SAM3X / SAM3A Series

Atmel | SMART ARM-based MCU

DATASHEET

Description

The Atmel® | SMART SAM3X/A series is a member of a family of Flash microcontrollers based on the high performance 32-bit ARM® Cortex®-M3 RISC processor. It operates at a maximum speed of 84 MHz and features up to 512 Kbytes of Flash and up to 100 Kbytes of SRAM. The peripheral set includes a High Speed USB Host and Device port with embedded transceiver, an Ethernet MAC, 2 CANs, a High Speed MCI for SDIO/SD/MMC, an External Bus Interface with NAND Flash Controller (NFC), 5 UARTs, 2 TWIs, 4 SPIs, as well as a PWM timer, three 3-channel general-purpose 32-bit timers, a low-power RTC, a low-power RTT, 256-bit General Purpose Backup Registers, a 12-bit ADC and a 12-bit DAC.

The SAM3X/A devices have three software-selectable low-power modes: Sleep, Wait and Backup. In Sleep mode, the processor is stopped while all other functions can be kept running. In Wait mode, all clocks and functions are stopped but some peripherals can be configured to wake up the system based on predefined conditions. In Backup mode, only the RTC, RTT, and wake-up logic are running.

The SAM3X/A series is ready for capacitive touch thanks to the QTouch library, offering an easy way to implement buttons, wheels and sliders.

The SAM3X/A architecture is specifically designed to sustain high-speed data transfers. It includes a multi-layer bus matrix as well as multiple SRAM banks, PDC and DMA channels that enable it to run tasks in parallel and maximize data throughput.

The device operates from 1.62V to 3.6V and is available in 100 and 144-lead LQFP, 100-ball TFBGA and 144-ball LFBGA packages.

The SAM3X/A devices are particularly well suited for networking applications: industrial and home/building automation, gateways.

1. Features

- Core
  - ARM Cortex-M3 revision 2.0 running at up to 84 MHz
  - Memory Protection Unit (MPU)
  - Thumb®-2 instruction set
  - 24-bit SysTick Counter
  - Nested Vector Interrupt Controller
- Memories
  - 256 to 512 Kbytes embedded Flash, 128-bit wide access, memory accelerator, dual bank
  - 32 to 100 Kbytes embedded SRAM with dual banks
  - 16 Kbytes ROM with embedded bootloader routines (UART, USB) and IAP routines
  - Static Memory Controller (SMC): SRAM, NOR, NAND support. NFC with 4 Kbyte RAM buffer and ECC
- System
  - Embedded voltage regulator for single supply operation
  - Power-on-Reset (POR), Brown-out Detector (BOD) and Watchdog for safe reset
  - Quartz or ceramic resonator oscillators: 3 to 20 MHz main and optional low power 32.768 kHz for RTC or device clock
  - High precision 8/12 MHz factory trimmed internal RC oscillator with 4 MHz default frequency for fast device startup
  - Slow Clock Internal RC oscillator as permanent clock for device clock in low-power mode
  - One PLL for device clock and one dedicated PLL for USB 2.0 High Speed Mini Host/Device
  - Temperature Sensor
  - Up to 17 peripheral DMA (PDC) channels and 6-channel central DMA plus dedicated DMA for High-Speed USB Mini Host/Device and Ethernet MAC
- Low-power Modes
  - Sleep, Wait and Backup modes, down to 2.5 µA in Backup mode with RTC, RTT, and GPBR
- Peripherals
  - USB 2.0 Device/Mini Host: 480 Mbps, 4 Kbyte FIFO, up to 10 bidirectional Endpoints, dedicated DMA
  - Up to 4 USARTs (ISO7816, IrDA®, Flow Control, SPI, Manchester and LIN support) and one UART
  - 2 TWI (I2C compatible), up to 6 SPIs, 1 SSC (I2S), 1 HSMCI (SDIO/SD/MMC) with up to 2 slots
  - 9-channel 32-bit Timer Counter (TC) for capture, compare and PWM mode, Quadrature Decoder Logic and 2-bit Gray Up/Down Counter for Stepper Motor
  - Up to 8-channel 16-bit PWM (PWMC) with Complementary Output, Fault Input, 12-bit Dead Time Generator Counter for Motor Control
  - 32-bit low-power Real-time Timer (RTT) and low-power Real-time Clock (RTC) with calendar and alarm features
  - 256-bit General Purpose Backup Registers (GPBR)
  - 16-channel 12-bit 1 msps ADC with differential input mode and programmable gain stage
  - 2-channel 12-bit 1 msps DAC
  - Ethernet MAC 10/100 (EMAC) with dedicated DMA
  - 2 CAN Controllers with 8 Mailboxes
  - True Random Number Generator (TRNG)
  - Register Write Protection
- I/O
  - Up to 103 I/O lines with external interrupt capability (edge or level sensitivity), debouncing, glitch filtering and on-die Series Resistor Termination
  - Up to six 32-bit Parallel Input/Outputs (PIO)

- Packages
  - 100-lead LQFP – 14 x 14 mm, pitch 0.5 mm
  - 100-ball TFBGA – 9 x 9 mm, pitch 0.8 mm
  - 144-lead LQFP – 20 x 20 mm, pitch 0.5 mm
  - 144-ball LFBGA – 10 x 10 mm, pitch 0.8 mm

1.1 Configuration Summary

The SAM3X/A series devices differ in memory sizes, package and features list. Table 1-1 summarizes the configurations.

Table 1-1. Configuration Summary

Feature	SAM3X8E	SAM3X8C	SAM3X4E	SAM3X4C	SAM3A8C	SAM3A4C
Flash	2 x 256 Kbytes	2 x 256 Kbytes	2 x 128 Kbytes	2 x 128 Kbytes	2 x 256 Kbytes	2 x 128 Kbytes
SRAM	64 + 32 Kbytes	64 + 32 Kbytes	32 + 32 Kbytes	32 + 32 Kbytes	64 + 32 Kbytes	32 + 32 Kbytes
NAND Flash Controller (NFC)	Yes	–	Yes	–	–	–
NFC SRAM(1)	4 Kbytes	–	4 Kbytes	–	–	–
Package	LQFP144 LFBGA144	LQFP100 TFBGA100	LQFP144 LFBGA144	LQFP100 TFBGA100	LQFP100 TFBGA100	LQFP100 TFBGA100
Number of PIOs	103	63	103	63	63	63
SHDN Pin	Yes	No	Yes	No	No	No
EMAC	MII/RMII	RMII	MII/RMII	RMII	–	–
External Bus Interface	16-bit data, 8 chip selects, 23-bit address	–	16-bit data, 8 chip selects, 23-bit address	–	–	–
SDRAM Controller(6)	–	–	–	–	–	–
Central DMA	6	4	6	4	4	4
12-bit ADC	16 ch.(2)	16 ch.(2)	16 ch.(2)	16 ch.(2)	16 ch.(2)	16 ch.(2)
12-bit DAC	2 ch.	2 ch.	2 ch.	2 ch.	2 ch.	2 ch.
32-bit Timer	9 ch.(3)	9 ch.(4)	9 ch.(3)	9 ch.(4)	9 ch.(3)	9 ch.(3)
PDC Channels	17	15	17	15	15	15
USART/UART	3/2(5)	3/1	3/2(5)	3/1	3/1	3/1
SPI	1 SPI controller, 4 chip selects + 3 USART with SPI mode	1 SPI controller, 4 chip selects + 3 USART with SPI mode	1 SPI controller, 4 chip selects + 3 USART with SPI mode	1 SPI controller, 4 chip selects + 3 USART with SPI mode	1 SPI controller, 4 chip selects + 3 USART with SPI mode	1 SPI controller, 4 chip selects + 3 USART with SPI mode
HSMCI	1 slot, 8 bits	1 slot, 4 bits	1 slot, 8 bits	1 slot, 4 bits	1 slot, 4 bits	1 slot, 4 bits

