD97 Classification	e3

Table 47-9. LQFP Package Characteristics

Moisture Sensitivity Level	3
----------------------------	---

This package respects the recommendations of the NEMI User Group.

## 47.4 144-lead LQFP Package Drawing

Figure 47-4. 144-lead LQFP Package Drawing

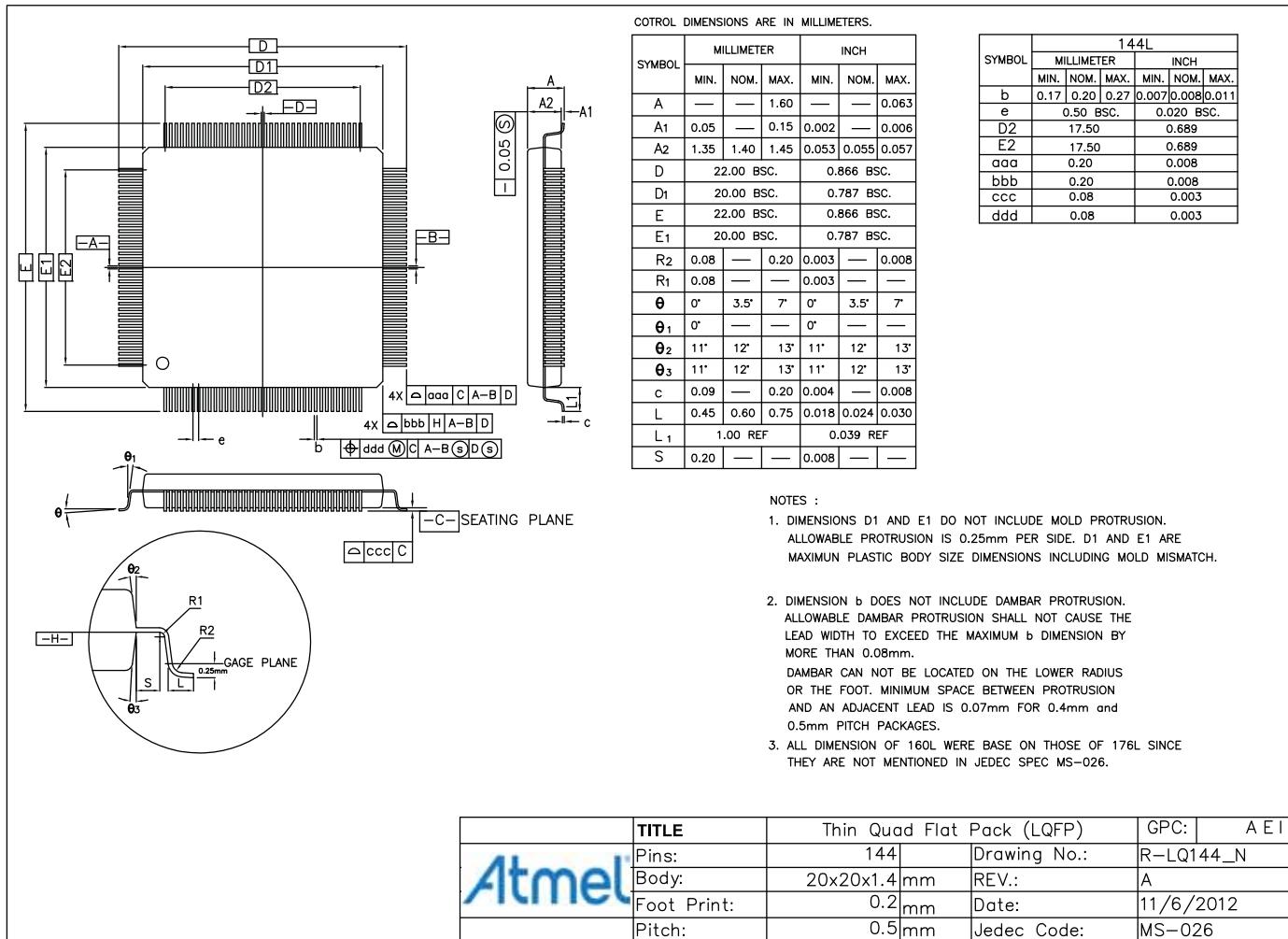


Table 47-10. Device and LQFP Package Maximum Weight (Preliminary)

SAM4E	900	mg
-------	-----	----

Table 47-11. LQFP Package Reference

JEDEC Drawing Reference	MS-026-C
JESD97 Classification	e3

Table 47-12. LQFP Package Characteristics

Moisture Sensitivity Level	3
----------------------------	---

This package respects the recommendations of the NEMI User Group.

## 47.5 Soldering Profile

Table 47-13 gives the recommended soldering profile from J-STD-020C.

**Table 47-13. Soldering Profile**

Profile Feature	Green Package
Average Ramp-up Rate (217°C to Peak)	3°C/sec. max.
Preheat Temperature 175°C ±25°C	180 sec. max.
Temperature Maintained Above 217°C	60 sec. to 150 sec.
Time within 5°C of Actual Peak Temperature	20 sec. to 40 sec.
Peak Temperature Range	260°C
Ramp-down Rate	6°C/sec. max.
Time 25°C to Peak Temperature	8 min. max.

Note: The package is certified to be backward compatible with Pb/Sn soldering profile.

A maximum of three reflow passes is allowed per component.

## 47.6 Packaging Resources

Land Pattern Definition.

Refer to the following IPC Standards:

- IPC-7351A and IPC-782 (*Generic Requirements for Surface Mount Design and Land Pattern Standards*)  
<http://landpatterns.ipc.org/default.asp>
- Atmel Green and RoHS Policy and Package Material Declaration Data Sheet <http://www.atmel.com/green/>