Notes:
1. 4 Kbytes RAM buffer of the NFC which can be used by the core if not used by the NFC
2. One channel is reserved for internal temperature sensor
3. Six TC channels are accessible through PIO
4. Three TC channels are accessible through PIO
5. USART3 in UART mode (RXD3 and TXD3 available)
6. Available only on SAM3X8H in LFBGA217 package, which is mounted on SAM3X-EK evaluation kit for SAM3X and SAM3A series. The SAM3X8H device is not commercially available.

2. SAM3X/A Block Diagram

Figure 2-1. SAM3A4/8C (100 pins) Block Diagram

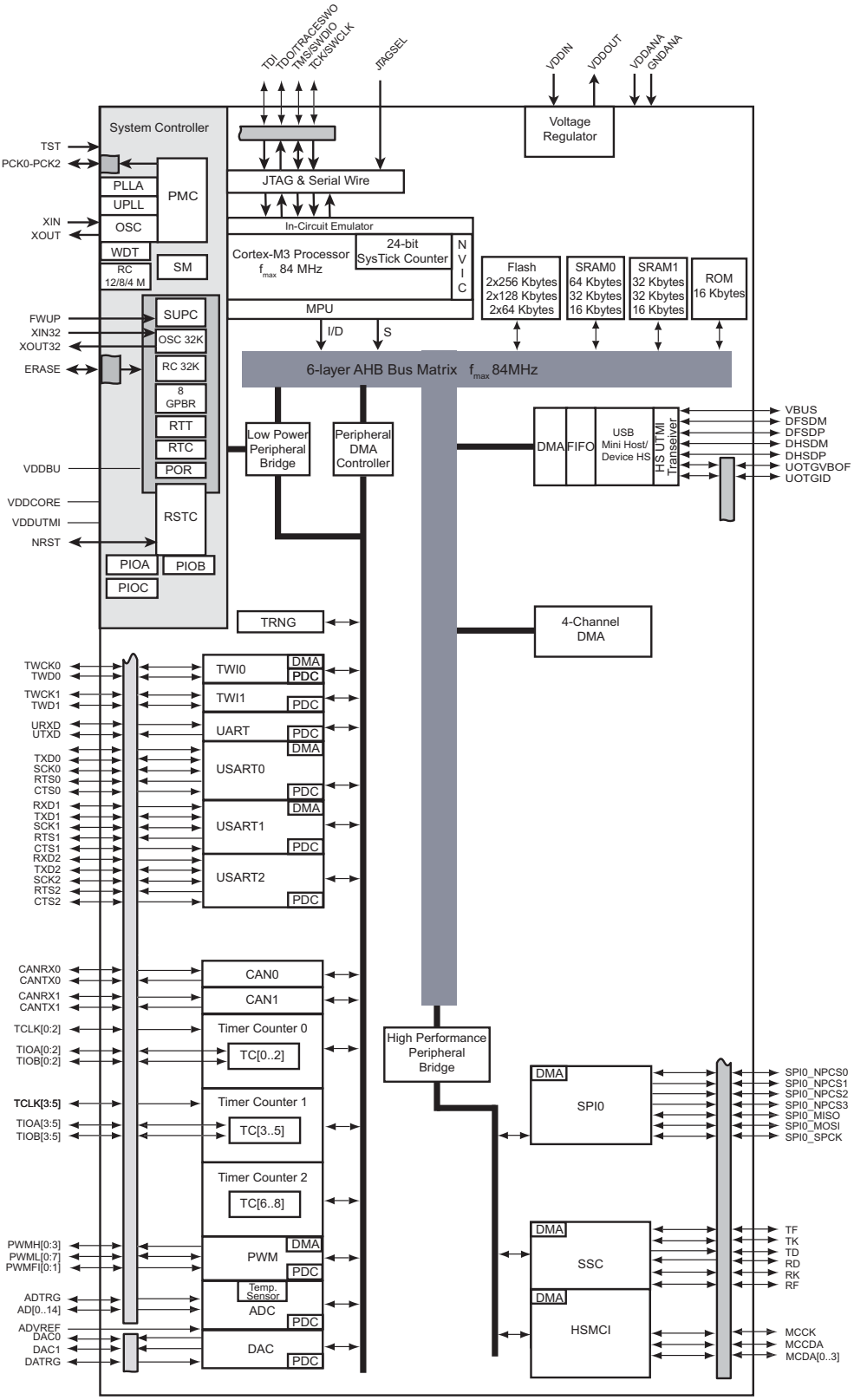


Figure 2-2. SAM3X4/8C (100 pins) Block Diagram

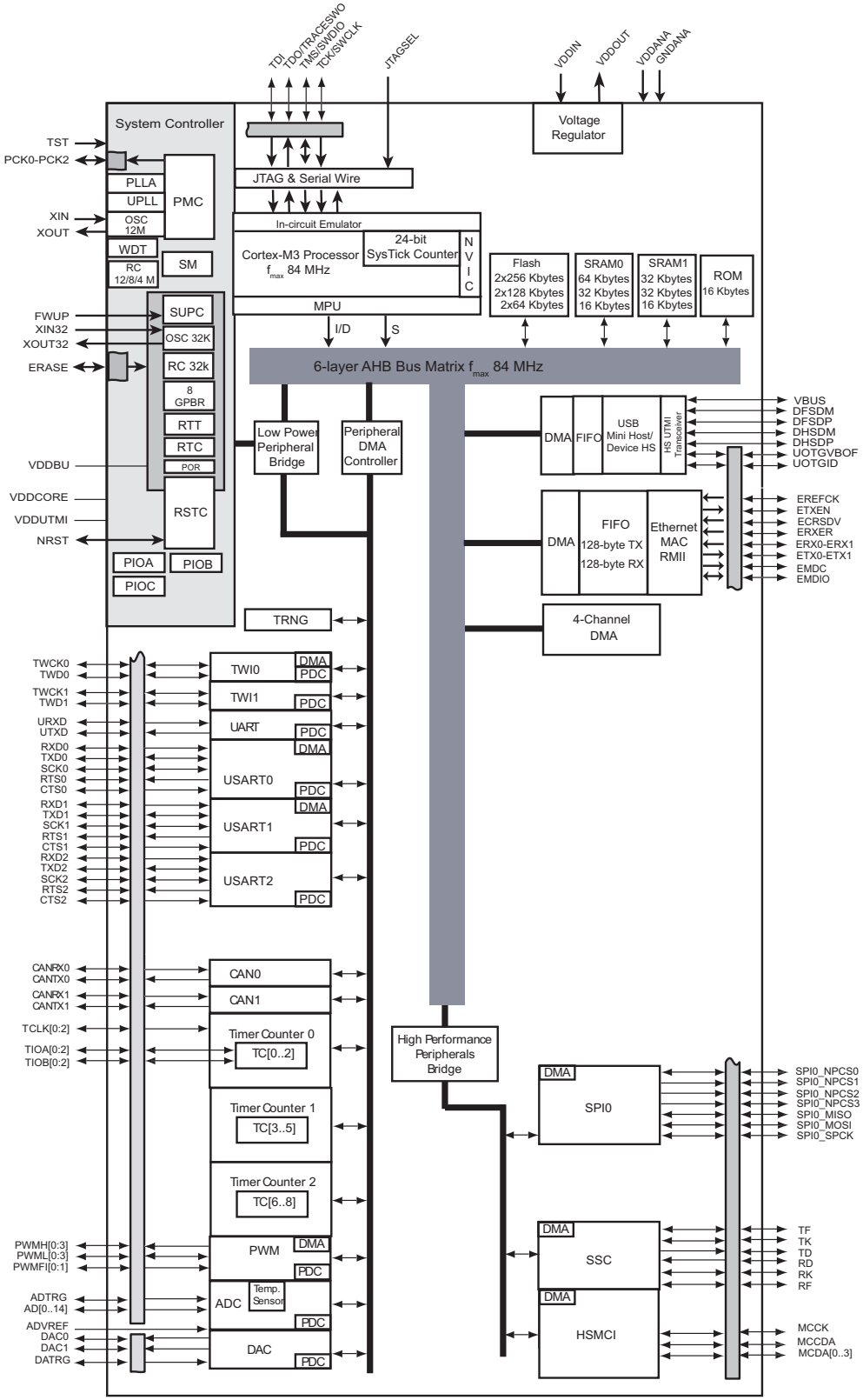


Figure 2-3. SAM3X4/8E (144 pins) Block Diagram

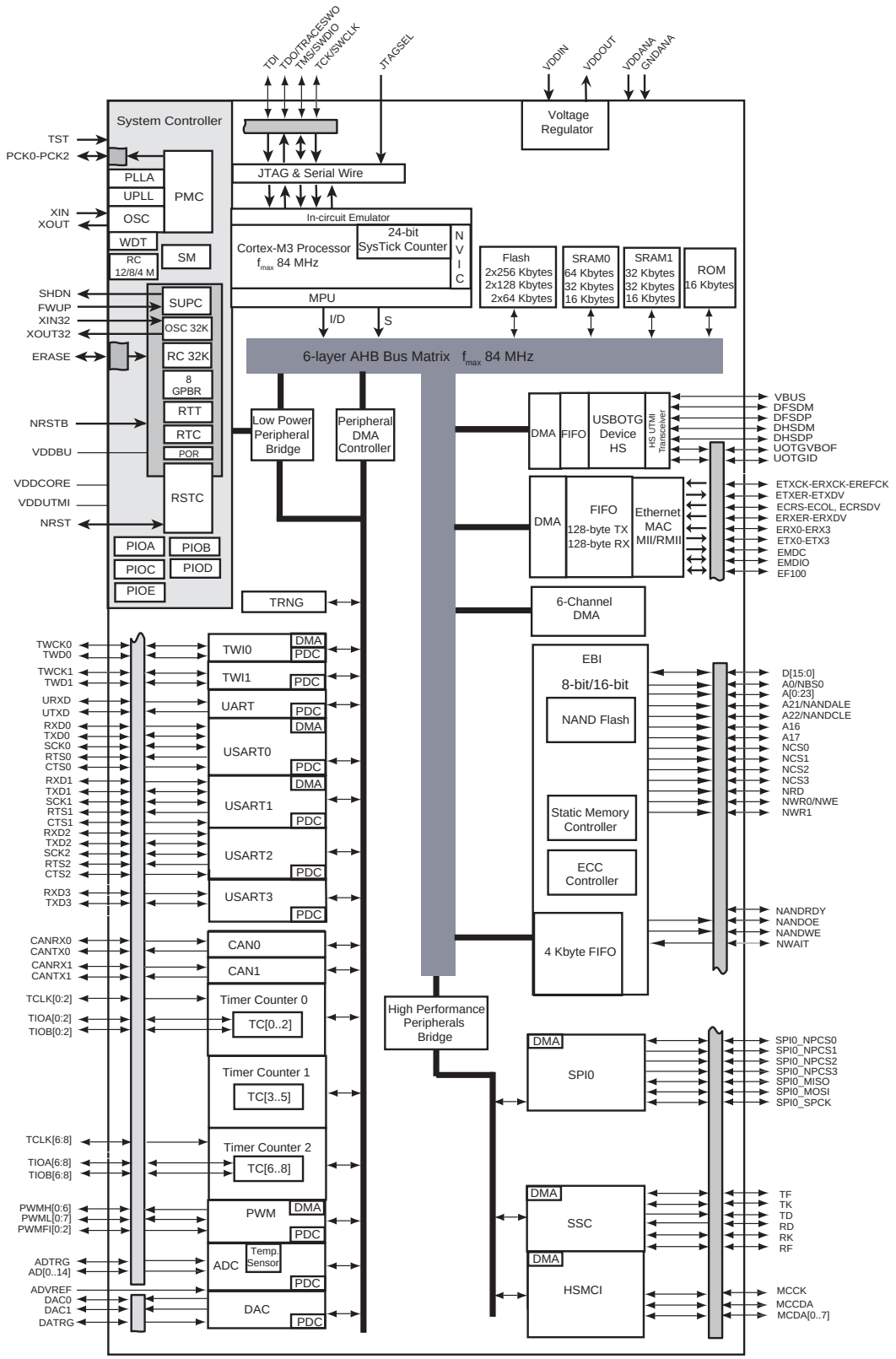
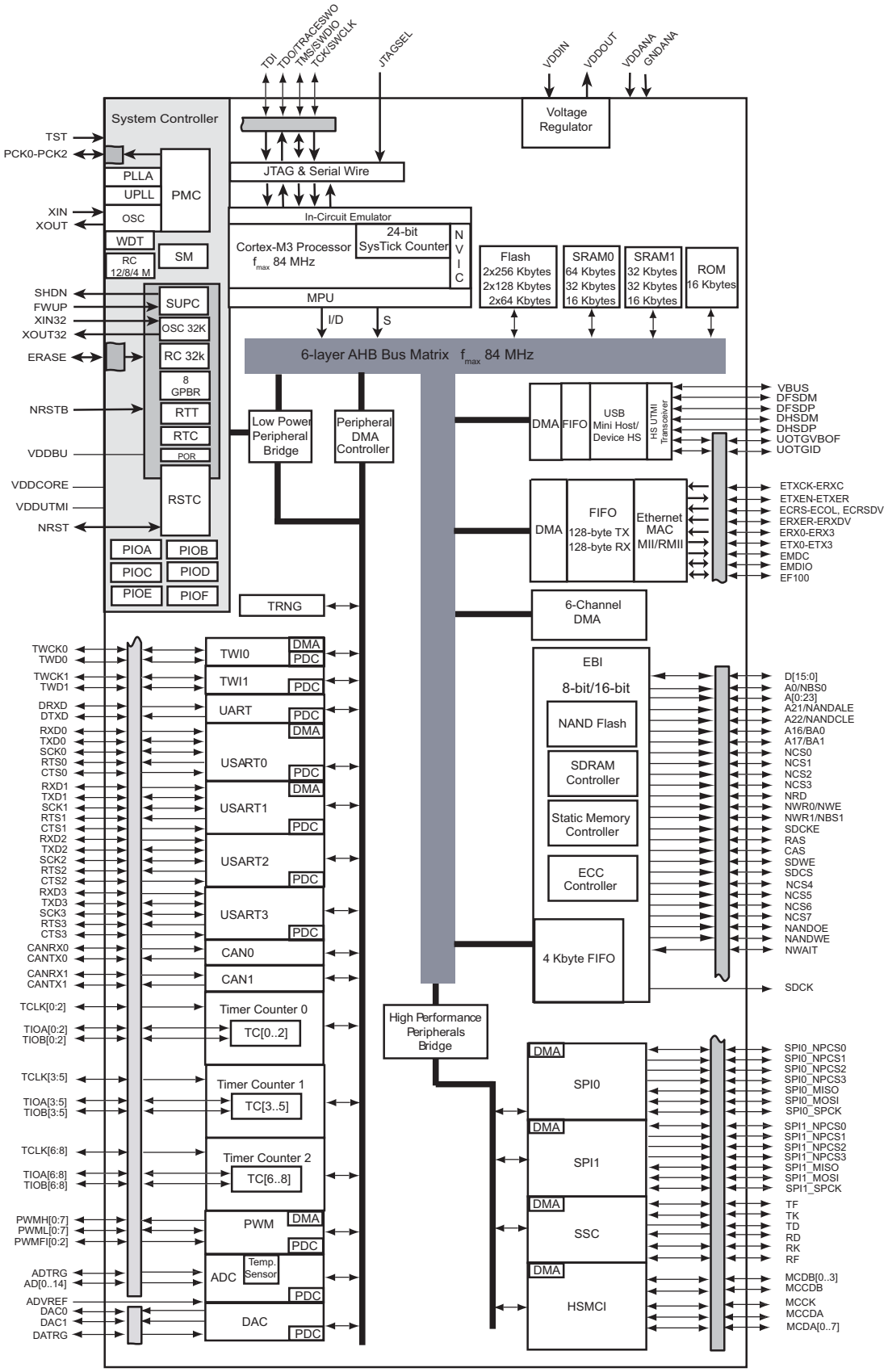