## 48. Ordering Information

Table 48-1. Ordering Codes for SAM4E Devices

Ordering Code	MRL	Flash (Kbytes)	RAM (Kbytes)	Package	Conditioning	Package Type	Temperature Operating Range
ATSAM4E16EA-CU	A	1024	128	LFBGA144	Tray	Green	Industrial (-40°C to 85°C)
ATSAM4E16EA-CUR	A				Reel		
ATSAM4E16EA-AU	A			LQFP144	Tray		Industrial (-40°C to 85°C)
ATSAM4E16EA-AUR	A				Reel		
ATSAM4E16EA-AN	A			LQFP144	Tray		Industrial (-40°C to 105°C)
ATSAM4E16EA-ANR	A				Reel		
ATSAM4E16CA-CU	A			TFBGA100	Tray		Industrial (-40°C to 85°C)
ATSAM4E16CA-CUR	A				Reel		
ATSAM4E16CA-AU	A			LQFP100	Tray		Industrial (-40°C to 85°C)
ATSAM4E16CA-AUR	A				Reel		
ATSAM4E16CA-AN	A			LQFP100	Tray		Industrial (-40°C to 105°C)
ATSAM4E16CA-ANR	A				Reel		
ATSAM4E8EA-CU	A	512	512	LFBGA144	Tray	Green	Industrial (-40°C to 85°C)
ATSAM4E8EA-CUR	A				Reel		
ATSAM4E8EA-AU	A			LQFP144	Tray		Industrial (-40°C to 85°C)
ATSAM4E8EA-AUR	A				Reel		
ATSAM4E8EA-AN	A			LQFP144	Tray		Industrial (-40°C to 105°C)
ATSAM4E8EA-ANR	A				Reel		
ATSAM4E8CA-CU	A			TFBGA100	Tray		Industrial (-40°C to 85°C)
ATSAM4E8CA-CUR	A				Reel		
ATSAM4E8CA-AU	A			LQFP100	Tray		Industrial (-40°C to 85°C)
ATSAM4E8CA-AUR	A				Reel		
ATSAM4E8CA-AN	A			LQFP100	Tray		Industrial (-40°C to 105°C)
ATSAM4E8CA-ANR	A				Reel		

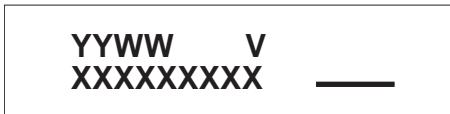


## 49. Errata on SAM4E Devices

### 49.1 Marking

All devices are marked with the Atmel logo and the ordering code.

Additional marking is as follows:



where

- "YY": manufactory year
- "WW": manufactory week
- "V": revision
- "XXXXXXXX": lot number

### 49.2 Errata

#### 49.2.1 Watchdog

##### 49.2.1.1 Watchdog Not Stopped in Wait Mode

When the Watchdog is enabled and the bit WAITMODE = 1 is used to enter wait mode, the watchdog is not halted. If the time spent in Wait Mode is longer than the Watchdog time-out, the device will be reset if Watchdog reset is enabled.

##### Problem Fix/Workaround

When entering wait mode, the Wait For Event (WFE) instruction of the processor Cortex-M4 must be used with the SLEEPDEEP of the System Control Register (SCB\_SCR) of the Cortex-M = 0.

#### 49.2.2 Brownout Detector

##### 49.2.2.1 Unpredictable Behavior if BOD is Disabled, VDDCORE is Lost and VDDIO is Connected

In active mode or in wait mode, if the Brownout Detector is disabled (SUPC\_MR: BODDIS=1) and power is lost on VDDCORE while VDDIO is powered, the device might not be properly reset and may behave unpredictably.

##### Problem Fix/Workaround

When the Brownout Detector is disabled in active or in wait mode, VDDCORE always needs to be powered.



## Revision History

In the tables that follow, the most recent version of the document appears first.

“rfo” indicates changes requested during the document review and approval loop.

Doc. Rev. 11157C	Comments	Change Request Ref.
	<p><b>Introduction</b></p> <p>In “Features”:</p> <ul style="list-style-type: none"> <li>- added information on Two-wire Interface in Peripherals section and Wake-on-LAN for EMAC</li> <li>- changed operating temperature range to 105°C</li> </ul> <p>Updated Table 1-1 “Configuration Summary” with TWI information</p> <p>In Section 4. “Package and Pinout”, added the FFPI signals to Table 4-1 “SAM4E 100-ball TFBGA Pinout”, Table 4-2 “SAM4E 144-ball LFBGA Pinout”, Table 4-3 “SAM4E 100-lead LQFP Pinout” and Table 4-4 “SAM4E 144-lead LQFP Pinout”.</p> <p>Updated Section 5.5 “Low-power Modes”. Added information on WFE.</p> <p>In Section 6.1 “General-purpose I/O Lines” on page 21, added information on GPIOs as analog input.</p> <p>Removed Section 7. “Processor and Architecture”. Removed Sections 11-4 to 11-16. Reordered introduction sections.</p> <p>Removed Note regarding PIOs and 144-pin package at bottom of Table 11-5, “Multiplexing on PIO Controller D (PIOD),” on page 40 and Table 11-6, “Multiplexing on PIO Controller E (PIOE),” on page 41.</p>	rfo rfo rfo 9045 9073 8992 rfo 9018
	<p><b>RSTC:</b></p> <p>In Section 15.3.6 “Reset Controller Status Register”, RSTTYP information corrected.</p>	8847
	<p><b>RTT:</b></p> <p>Added notes in Section 16.4 “Functional Description”, Section 16.5.1 “Real-time Timer Mode Register” and in Section 16.5.2 “Real-time Timer Alarm Register”.</p>	9026
	<p><b>RTC:</b></p> <p>In Section 18.5.3 “Alarm”, added new information and note.</p> <p>In Section 18.5.7 “RTC Accurate Clock Calibration”, updated information on temperature range.</p> <p>In Section 18.6.5 “RTC Time Alarm Register” and Section 18.6.6 “RTC Calendar Alarm Register”, added notes.</p>	8900, 9027 9033 9027
	<p><b>WDT:</b></p> <p>In Section 19.1 “Description”, added information on slow clock at 32 kHz.</p> <p>In Section 19.2 “Embedded Characteristics”, added that Watchdog Clock is independent from Processor Clock.</p> <p>Moved note (WDD, WDV) from Section 19.5.3 “Watchdog Timer Status Register” to Section 19.5.2 “Watchdog Timer Mode Register”.</p>	8429 rfo
	<p><b>EFC:</b></p> <p>In Section 22.4.3.5 “Lock Bit Protection”, added notes on FARG exceeding limits. Updated existing note in Section 22.4.3.6 “GPNVM Bit”.</p> <p>Added Section 22.4.3.3 “Optimized Partial Programming”. Added note on programming limitations in Section 22.4.3.2 “Write Commands”.</p>	8854 8985
	<p><b>FFPI:</b></p> <p>Removed information throughout on Serial Fast Flash Programming not available for device.</p>	9007

Doc. Rev. 11157C	Comments	Change Request Ref.
	<b>CMCC:</b> In <a href="#">Table 24.5 “Cortex M Cache Controller (CMCC) User Interface”</a> , updated reset value of CMCC_SR.	8903
	<b>PMC:</b> <a href="#">Section 30.1.4.2 “Slow Clock Crystal Oscillator”</a> , replaced “...in MOSCSEL bit of CKGR_MOR,...” with “...in XTALSEL bit of SUPC_CR,...” in the last phrase of the 3d paragraph. <a href="#">Section 30.1.4.2 “Slow Clock Crystal Oscillator”</a> , added references on the OSCSEL bit of PMC_SR in the 3d paragraph. Register names in Clock Generator: Replaced “PLL_MCKR” with “PMC_MCKR” and “PLL_SR” with “PMC_SR” in <a href="#">Section 30.1.5.5 “Software Sequence to Detect the Presence of Fast Crystal”</a> . In <a href="#">Section 30.1.6.1 “Divider and Phase Lock Loop Programming”</a> , 3rd bullet, replaced PMC_IER with PMC_SR. Deleted previous 4th bullet (was useless sentence “Disable and then enable the PLL...”). In <a href="#">Figure 30-3 “Main Clock Block Diagram”</a> and <a href="#">Section 30.1.5.3 “3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator”</a> paragraph 5, replaced MOSCXTCNT with MOSCXTST. Added code example in step 1. of <a href="#">Section 30.2.13 “Programming Sequence”</a> . Corrected reset value of CKGR_MOR register in <a href="#">Table 30-2, “Register Mapping”</a> .	9069 rfo 8970 8963 8447
	<b>USART:</b> In <a href="#">Table 37-10 “Maximum Timeguard Length Depending on Baud Rate”</a> and in <a href="#">Table 37-11 “Maximum Timeout Period”</a> , modified 33400 baudrate to 38400. Updated <a href="#">Figure 37-22 “Parity Error”</a> with corrected stop bit value.	8943
	<b>TC:</b> In <a href="#">Section 38.1 “Description”</a> , corrected reference to TIOA1 with TIOB1. In <a href="#">Section 38.7.3 “TC Channel Mode Register: Waveform Mode”</a> , added note for ENETRG description.	8885
	<b>HSMCI:</b> In <a href="#">Figure 39-8 “Read Functional Flow Diagram”</a> , <a href="#">Figure 39-9 “Write Functional Flow Diagram”</a> and <a href="#">Figure 39-10 “Multiple Write Functional Flow Diagram”</a> , corrected HSMCI_MR to HSMCI_BLKR when referring to Block Length field that is not available in HSMCI_MR. Removed related Note 2. In <a href="#">Section 39.14.7 “HSMCI Block Register”</a> , BLKLEN bit description, removed reference on accessibility in Mode Register.	9012 rfo



Doc. Rev. 11157B	Comments	Change Request Ref.
	<p><b>Introduction:</b>            Updated the section structure and added references to 100-ball TFBGA and 100-lead LQFP packages in:            - <a href="#">Section 1. "Features"</a>            - <a href="#">Table 1-1 "Configuration Summary"</a>            - <a href="#">Figure 2-1 "SAM4E 100-pin Block Diagram"</a>            - <a href="#">Section 4.1 "100-ball TFBGA Package and Pinout"</a>            - <a href="#">Section 4.3 "100-lead LQFP Package and Pinout"</a>            - <a href="#">Table 11-1 "PIO available according to Pin Count"</a>              Added Analog Comparator (ACC) and Reinforced Safety Watchdog Timer (RSWDT) blocks in <a href="#">Figure 2-2 "SAM4E 144-pin Block Diagram"</a>.              Updated the description of power supply pins in <a href="#">Section 5.1 "Power Supplies"</a>.            Updated <a href="#">Figure 11-3 "Power Management Controller Block Diagram"</a>.            Removed RC80M references in <a href="#">Figure 10-1 "System Controller Block Diagram"</a> and <a href="#">Figure 11-2 "Clock Generator Block Diagram"</a>.              Add data on consumption and wake-up time in <a href="#">Table 5-1 "Low-power Mode Configuration Summary"</a>.            Removed "AT91SAM" from the document title and further on in the entire document (where appropriate).            Replaced "Cortex™" references with "Cortex®" in <a href="#">"Description"</a> and further on in the entire document.  <a href="#">Section 11.14 "Chip Identification"</a>, replaced "Table 11-1. SAM4E Chip ID Register" with a cross-reference to the corresponding <a href="#">Table 14-1 "SAM4E Chip ID Registers"</a> (<a href="#">Section 14. "Chip Identifier (CHIPID)"</a>).            Removed package dimension references in <a href="#">Section 4. "Package and Pinout"</a>.            Added a phrase on the flash write commands usage in <a href="#">Section 8.1.3.1 "Flash Overview"</a> (the last paragraph).            Updated package information in <a href="#">Section 11.2 "Peripheral Signal Multiplexing on I/O Lines"</a>:            - replaced "100/144 pin version" with "144 pin version" in <a href="#">Table 11-4 "Multiplexing on PIO Controller C (PIOC)"</a>            - removed "144 pin version" in <a href="#">Table 11-5 "Multiplexing on PIO Controller D (PIOD)"</a>              Updated <a href="#">Figure 7-1 "SAM4E Product Mapping"</a>.              Replaced GRX by GRX1 on line PD6 in <a href="#">Table 11-5 "Multiplexing on PIO Controller D (PIOD)"</a>.</p>	8580  8605/rfo  8644  rfo  8724  rfo  rfo  rfo  rfo  rfo  rfo  rfo  8825  8897
	<p><b>CHIPID:</b>  <a href="#">Section 14.3 "Chip Identifier (CHIPID) User Interface"</a>, updated the ARCH bitfield table in <a href="#">"ARCH: Architecture Identifier"</a> (removed rows with not relevant information: 0x43, 0x88, 0x89, 0x8A, 0x93, 0x94, and 0x95).  <a href="#">Section 14.2 "Embedded Characteristics"</a>, replaced '0x0011_0201' with '0x0012_0201' and '0x0011_0209' with '0x0012_0209' in <a href="#">Table 14-1 "SAM4E Chip ID Registers"</a>.  <a href="#">Section 14.3.2 "Chip ID Extension Register"</a>, updated value in the Flash Size table and removed package references in the Product Number table.</p>	8603  8851  
	<p><b>RTT:</b>  <a href="#">Section 16.3 "Block Diagram"</a>, replaced 'CLKSRC' source reference with 'RTC1HZ' in <a href="#">Figure 16-1 "Real-time Timer"</a>.            Updated the 4th and the 8th paragraphs in <a href="#">Section 16.4 "Functional Description"</a> ("Setting the RTC 1 Hz clock to 1..." and "The RTTINC bit in RTT_SR is set..." respectively).  <a href="#">Section 16.5.1 "Real-time Timer Mode Register"</a>, added notes in <a href="#">"RTTDIS: Real-time Timer Disable"</a> and <a href="#">"RTC1HZ: Real-Time Clock 1Hz Clock Selection"</a>.</p>	8574  8621
	<p><b>RSWDT:</b>            Added a new component: <a href="#">Section 17. "Reinforced Safety Watchdog Timer (RSWDT)"</a>.</p>	rfo