Figure 2-4. SAM3X8H (217 pins) Block Diagram (not commercially available)



3. Signal Description

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

Table 3-1. Signal Description List

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
Power Supplies					
VDDIO	Peripherals I/O Lines Power Supply	Power			1.62V to 3.6V
VDDUTMI	USB UTMI+ Interface Power Supply	Power			3.0V to 3.6V
VDDOUT	Voltage Regulator Output	Power			
VDDIN	Voltage Regulator, ADC and DAC Power Supply	Power			
GNDUTMI	USB UTMI+ Interface Ground	Ground			
VDDBU	Backup I/O Lines Power Supply	Power			1.62V to 3.6V
GNDBU	Backup Ground	Ground			
VDDPLL	PLL A, UPLL and Oscillator Power Supply	Power			1.62 V to 1.95V
GNDPLL	PLL A, UPLL and Oscillator Ground	Ground			
VDDANA	ADC and DAC Analog Power Supply	Power			2.0V to 3.6V
GNDANA	ADC and DAC Analog Ground	Ground			
VDDCORE	Core Chip Power Supply	Power			1.62V to 1.95V
GND	Ground	Ground			
Clocks, Oscillators and PLLs					
XIN	Main Oscillator Input	Input		VDDPLL	
XOUT	Main Oscillator Output	Output			
XIN32	Slow Clock Oscillator Input	Input		VDDBU	
XOUT32	Slow Clock Oscillator Output	Output			
VBG	Bias Voltage Reference	Analog			
PCK0–PCK2	Programmable Clock Output	Output			
Shutdown, Wakeup Logic					
SHDN	Shut-Down Control	Output		VDDBU	0: Device is in backup mode 1: Device is running (not in backup mode)
FWUP	Force Wake-up Input	Input		VDDBU	Needs external pull-up

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
ICE and JTAG					
TCK/SWCLK	Test Clock/Serial Wire Clock	Input		VDDIO	Reset State: - SWJ-DP Mode - Internal pull-up disabled[1]
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 / I/O			
JTAGSEL	JTAG Selection	Input	High	VDDBU	Permanent Internal pull-down
Flash Memory					
ERASE	Flash and NVM Configuration Bits Erase Command	Input	High	VDDIO	Pull-down resistor
Reset/Test					
NRST	Microcontroller Reset	I/O	Low	VDDIO	Pull-up resistor
NRSTB	Asynchronous Microcontroller Reset	Input	Low	VDDBU	Pull-up resistor
TST	Test Mode Select	Input		VDDBU	Pull-down resistor
Universal Asynchronous Receiver Transceiver - UART					
URXD	UART Receive Data	Input			
UTXD	UART Transmit Data	Output			
PIO Controller - PIOA - PIOB - PIOC - PIOD - PIOE - PIOF					
PA0–PA31	Parallel IO Controller A	I/O		VDDIO	Schmitt Trigger[3] Reset State: - PIO Input - Internal pull-up enabled
PB0–PB31	Parallel IO Controller B	I/O			Schmitt Trigger[4] Reset State: - PIO Input - Internal pull-up enabled
PC0–PC30	Parallel IO Controller C	I/O			Schmitt Trigger[5] Reset State: - PIO Input - Internal pull-up enabled
PD0–PD30	Parallel IO Controller D	I/O			Schmitt Trigger[6] Reset State: - PIO Input - Internal pull-up enabled
PE0–PE31	Parallel IO Controller E	I/O			Schmitt Trigger[7] Reset State: - PIO Input - Internal pull-up enabled
PF0–PF6	Parallel IO Controller F	I/O			Schmitt Trigger[7] Reset State: - PIO Input - Internal pull-up enabled