Doc. Rev. 11157B	Comments	Change Request Ref.
	<p><b>RTC:</b>  <a href="#">Section 18.2 "Embedded Characteristics"</a>, added a new bullet "Safety/Security features", right indented the 2 following bullets.  <a href="#">Section 18.5 "Functional Description"</a>, added the last paragraph ("The RTC can generate...").</p>	8544 8563
	<p><b>SUPC:</b>  Updated <a href="#">Figure 20-1 "Supply Controller Block Diagram"</a>.  Updated the 2-nd paragraph in <a href="#">Section 20.4.7.4 "Low-power Tamper Detection Inputs"</a> and placed this section just after <a href="#">Section 20.4.7.3 "Low-power Debouncer Inputs"</a>.</p>	8799 8577/8578
	<p><b>EFC:</b>  Typo fixed in <a href="#">Section 22.4.3.5 "GPNVM Bit"</a> and added title in <a href="#">Section 22.4.3.6 "Calibration Bit"</a>.  Added notes when FARG exceeds limits in <a href="#">Section 22.4.3.4 "Lock Bit Protection"</a> and reworked the existing note in <a href="#">Section 22.4.3.5 "GPNVM Bit"</a>.</p>	8854
	<p><b>FFPI:</b>  Removed duplicate and erroneous figures in:  <ul style="list-style-type: none"> <li>- <a href="#">Section 23.3.1 "Device Configuration"</a></li> <li>- <a href="#">Section 23.3.4.1 "Write Handshaking"</a></li> <li>- <a href="#">Section 23.3.4.2 "Read Handshaking"</a></li> <li>- <a href="#">Section 23.3.5.8 "Get Version Command"</a></li> </ul> Fixed the section structure. </p>	rfo  rfo
	<p><b>MATRIX:</b>  Removed references to Special Function Registers (SFR) and to Bus Matrix Priority Registers B for Slaves.  Updated sections:  <ul style="list-style-type: none"> <li>- <a href="#">Section 26.1 "Description"</a></li> <li>- <a href="#">Section 26.2 "Embedded Characteristics"</a></li> <li>- <a href="#">Section 26.12 "Bus Matrix (MATRIX) User Interface"</a>: <ul style="list-style-type: none"> <li>- <a href="#">Table 26-3 "Register Mapping"</a></li> <li>- <a href="#">Section 26.12.1 - Section 26.12.4</a></li> <li>- <a href="#">Section 26.12.7</a></li> </ul> </li> </ul> Updated register names to "MATRIX_MCFGx [x=0..6]" and so on in <a href="#">Section 26.12.1 "Bus Matrix Master Configuration Registers"</a>, <a href="#">Section 26.12.2 "Bus Matrix Slave Configuration Registers"</a>, and <a href="#">Section 26.12.3 "Bus Matrix Priority Registers A For Slaves"</a>.  Replaced the WPKEY bitfield description with the corresponding table in <a href="#">Section 26.12.7 "Write Protect Mode Register"</a>. </p>	rfo  rfo  rfo
	<p><b>PMC:</b>  <a href="#">Section 30.2.16.9 "PMC Clock Generator PLLA Register"</a>, removed "x8" in "PLLACOUNT: PLLA Counter" bitfield description.  <a href="#">Section 30.2.16.10 "PMC Write Protect Mode Register"</a>, replaced the WPKEY bitfield description with a table.  Updated the last paragraph in <a href="#">Section 30.1.5.2 "Fast RC Oscillator Clock Frequency Adjustment"</a> and added the corresponding note in <a href="#">Section 30.2.16.8 "PMC Clock Generator Main Clock Frequency Register"</a>.</p>	8564  8837  8838  8853

Doc. Rev. 11157B	Comments	Change Request Ref.
	<b>AES:</b> Updated <a href="#">Section 31.4.4 “DMA Mode”</a> and <a href="#">Section 31.4.7 “DMA Mode”</a> (“PDC Mode” --> “DMA Mode”). <a href="#">Section 31.6.2 “AES Mode Register”</a> , replaced the CKEY bitfield description with a table.	rfo 8859
	<b>PIO:</b> <a href="#">Section 33.5 “Functional Description”</a> , added pull-down resistor and registers in <a href="#">Figure 33-5 “Input Glitch Filter Timing”</a> . <a href="#">Section 33.7.46 “PIO Write Protect Mode Register”</a> , replaced the WPKEY bitfield description with a table. Added missing dashes for reserved registers in <a href="#">Table 33-3 “Register Mapping”</a> . Replaced “DIVx” with “DIV” in <a href="#">Section 33.7.29 “PIO Slow Clock Divider Debouncing Register”</a> . Updated the SCHMITTx bitfield description in <a href="#">Section 33.7.48 “PIO Schmitt Trigger Register”</a> and updated the Delayx bitfield description in <a href="#">Section 33.7.49 “PIO I/O Delay Register”</a> .	8522
	<b>SPI:</b> <a href="#">Section 34.7.3.2 “Master Mode Flow Diagram”</a> , added TDRE references in <a href="#">Figure 34-8 “PDC Status Register Flags Behavior”</a> . <a href="#">Section 34.7.4 “SPI Slave Mode”</a> , updated the next-to-last paragraph (“Then, a new data is loaded...”). Replaced offset 0x4C with 0x40, 0xE8 with 0xEC in <a href="#">Section 34.8 “Serial Peripheral Interface (SPI) User Interface”</a> .	8798 8792 8840
	<b>USART:</b> Added a paragraph on IRDA_FILTER programming criteria in <a href="#">Section 37.7.5.3 “IrDA Demodulator”</a> and in the corresponding bitfield description in <a href="#">Section 37.8.20 “USART IrDA FILTER Register”</a> . <a href="#">Section 37.8.18 “USART FI DI RATIO Register”</a> , expanded FI_DI_RATIO bitfield to 16 bits in the register table. Added RXBUFF and TXBUFE bitfields and their descriptions in: <ul style="list-style-type: none"> <li>- <a href="#">Section 37.8.6 “USART Interrupt Enable Register (SPI_MODE)”</a></li> <li>- <a href="#">Section 37.8.8 “USART Interrupt Disable Register (SPI_MODE)”</a></li> <li>- <a href="#">Section 37.8.10 “USART Interrupt Mask Register (SPI_MODE)”</a></li> <li>- <a href="#">Section 37.8.12 “USART Channel Status Register (SPI_MODE)”</a></li> </ul>	8508 8643 8704
	<b>TC:</b> Fixed a typo in <a href="#">Section 38.1 “Description”</a> : “TIOA1” --> “TIOB1”.	8885
	<b>ACC:</b> Added <a href="#">Table 42-2 “Analog Comparator Inputs”</a> .	rfo

Doc. Rev. 11157B	Comments	Change Request Ref.
	<p><b>AFEC:</b></p> <p>Fixed a typo in <a href="#">Section 43.7.25 “AFEC Write Protect Mode Register”</a>: replaced ‘(“AFE” in ASCII)’ with ‘(“ADC” in ASCII)’ in the WPEN and WPKEY bitfield descriptions.</p> <p>Updated register tables (replaced bitfield data with “-” for bits from 16 to 23) in:</p> <ul style="list-style-type: none"> <li>- <a href="#">Section 43.7.6 “AFEC Channel Enable Register”</a></li> <li>- <a href="#">Section 43.7.7 “AFEC Channel Disable Register”</a></li> <li>- <a href="#">Section 43.7.8 “AFEC Channel Status Register”</a></li> <li>- <a href="#">Section 43.7.17 “AFEC Channel Calibration DC Offset Register”</a></li> </ul> <p>Updated the acronym from ‘AFE’ to ‘AFEC’ in the entire document (except of ‘AFE Controller’).</p> <p>Updated <a href="#">Section 43.2 “Embedded Characteristics”</a>.</p> <p>Reworked Section <a href="#">Section 43.6.9 “Input Gain and Offset”</a>:</p> <ul style="list-style-type: none"> <li>- updated the first and the last paragraphs</li> <li>- removed Table 42-7 Offset of the Sample and Hold Unit: OFFSET DIFF and Gain (G)</li> <li>- updated <a href="#">Figure 43-7 “Analog Full Scale Ranges in Single Ended/Differential Applications Versus Gain”</a>.</li> </ul> <p>Rewritten <a href="#">Section 43.6.13 “Automatic Calibration”</a>.</p> <p>Updated <a href="#">Section 43.7 “Analog-Front-End Controller (AFEC) User Interface”</a>:</p> <ul style="list-style-type: none"> <li>- AFEC_CSELR, AFEC_COOR are declared as Read-write registers</li> <li>- “Channel DC Offset Register” --&gt; ‘Channel Calibration DC Offset Register’</li> <li>- updated “ANACH: Analog Change” bitfield description table in <a href="#">Section 43.7.2 “AFEC Mode Register”</a></li> <li>- updated the OFFx bitfield description and added a note in <a href="#">Section 43.7.17 “AFEC Channel Calibration DC Offset Register”</a>.</li> </ul> <p>Updated the last paragraph in <a href="#">Section 43.6.3.1 “Enhanced Resolution Mode”</a>.</p>	8568 rfo 8806 rfo
	<p><b>GMAC:</b></p> <p>Replaced “at all three speeds” by “at all supported speeds” in <a href="#">Section 44.1 “Description”</a> and <a href="#">Section 44.2 “Embedded Characteristics”</a>.</p> <p><a href="#">Section 44.7.1 “Network Control Register”</a>, removed the LB bitfield and its description and updated the LBL bitfield description (removed phrases: “Bit 11 of GMAC_R ... loopback mode.” and “Local loopback functionality is optional.”).</p> <p><a href="#">Section 44.7.4 “User Register”</a>, removed the BPDG, HDFC and RMII bitfields and their descriptions.</p> <p>Removed references to external FIFO/external FIFO interface in the entire document.</p>	8424 8427

Doc. Rev.	Comments		Change	Request	Ref.
11157B					
	<b>Electrical Characteristics:</b>				
	Added references to 100-ball TFBGA and 100-lead LQFP packages in <a href="#">Table 46-1 “Absolute Maximum Ratings”</a> .			8580	
	Updated data in:			rfo	
	<ul style="list-style-type: none"> <li>- <a href="#">Table 46-2 “DC Characteristics”</a></li> <li>- <a href="#">Table 46-3 “1.2V Voltage Regulator Characteristics”</a></li> <li>- <a href="#">Section 46.3.1.2 “Configuration B: 32768 kHz Crystal Oscillator Enabled”</a></li> <li>- <a href="#">Section 46.3.2.1 “Sleep Mode”</a></li> <li>- <a href="#">Section 46.3.2.2 “Wait Mode”, including <a href="#">Table 46-12 “Typical Current Consumption in Wait Mode”</a></a></li> <li>- <a href="#">Table 46-13 “Active Power Consumption with VDDCORE @ 1.2V Running from Embedded Memory (IDDCORE-AMP1)”</a></li> <li>- <a href="#">Table 46-15 “Power Consumption on VDDCORE<sup>(1)</sup>”</a></li> <li>- <a href="#">Table 46-16 “32 kHz RC Oscillator Characteristics”</a></li> <li>- <a href="#">Table 46-27 “Analog Power Supply Characteristics”</a></li> <li>- <a href="#">Table 46-28 “Channel Conversion Time and ADC Clock”</a></li> <li>- <a href="#">Table 46-29 “External Voltage Reference Input”</a></li> <li>- <a href="#">Section 46.7.1 “ADC Resolution”</a></li> <li>- <a href="#">Section 46.7.2 “Static Performance Characteristics”</a></li> <li>- <a href="#">Section 46.7.3 “Dynamic Performance Characteristics”</a></li> <li>- Notes in <a href="#">Table 46-47 “I/O Characteristics”</a></li> </ul>				
	Added:				
	<ul style="list-style-type: none"> <li>- <a href="#">Figure 46-6 “Current Consumption in Sleep Mode (AMP1) versus Master Clock Ranges (Condition from Table 46-10)”</a></li> <li>- <a href="#">Figure 46-9 “Active Power Consumption with VDDCORE @ 1.2V”</a></li> <li>- <a href="#">Section 46.3.3.2 “SAM4E Active Total Power Consumption”</a></li> <li>- <a href="#">Figure 46-14 “12-bit AFE (Analog Front End) Diagram”</a></li> </ul>			rfo	
	Updated temperature range from “-40°C - +125°C” to “-40°C - +85°C” in <a href="#">Table 46-16 “32 kHz RC Oscillator Characteristics”</a> .			rfo	
	Added missing titles in:			rfo	
	<ul style="list-style-type: none"> <li>- <a href="#">Figure 46-11 “32.768 kHz Crystal Oscillator Schematics”</a></li> <li>- <a href="#">Figure 46-12 “3 to 20 MHz Crystal Oscillator Schematics”</a></li> </ul>				