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
External Memory Bus					
D0–D15	Data Bus	I/O			Pulled-up input at reset
A0–A23	Address Bus	Output			0 at reset
Static Memory Controller - SMC					
NCS0–NCS7	Chip Select Lines	Output	Low		
NWR0–NWR1	Write Signal	Output	Low		
NRD	Read Signal	Output	Low		
NWE	Write Enable	Output	Low		
NBS0–NBS1	Byte Mask Signal	Output	Low		
NWAIT	External Wait Signal	Input	Low		
NAND Flash Controller - NFC					
NANDOE	NAND Flash Output Enable	Output	Low		
NANDWE	NAND Flash Write Enable	Output	Low		
NANDRDY	NAND Ready	Input			
NANDCLE	NAND Flash Command Line Enable	Output	Low		
NANDALE	NAND Flash Address Line Enable	Output	Low		
SDRAM Controller - SDRAMC					
SDCK	SDRAM Clock	Output			Tied low after reset
SDCKE	SDRAM Clock Enable	Output	High		
SDCS	SDRAM Controller Chip Select Line	Output	Low		
BA[1:0]	Bank Select	Output			
SDWE	SDRAM Write Enable	Output	Low		
RAS - CAS	Row and Column Signal	Output	Low		
NBS[1:0]	Byte Mask Signals	Output	Low		
SDA10	SDRAM Address 10 Line	Output			
High Speed Multimedia Card Interface - HSMCI					
MCCK	Multimedia Card Clock	I/O			
MCCDA	Multimedia Card Slot A Command	I/O			
MCDA0–MCDA7	Multimedia Card Slot A Data	I/O			
MCCDB	Multimedia Card Slot B Command	I/O			
MCDB0–MCDB3	Multimedia Card Slot A Data	I/O			
Universal Synchronous Asynchronous Receiver Transmitter - USARTx					
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			

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
Ethernet MAC 10/100 - EMAC					
EREFCK	Reference Clock	Input		RMII only	
ETXCK	Transmit Clock	Input		MII only	
ERXCK	Receive Clock	Input		MII only	
ETXEN	Transmit Enable	Output			
ETX0–ETX3	Transmit Data	Output		ETX0–ETX1 only in RMII	
ETXER	Transmit Coding Error	Output		MII only	
ERXDV	Receive Data Valid	Input		MII only	
ECRSDV	Carrier Sense and Data Valid	Input		RMII only	
ERX0–ERX3	Receive Data	Input		ERX0–ERX1 only in RMII	
ERXER	Receive Error	Input			
ECRS	Carrier Sense	Input		MII only	
ECOL	Collision Detected	Input		MII only	
EMDC	Management Data Clock	Output			
EMDIO	Management Data Input/Output	I/O			
CAN Controller - CANx					
CANRXx	CAN Input	Input			
CANTXx	CAN Output	Output			
Synchronous Serial Controller - SSC					
TD	SSC Transmit Data	Output			
RD	SSC Receive Data	Input			
TK	SSC Transmit Clock	I/O			
RK	SSC Receive Clock	I/O			
TF	SSC Transmit Frame Sync	I/O			
RF	SSC Receive Frame Sync	I/O			
Timer/Counter - TC					
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			
Pulse Width Modulation Controller - PWMC					
PWMHx	PWM Waveform Output High for channel x	Output			
PWMLx	PWM Waveform Output Low for channel x	Output			Only output in complementary mode when dead time insertion is enabled
PWMFIx	PWM Fault Input for channel x	Input			

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
Serial Peripheral Interface - SPIx					
SPIx_MISO	Master In Slave Out	I/O			
SPIx_MOSI	Master Out Slave In	I/O			
SPIx_SPCK	SPI Serial Clock	I/O			
SPIx_NPCS0	SPI Peripheral Chip Select 0	I/O	Low		
SPIx_NPCS1–SPIx_NPCS3	SPI Peripheral Chip Select	Output	Low		
Two-Wire Interface - TWIx					
TWDx	TWIx Two-wire Serial Data	I/O			
TWCKx	TWIx Two-wire Serial Clock	I/O			
Analog-to-Digital Converter - ADC					
AD0–AD14	Analog Inputs	Analog			
ADTRG	ADC Trigger	Input			
ADVREF	ADC and DAC Reference	Analog			
Digital-to-Analog Converter Controller - DACC					
DAC0	DAC channel 0 analog output	Analog			
DAC1	DAC channel 1 analog output	Analog			
DATRG	DAC Trigger				
Fast Flash Programming Interface - FFPI					
PGMEN0–PGMEN2	Programming Enabling	Input		VDDIO	
PGMM0–PGMM3	Programming Mode	Input		VDDIO	
PGMD0–PGMD15	Programming Data	I/O		VDDIO	
PGMRDY	Programming Ready	Output	High	VDDIO	
PGMNVALID	Data Direction	Output	Low	VDDIO	
PGMNOE	Programming Read	Input	Low	VDDIO	
PGMCK	Programming Clock	Input		VDDIO	
PGMNCMD	Programming Command	Input	Low	VDDIO	
USB High Speed Device					
VBUS	USB Bus Power Measurement Mini Host/Device	Analog			
DFSDM	USB Full Speed Data -	Analog		VDDUTMI	
DFSDP	USB Full Speed Data +	Analog		VDDUTMI	
DHSDM	USB High Speed Data -	Analog		VDDUTMI	
DHSDP	USB High Speed Data +	Analog		VDDUTMI	
UOTGVBOF	USB VBus On/Off: Bus Power Control Port			VDDIO	
UOTGID	USB Identification: Mini Connector Identification Port			VDDIO	

Notes:
1. TDO pin is set in input mode when the Cortex-M3 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.
2. PIOA: Schmitt Trigger on all, except PA0, PA9, PA26, PA29, PA30, PA31
3. PIOB: Schmitt Trigger on all, except PB14 and PB22

4. PIOC: Schmitt Trigger on all, except PC2 to PC9, PC15 to PC24
5. PIOD: Schmitt Trigger on all, except PD10 to PD30
6. PIOE: Schmitt Trigger on all, except PE0 to PE4, PE15, PE17, PE19, PE21, PE23, PE25, PE29
7. PIOF: Schmitt Trigger on all PIOs

3.1 Design Considerations

To facilitate schematic capture when using a SAM3X/A design, refer to the application note *Atmel AT03462: ATSAM3X and ATSAM3A Series - Checklist* (literature No. 42187) available on www.atmel.com.

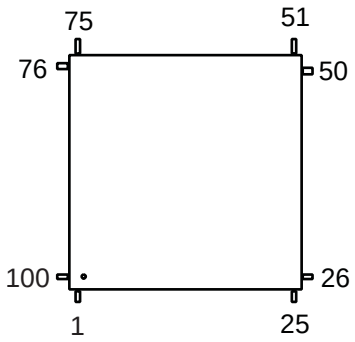
4. Package and Pinout

4.1 SAM3A4/8C and SAM3X4/8C Package and Pinout

The SAM3A4/8C and SAM3X4/8C are available in 100-lead LQFP and 100-ball TFBGA packages.

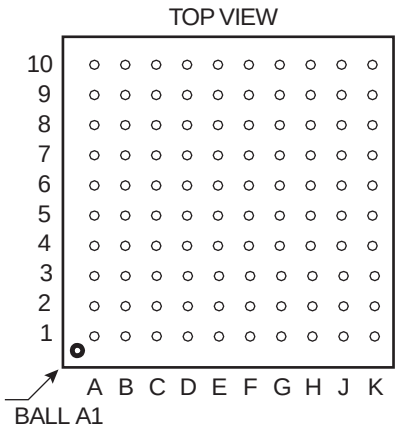
4.1.1 100-lead LQFP Package Outline

Figure 4-1. Orientation of the 100-lead LQFP Package



4.1.2 100-ball TFBGA Package Outline

Figure 4-2. Orientation of the 100-ball TFBGA Package



4.1.3 100-lead LQFP Pinout

Table 4-1. 100-lead LQFP SAM3A4/8C and SAM3X4/8C Pinout

1	PB26	26	DHSDP	51	VDDANA	76	PA26
2	PA9	27	DHSDM	52	GNDANA	77	PA27
3	PA10	28	VBUS	53	ADVREF	78	PA28
4	PA11	29	VBG	54	PB15	79	PA29
5	PA12	30	VDDUTMI	55	PB16	80	PB0
6	PA13	31	DFSDP	56	PA16	81	PB1
7	PA14	32	DFSDM	57	PA24	82	PB2
8	PA15	33	GNDUTMI	58	PA23	83	PB3
9	PA17	34	VDDCORE	59	PA22	84	PB4
10	VDDCORE	35	JTAGSEL	60	PA6	85	PB5
11	VDDIO	36	XIN32	61	PA4	86	PB6
12	GND	37	XOUT32	62	PA3	87	PB7
13	PA0	38	TST	63	PA2	88	PB8
14	PA1	39	VDDBU	64	PB12	89	VDDCORE
15	PA5	40	FWUP	65	PB13	90	VDDIO
16	PA7	41	GND	66	PB17	91	GND
17	PA8	42	VDDOUT	67	PB18	92	PB9
18	PB28	43	VDDIN	68	PB19	93	PB10
19	PB29	44	GND	69	PB20	94	PB11
20	PB30	45	VDDCORE	70	PB21	95	PC0
21	PB31	46	PB27	71	VDDCORE	96	PB14
22	GNDPLL	47	NRST	72	VDDIO	97	PB22
23	VDDPLL	48	PA18	73	GND	98	PB23
24	XOUT	49	PA19	74	PA21	99	PB24
25	XIN	50	PA20	75	PA25	100	PB25

4.1.4 100-ball TFBGA Pinout

Table 4-2. 100-ball TFBGA SAM3X4/8E Package and Pinout

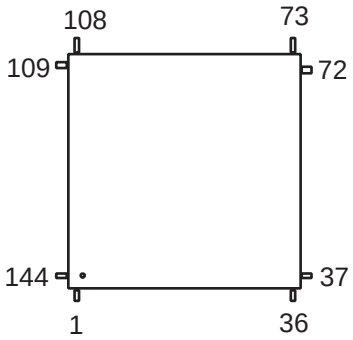
A1	PB26	C6	PB11	F1	VDDPLL	H6	NRST
A2	PB24	C7	PB8	F2	GNDPLL	H7	PA19
A3	PB22	C8	PB4	F3	PB30	H8	PA4
A4	PB14	C9	PB0	F4	PB29	H9	PA6
A5	PC0	C10	PA25	F5	GND	H10	PA22
A6	PB9	D1	PA5	F6	GND	J1	VBUS
A7	PB6	D2	PA0	F7	VDDIO	J2	DHSDP
A8	PB2	D3	PA1	F8	PB13	J3	DHSDM
A9	PA28	D4	VDDCORE	F9	PB17	J4	JTAGSEL
A10	PA26	D5	VDDIO	F10	PB18	J5	XIN32
B1	PA11	D6	VDDCORE	G1	XOUT	J6	VDDIN
B2	PB25	D7	VDDCORE	G2	VDDUTMI	J7	PA23
B3	PB23	D8	PB5	G3	PB31	J8	PA24
B4	PA10	D9	PB1	G4	GNDBU	J9	PB16
B5	PA9	D10	PA21	G5	PB27	J10	PA16
B6	PB10	E1	PB28	G6	PA18	K1	VBG
B7	PB7	E2	PA7	G7	PA20	K2	DFSDP
B8	PB3	E3	PA8	G8	PA3	K3	DFSDM
B9	PA29	E4	VDDCORE	G9	PA2	K4	VDDCORE
B10	PA27	E5	GND	G10	PB12	K5	XOUT32
C1	PA12	E6	GND	H1	XIN	K6	VDDOUT
C2	PA14	E7	VDDIO	H2	GNDUTMI	K7	VDDANA
C3	PA13	E8	PB19	H3	TST	K8	GNDANA
C4	PA17	E9	PB20	H4	VDDBU	K9	ADVREF
C5	PA15	E10	PB21	H5	FWUP	K10	PB15

4.2 SAM3X4/8E Package and Pinout

The SAM3X4/8E is available in 144-lead LQFP and 144-ball LFBGA packages.

4.2.1 144-lead LQFP Package Outline

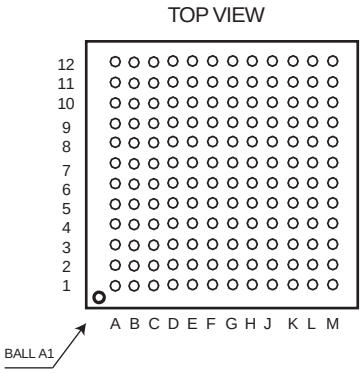
Figure 4-3. Orientation of the 144-lead LQFP Package



4.2.2 144-ball LFBGA Package Outline

The 144-ball LFBGA package has a 0.8 mm ball pitch and respects Green Standards. Its dimensions are 10 x 10 x 1.4 mm.

Figure 4-4. Orientation of the 144-ball LFBGA Package



4.2.3 144-lead LQFP Pinout

Table 4-3. 144-lead LQFP SAM3X4/8E Pinout

1	PB26	37	DHSDP	73	VDDANA	109	PA26
2	PA9	38	DHSDM	74	GNDANA	110	PA27
3	PA10	39	VBUS	75	ADVREF	111	PA28
4	PA11	40	VBG	76	PB15	112	PA29
5	PA12	41	VDDUTMI	77	PB16	113	PB0
6	PA13	42	DFSDP	78	PA16	114	PB1
7	PA14	43	DFSDM	79	PA24	115	PB2
8	PA15	44	GNDUTMI	80	PA23	116	PC4
9	PA17	45	VDDCORE	81	PA22	117	PC10
10	VDDCORE	46	JTAGSEL	82	PA6	118	PB3
11	VDDIO	47	NRSTB	83	PA4	119	PB4
12	GND	48	XIN32	84	PA3	120	PB5
13	PD0	49	XOUT32	85	PA2	121	PB6
14	PD1	50	SHDN	86	PB12	122	PB7
15	PD2	51	TST	87	PB13	123	PB8
16	PD3	52	VDDBU	88	PB17	124	VDDCORE
17	PD4	53	FWUP	89	PB18	125	VDDIO
18	PD5	54	GNDBU	90	PB19	126	GND
19	PD6	55	PC1	91	PB20	127	PB9
20	PD7	56	VDDOUT	92	PB21	128	PB10
21	PD8	57	VDDIN	93	PC11	129	PB11
22	PD9	58	GND	94	PC12	130	PC0
23	PA0	59	PC2	95	PC13	131	PC20
24	PA1	60	PC3	96	PC14	132	PC21
25	PA5	61	VDDCORE	97	PC15	133	PC22
26	PA7	62	VDDIO	98	PC16	134	PC23
27	PA8	63	PC5	99	PC17	135	PC24
28	PB28	64	PC6	100	PC18	136	PC25
29	PB29	65	PC7	101	PC19	137	PC26
30	PB30	66	PC8	102	PC29	138	PC27
31	PB31	67	PC9	103	PC30	139	PC28
32	PD10	68	PB27	104	VDDCORE	140	PB14
33	GNDPLL	69	NRST	105	VDDIO	141	PB22
34	VDDPLL	70	PA18	106	GND	142	PB23
35	XOUT	71	PA19	107	PA21	143	PB24
36	XIN	72	PA20	108	PA25	144	PB25