Doc. Rev. 11157A	Comments	Change Request Ref.
	Initial release	



## Table of Contents

---

	.....	Description
1.	Features .....	2
1.1	Configuration Summary .....	4
2.	Block Diagram .....	5
3.	Signal Description .....	7
4.	Package and Pinout .....	11
4.1	100-ball TFBGA Package and Pinout .....	11
4.2	144-ball LFBGA Package and Pinout .....	12
4.3	100-lead LQFP Package and Pinout .....	13
4.4	144-lead LQFP Package and Pinout .....	14
5.	Power Considerations .....	15
5.1	Power Supplies .....	15
5.2	Voltage Regulator .....	15
5.3	Typical Powering Schematics .....	15
5.4	Low-power Modes .....	16
5.5	Wake-up Sources .....	19
5.6	Fast Start-up .....	20
6.	Input/Output Lines .....	21
6.1	General-purpose I/O Lines .....	21
6.2	System I/O Lines .....	22
7.	Product Mapping .....	23
8.	Memories .....	24
8.1	Embedded Memories .....	24
8.2	External Memories .....	28
8.3	Cortex-M Cache Controller (CMCC) .....	28
9.	Real-time Event Management .....	29
9.1	Embedded Characteristics .....	29
9.2	Real-time Event Mapping List .....	30
10.	System Controller .....	31
10.1	System Controller and Peripherals Mapping .....	32
10.2	Power-on-Reset, Brownout and Supply Monitor .....	32
10.3	Reset Controller .....	32
11.	Peripherals .....	34
11.1	Peripheral Identifiers .....	34
11.2	Peripheral Signal Multiplexing on I/O Lines .....	36
12.	ARM Cortex-M4 .....	43
12.1	Description .....	43

12.2	Embedded Characteristics . . . . .	44
12.3	Block Diagram . . . . .	45
12.4	Cortex-M4 Models . . . . .	46
12.5	Power Management . . . . .	75
12.6	Cortex-M4 Instruction Set . . . . .	77
12.7	Cortex-M4 Core Peripherals . . . . .	188
12.8	Nested Vectored Interrupt Controller (NVIC) . . . . .	189
12.9	System Control Block (SCB) . . . . .	199
12.10	System Timer (SysTick) . . . . .	225
12.11	Memory Protection Unit (MPU) . . . . .	230
12.12	Floating Point Unit (FPU) . . . . .	243
12.13	Glossary . . . . .	252
<b>13.</b>	<b>Debug and Test Features . . . . .</b>	<b>257</b>
13.1	Description . . . . .	257
13.2	Embedded Characteristics . . . . .	257
13.3	Debug and Test Block Diagram . . . . .	258
13.4	Application Examples . . . . .	259
13.5	Debug and Test Pin Description . . . . .	260
13.6	Functional Description . . . . .	261
<b>14.</b>	<b>Chip Identifier (CHIPID) . . . . .</b>	<b>267</b>
14.1	Description . . . . .	267
14.2	Embedded Characteristics . . . . .	267
14.3	Power Management Controller (PMC) User Interface . . . . .	268
<b>15.</b>	<b>Reset Controller (RSTC) . . . . .</b>	<b>275</b>
15.1	Embedded Characteristics . . . . .	275
15.2	Block Diagram . . . . .	275
15.3	Functional Description . . . . .	275
15.4	Reset Controller (RSTC) User Interface . . . . .	282
<b>16.</b>	<b>Real-time Timer (RTT) . . . . .</b>	<b>287</b>
16.1	Description . . . . .	287
16.2	Embedded Characteristics . . . . .	287
16.3	Block Diagram . . . . .	287
16.4	Functional Description . . . . .	288
16.5	Real-time Timer (RTT) User Interface . . . . .	290
<b>17.</b>	<b>Reinforced Safety Watchdog Timer (RSWDT) . . . . .</b>	<b>295</b>
17.1	Description . . . . .	295
17.2	Embedded Characteristics . . . . .	295
17.3	Block Diagram . . . . .	296
17.4	Functional Description . . . . .	297
17.5	Reinforced Safety Watchdog Timer (RSWDT) User Interface . . . . .	299
<b>18.</b>	<b>Real-time Clock (RTC) . . . . .</b>	<b>303</b>
18.1	Description . . . . .	303
18.2	Embedded Characteristics . . . . .	303
18.3	Block Diagram . . . . .	304
18.4	Product Dependencies . . . . .	304
18.5	Functional Description . . . . .	304

18.6	Real-time Clock (RTC) User Interface .....	310
<b>19.</b>	<b>Watchdog Timer (WDT)</b> .....	<b>325</b>
19.1	Description .....	325
19.2	Embedded Characteristics .....	325
19.3	Block Diagram .....	326
19.4	Functional Description .....	327
19.5	Watchdog Timer (WDT) User Interface .....	329
<b>20.</b>	<b>Supply Controller (SUPC)</b> .....	<b>335</b>
20.1	Description .....	335
20.2	Embedded Characteristics .....	335
20.3	Block Diagram .....	336
20.4	Supply Controller Functional Description .....	337
20.5	Supply Controller (SUPC) User Interface .....	345
<b>21.</b>	<b>General Purpose Backup Registers (GPBR)</b> .....	<b>355</b>
21.1	Description .....	355
21.2	Embedded Characteristics .....	355
21.3	General Purpose Backup Register (GPBR) User Interface .....	356
<b>22.</b>	<b>Embedded Flash Controller (EFC)</b> .....	<b>357</b>
22.1	Description .....	357
22.2	<b>Embedded Characteristics</b> .....	<b>357</b>
22.3	Product Dependencies .....	357
22.4	Functional Description .....	358
22.5	Enhanced Embedded Flash Controller (EEFC) User Interface .....	371
<b>23.</b>	<b>Fast Flash Programming Interface (FFPI)</b> .....	<b>377</b>
23.1	Description .....	377
23.2	<b>Embedded Characteristics</b> .....	<b>377</b>
23.3	Parallel Fast Flash Programming .....	378
<b>24.</b>	<b>Cortex M Cache Controller (CMCC)</b> .....	<b>387</b>
24.1	Description .....	387
24.2	Embedded Characteristics .....	387
24.3	Block Diagram .....	388
24.4	Functional Description .....	388
24.5	Cortex M Cache Controller (CMCC) User Interface .....	390
<b>25.</b>	<b>SAM-BA Boot Program for SAM4E Microcontrollers</b> .....	<b>403</b>
25.1	Description .....	403
25.2	<b>Embedded Characteristics</b> .....	<b>403</b>
25.3	Hardware and Software Constraints .....	403
25.4	Flow Diagram .....	404
25.5	Device Initialization .....	404
25.6	SAM-BA Monitor .....	405
<b>26.</b>	<b>Bus Matrix (MATRIX)</b> .....	<b>409</b>
26.1	Description .....	409
26.2	<b>Embedded Characteristics</b> .....	<b>409</b>
26.3	Memory Mapping .....	411