4.2.4 144-ball LFBGA Pinout

Table 4-4. 144-ball LFBGA SAM3X4/8E Pinout

A1	PA9	D1	PA17	G1	PA5	K1	VDDCORE
A2	PB23	D2	PD0	G2	PA7	K2	GNDUTMI
A3	PB14	D3	PA11	G3	PA8	K3	VDDPLL
A4	PC26	D4	PA15	G4	PA1	K4	NRSTB
A5	PC24	D5	PA14	G5	GND	K5	SHDN
A6	PC20	D6	PC27	G6	GND	K6	PC3
A7	PB10	D7	PC25	G7	GND	K7	PC6
A8	PB6	D8	VDDIO	G8	PC16	K8	PC7
A9	PB4	D9	PB5	G9	PC15	K9	PA18
A10	PC4	D10	PB0	G10	PC13	K10	PA23
A11	PA28	D11	PC30	G11	PB13	K11	PA16
A12	PA27	D12	PC19	G12	PB18	K12	PA24
B1	PA10	E1	PD1	H1	XOUT	L1	DHSDP
B2	PB26	E2	PD2	H2	PB30	L2	DHSDM
B3	PB24	E3	PD3	H3	PB28	L3	VDDUTMI
B4	PC28	E4	PD4	H4	PB29	L4	JTAGSEL
B5	PC23	E5	PD5	H5	VDDBU	L5	GNDBU
B6	PC0	E6	VDDCORE	H6	VDDCORE	L6	PC1
B7	PB9	E7	VDDCORE	H7	VDDIO	L7	PC2
B8	PB8	E8	VDDCORE	H8	PC12	L8	PC5
B9	PB3	E9	PB1	H9	PC11	L9	PC9
B10	PB2	E10	PC18	H10	PA3	L10	PA20
B11	PA26	E11	PB19	H11	PB12	L11	VDDANA
B12	PA25	E12	PB21	H12	PA2	L12	PB16
C1	PA13	F1	PD8	J1	XIN	M1	DFSDP
C2	PA12	F2	PD6	J2	GNDPLL	M2	DFSDM
C3	PB25	F3	PD9	J3	PD10	M3	VBG
C4	PB22	F4	PA0	J4	PB31	M4	VBUS
C5	PC22	F5	PD7	J5	TST	M5	XIN32
C6	PC21	F6	GND	J6	FWUP	M6	XOUT32
C7	PB11	F7	GND	J7	PB27	M7	VDDOUT
C8	PB7	F8	VDDIO	J8	NRST	M8	VDDIN
C9	PC10	F9	PC17	J9	PA19	M9	PC8
C10	PA29	F10	PC14	J10	PA22	M10	GNDANA
C11	PA21	F11	PB20	J11	PA4	M11	ADVREF
C12	PC29	F12	PB17	J12	PA6	M12	PB15

5. Power Considerations

5.1 Power Supplies

The SAM3X/A series product has several types of power supply pins:

- VDDCORE pins: Power the core, the embedded memories and the peripherals; voltage ranges from 1.62V to 1.95V.
- VDDIO pins: Power the peripherals I/O lines; voltage ranges from 1.62V to 3.6V.
- VDDIN pin: Powers the voltage regulator
- VDDOUT pin: Output of the voltage regulator
- VDDBU pin: Powers the Slow Clock oscillator and a part of the System Controller; voltage ranges from 1.62V to 3.6V. VDDBU must be supplied before or at the same time as VDDIO and VDDCORE.
- VDDPLL pin: Powers the PLL A, UPLL and 3–20 MHz Oscillator; voltage ranges from 1.62V to 1.95V.
- VDDUTMI pin: Powers the UTMI+ interface; voltage ranges from 3.0V to 3.6V, 3.3V nominal.
- VDDANA pin: Powers the ADC and DAC cells; voltage ranges from 2.0V to 3.6V.

Ground pins GND are common to VDDCORE and VDDIO pins power supplies.

Separated ground pins are provided for VDDBU, VDDPLL, VDDUTMI and VDDANA. These ground pins are respectively GNDBU, GNDPLL, GNDUTMI and GNDANA.

5.2 Power-up Considerations

5.2.1 VDDIO Versus VDDCORE

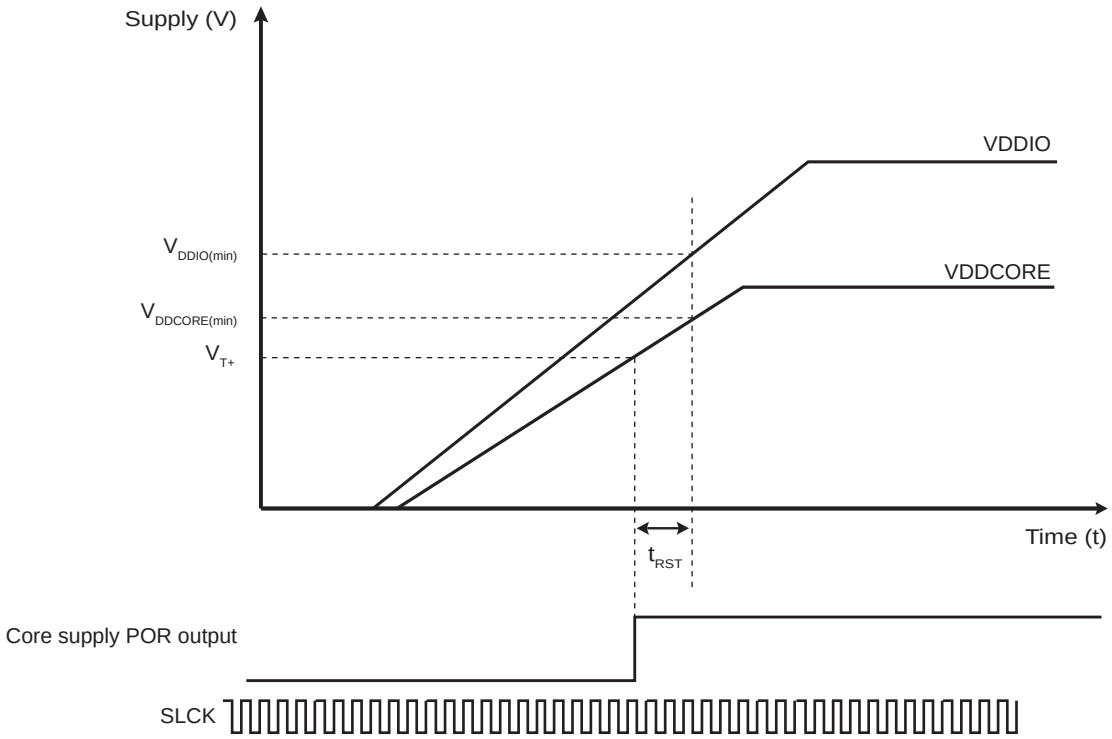
V_{DDIO} must always be higher than or equal to V_{DDCORE}.