26.4	Special Bus Granting Mechanism . . . . .	411
26.5	No Default Master . . . . .	411
26.6	Last Access Master . . . . .	412
26.7	Fixed Default Master . . . . .	412
26.8	Arbitration . . . . .	412
26.9	System I/O Configuration . . . . .	415
26.10	SMC NAND Flash Chip Select Configuration . . . . .	415
26.11	Write Protect Registers . . . . .	416
26.12	Bus Matrix (MATRIX) User Interface . . . . .	417
<b>27.</b>	<b>DMA Controller (DMAC)</b> . . . . .	<b>427</b>
27.1	Description . . . . .	427
27.2	Embedded Characteristics . . . . .	427
27.3	Block Diagram . . . . .	428
27.4	Functional Description . . . . .	430
27.5	DMAC Software Requirements . . . . .	444
27.6	Write Protection Registers . . . . .	445
27.7	DMA Controller (DMAC) User Interface . . . . .	446
<b>28.</b>	<b>Static Memory Controller (SMC)</b> . . . . .	<b>469</b>
28.1	Description . . . . .	469
28.2	<b>Embedded Characteristics</b> . . . . .	<b>469</b>
28.3	I/O Lines Description . . . . .	470
28.4	Product Dependencies . . . . .	470
28.5	External Memory Mapping . . . . .	471
28.6	Connection to External Devices . . . . .	472
28.7	Application Example . . . . .	474
28.8	Standard Read and Write Protocols . . . . .	476
28.9	Scrambling/Unscrambling Function . . . . .	483
28.10	Automatic Wait States . . . . .	483
28.11	Data Float Wait States . . . . .	488
28.12	External Wait . . . . .	492
28.13	Slow Clock Mode . . . . .	498
28.14	Asynchronous Page Mode . . . . .	500
28.15	Static Memory Controller (SMC) User Interface . . . . .	503
<b>29.</b>	<b>Peripheral DMA Controller (PDC)</b> . . . . .	<b>515</b>
29.1	Description . . . . .	515
29.2	Embedded Characteristics . . . . .	515
29.3	Block Diagram . . . . .	516
29.4	Functional Description . . . . .	517
29.5	Peripheral DMA Controller (PDC) User Interface . . . . .	519
<b>30.</b>	<b>Power Management Controller (PMC)</b> . . . . .	<b>531</b>
30.1	Clock Generator . . . . .	531
30.2	Power Management Controller (PMC) . . . . .	539
<b>31.</b>	<b>Advanced Encryption Standard (AES)</b> . . . . .	<b>581</b>
31.1	Description . . . . .	581
31.2	Embedded Characteristics . . . . .	581
31.3	Product Dependencies . . . . .	582

31.4	Functional Description .....	582
31.5	Security Features .....	587
31.6	Advanced Encryption Standard (AES) User Interface .....	588
<b>32.</b>	<b>Controller Area Network (CAN) Programmer Datasheet .....</b>	<b>599</b>
32.1	Description .....	599
32.2	Embedded Characteristics .....	599
32.3	Block Diagram .....	600
32.4	Application Block Diagram .....	600
32.5	I/O Lines Description .....	601
32.6	Product Dependencies .....	601
32.7	CAN Controller Features .....	602
32.8	Functional Description .....	613
32.9	Controller Area Network (CAN) User Interface .....	626
<b>33.</b>	<b>Parallel Input/Output (PIO) Controller .....</b>	<b>657</b>
33.1	Description .....	657
33.2	Embedded Characteristics .....	657
33.3	Block Diagram .....	657
33.4	Product Dependencies .....	660
33.5	Functional Description .....	661
33.6	I/O Lines Programming Example .....	674
33.7	Parallel Input/Output Controller (PIO) User Interface .....	676
<b>34.</b>	<b>Serial Peripheral Interface (SPI) .....</b>	<b>721</b>
34.1	Description .....	721
34.2	Embedded Characteristics .....	722
34.3	Block Diagram .....	723
34.4	Application Block Diagram .....	723
34.5	Signal Description .....	724
34.6	Product Dependencies .....	724
34.7	Functional Description .....	725
34.8	Serial Peripheral Interface (SPI) User Interface .....	739
<b>35.</b>	<b>Two-wire Interface (TWI) .....</b>	<b>755</b>
35.1	Description .....	755
35.2	Embedded Characteristics .....	755
35.3	List of Abbreviations .....	756
35.4	Block Diagram .....	756
35.5	Application Block Diagram .....	757
35.6	Product Dependencies .....	757
35.7	Functional Description .....	758
35.8	Master Mode .....	759
35.9	Multi-master Mode .....	772
35.10	Slave Mode .....	775
35.11	Write Protection System .....	782
35.12	Two-wire Interface (TWI) User Interface .....	783
<b>36.</b>	<b>Universal Asynchronous Receiver Transmitter (UART) .....</b>	<b>801</b>
36.1	Description .....	801
36.2	Embedded Characteristics .....	801
36.3	Block Diagram .....	802

36.4	Product Dependencies .....	803
36.5	UART Operations .....	804
36.6	Universal Asynchronous Receiver Transmitter (UART) User Interface .....	810
<b>37.</b>	<b>Universal Synchronous Asynchronous Receiver Transmitter (USART)</b>	
	<b>821</b>	
37.1	Description .....	821
37.2	Embedded Characteristics .....	821
37.3	Block Diagram .....	822
37.4	Application Block Diagram .....	823
37.5	I/O Lines Description .....	824
37.6	Product Dependencies .....	825
37.7	Functional Description .....	827
37.8	Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface	860
<b>38.</b>	<b>Timer Counter (TC)</b>	<b>895</b>
38.1	Description .....	895
38.2	Embedded Characteristics .....	895
38.3	Block Diagram .....	896
38.4	Pin Name List .....	897
38.5	Product Dependencies .....	897
38.6	Functional Description .....	899
38.7	Timer Counter (TC) User Interface .....	921
<b>39.</b>	<b>High Speed MultiMedia Card Interface (HSMCI)</b>	<b>953</b>
39.1	Description .....	953
39.2	Embedded Characteristics .....	953
39.3	Block Diagram .....	954
39.4	Application Block Diagram .....	955
39.5	Pin Name List .....	955
39.6	Product Dependencies .....	956
39.7	Bus Topology .....	956
39.8	High Speed MultiMedia Card Operations .....	959
39.9	SD/SDIO Card Operation .....	967
39.10	CE-ATA Operation .....	967
39.11	HSMCI Boot Operation Mode .....	969
39.12	HSMCI Transfer Done Timings .....	969
39.13	Write Protection Registers .....	971
39.14	High Speed MultiMedia Card Interface (HSMCI) User Interface .....	972
<b>40.</b>	<b>Pulse Width Modulation Controller (PWM)</b>	<b>999</b>
40.1	Description .....	999
40.2	Embedded Characteristics .....	1000
40.3	Block Diagram .....	1001
40.4	I/O Lines Description .....	1001
40.5	Product Dependencies .....	1002
40.6	Functional Description .....	1004
40.7	Pulse Width Modulation Controller (PWM) Controller User Interface .....	1034
<b>41.</b>	<b>USB Device Port (UDP)</b>	<b>1087</b>
41.1	Description .....	1087

41.2	Embedded Characteristics . . . . .	1087
41.3	Block Diagram . . . . .	1088
41.4	Product Dependencies . . . . .	1088
41.5	Typical Connection . . . . .	1090
41.6	Functional Description . . . . .	1091
41.7	USB Device Port (UDP) User Interface . . . . .	1104
<b>42.</b>	<b>Analog Comparator Controller (ACC) . . . . .</b>	<b>1127</b>
42.1	Description . . . . .	1127
42.2	Embedded characteristics . . . . .	1127
42.3	Block Diagram . . . . .	1128
42.4	Pin Name List . . . . .	1129
42.5	Product Dependencies . . . . .	1129
42.6	Functional Description . . . . .	1130
42.7	Analog Comparator Controller (ACC) User Interface . . . . .	1131
<b>43.</b>	<b>Analog-Front-End Controller (AFEC) . . . . .</b>	<b>1141</b>
43.1	Description . . . . .	1141
43.2	Embedded Characteristics . . . . .	1141
43.3	Block Diagram . . . . .	1142
43.4	Signal Description . . . . .	1142
43.5	Product Dependencies . . . . .	1143
43.6	Functional Description . . . . .	1145
43.7	Analog-Front-End Controller (AFEC) User Interface . . . . .	1161
<b>44.</b>	<b>Ethernet MAC (GMAC) . . . . .</b>	<b>1191</b>
44.1	Description . . . . .	1191
44.2	Embedded Characteristics . . . . .	1191
44.3	Block Diagram . . . . .	1192
44.4	Signal Interface . . . . .	1193
44.5	Functional Description . . . . .	1194
44.6	Programming Interface . . . . .	1213
44.7	Ethernet MAC (GMAC) User Interface . . . . .	1217
<b>45.</b>	<b>Digital-to-Analog Converter Controller (DACC) . . . . .</b>	<b>1287</b>
45.1	Description . . . . .	1287
45.2	Embedded Characteristics . . . . .	1287
45.3	Block Diagram . . . . .	1288
45.4	Signal Description . . . . .	1289
45.5	Product Dependencies . . . . .	1289
45.6	Functional Description . . . . .	1290
45.7	Digital-to-Analog Converter Controller (DACC) User Interface . . . . .	1293
<b>46.</b>	<b>SAM4E Electrical Characteristics . . . . .</b>	<b>1309</b>
46.1	Absolute Maximum Ratings . . . . .	1309
46.2	DC Characteristics . . . . .	1310
46.3	Power Consumption . . . . .	1316
46.4	Oscillator Characteristics . . . . .	1323
46.5	PLL Characteristics . . . . .	1327
46.6	USB Transceiver Characteristics . . . . .	1328
46.7	12-bit AFE (Analog Front End) Characteristics . . . . .	1330

46.8	12-bit DAC Characteristics . . . . .	1336
46.9	Analog Comparator Characteristics . . . . .	1338
46.10	Temperature Sensor . . . . .	1338
46.11	AC Characteristics . . . . .	1339
<b>47.</b>	<b>SAM4E Mechanical Characteristics . . . . .</b>	<b>1355</b>
47.1	100-ball TFBGA Package Drawing . . . . .	1355
47.2	144-ball LFBGA Package Drawing . . . . .	1356
47.3	100-lead LQFP Package Drawing . . . . .	1357
47.4	144-lead LQFP Package Drawing . . . . .	1358
47.5	Soldering Profile . . . . .	1359
47.6	Packaging Resources . . . . .	1359
<b>48.</b>	<b>Ordering Information . . . . .</b>	<b>1361</b>
<b>49.</b>	<b>Errata on SAM4E Devices . . . . .</b>	<b>1363</b>
49.1	Marking . . . . .	1363
49.2	Errata . . . . .	1363
	..... Revision History	1365
	..... Table of Contents	i



**Atmel Corporation**  
1600 Technology Drive  
San Jose, CA 95110  
USA  
**Tel:** (+1) (408) 441-0311  
**Fax:** (+1) (408) 487-2600  
[www.atmel.com](http://www.atmel.com)

**Atmel Asia Limited**  
Unit 01-5 & 16, 19F  
BEA Tower, Millennium City 5  
418 Kwun Tong Road  
Kwun Tong, Kowloon  
HONG KONG  
**Tel:** (+852) 2245-6100  
**Fax:** (+852) 2722-1369

**Atmel Munich GmbH**  
Business Campus  
Parkring 4  
D-85748 Garching b. Munich  
GERMANY  
**Tel:** (+49) 89-31970-0  
**Fax:** (+49) 89-3194621

**Atmel Japan G.K.**  
16F Shin-Osaki Kangyo Bldg  
1-6-4 Osaki, Shinagawa-ku  
Tokyo 141-0032  
JAPAN  
**Tel:** (+81) (3) 6417-0300  
**Fax:** (+81) (3) 6417-0370

© 2013 Atmel Corporation. All rights reserved. / Rev.: 11157C-ATARM-25-Jul-13

Atmel®, logo and combinations thereof, Enabling Unlimited Possibilities®, SAM-BA® and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. ARM®, Thumb® and Cortex® are registered trademarks or trademarks of ARM Ltd. Windows® and others are registered trademarks or trademarks of Microsoft Corporation in the US and/or other countries. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATTEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATTEL WEBSITE, ATTEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATTEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.