V_{DDIO} must reach its minimum operating voltage (1.60 V) before V_{DDCORE} has reached $V_{DDCORE(min)}$. The minimum slope for V_{DDCORE} is defined by ($V_{DDCORE(min)}$ - V_{T+}) / t_{RST}.

If V_{DDCORE} rises at the same time as V_{DDIO}, the V_{DDIO} rising slope must be higher than or equal to 5 V/ms.

If VDDCORE is powered by the internal regulator, all power-up considerations are met.

Figure 5-1. VDDCORE and VDDIO Constraints at Startup



5.2.2 VDDIO Versus VDDIN

At power-up, V_{DDIO} needs to reach 0.6 V before V_{DDIN} reaches 1.0 V.

VDDIO voltage needs to be equal to or below (VDDIN voltage + 0.5 V).

5.3 Voltage Regulator

The SAM3X/A series embeds a voltage regulator that is managed by the Supply Controller.

This internal regulator is intended to supply the internal core of SAM3X/A series but can be used to supply other parts in the application. It features two different operating modes:

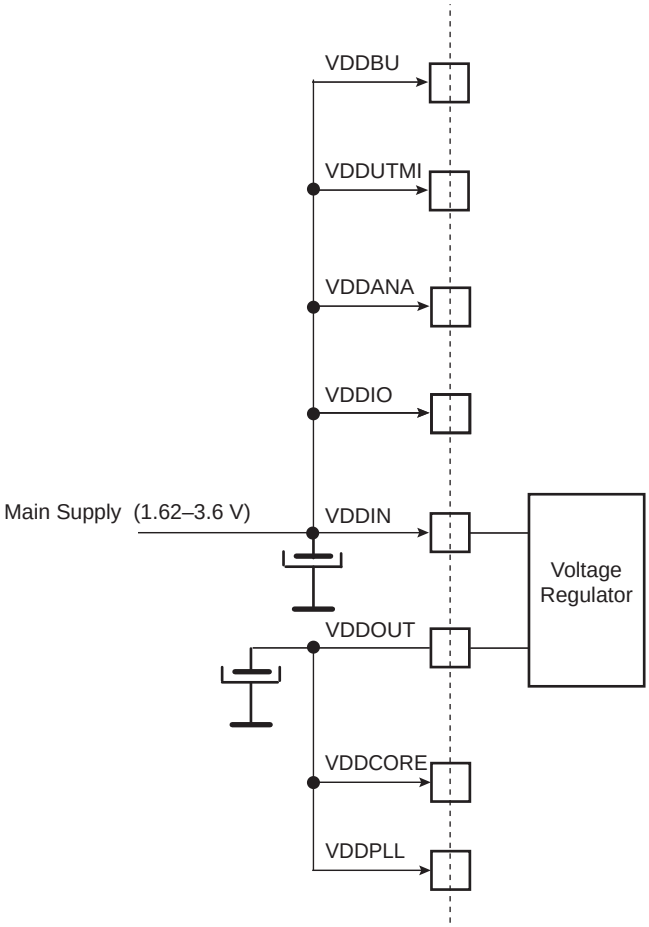
- In Normal mode, the voltage regulator consumes less than 700 µA static current and draws 150 mA of output current. Internal adaptive biasing adjusts the regulator quiescent current depending on the required load current. In Wait Mode or when the output current is low, quiescent current is only 7 µA.
- In Shutdown mode, the voltage regulator consumes less than 1 µA while its output is driven internally to GND. The default output voltage is 1.80V and the startup time to reach Normal mode is inferior to 400 µs.

For adequate input and output power supply decoupling/bypassing, refer to Table 45-3 ”1.8V Voltage Regulator Characteristics”.

5.4 Typical Powering Schematics

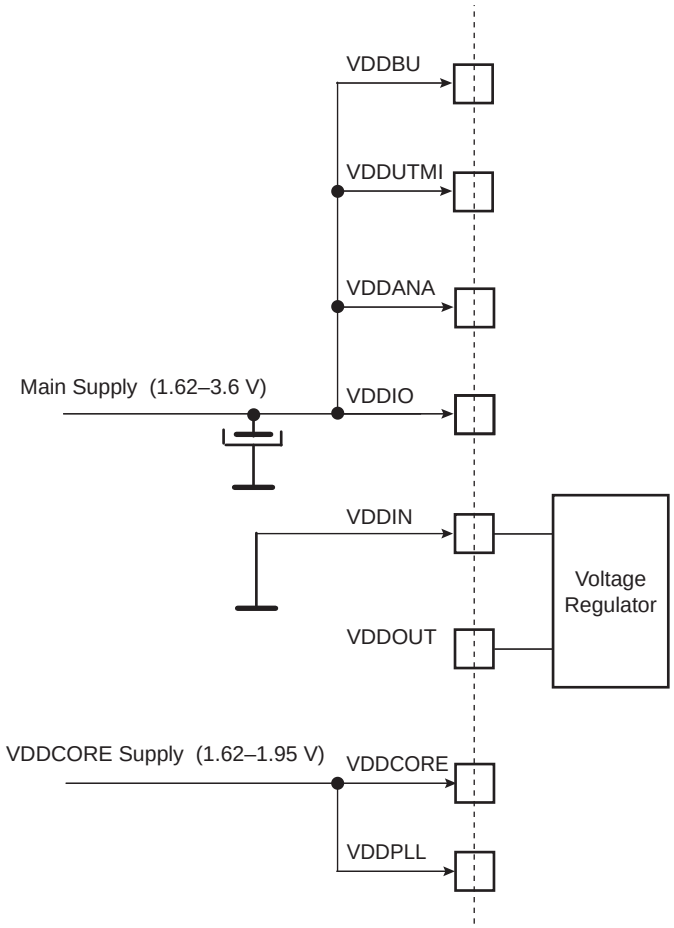
The SAM3X/A series supports a 1.62–3.6 V single supply mode. The internal regulator input connected to the source and its output feeds VDDCORE. Figure 5-2 shows the power schematics.

Figure 5-2. Single Supply



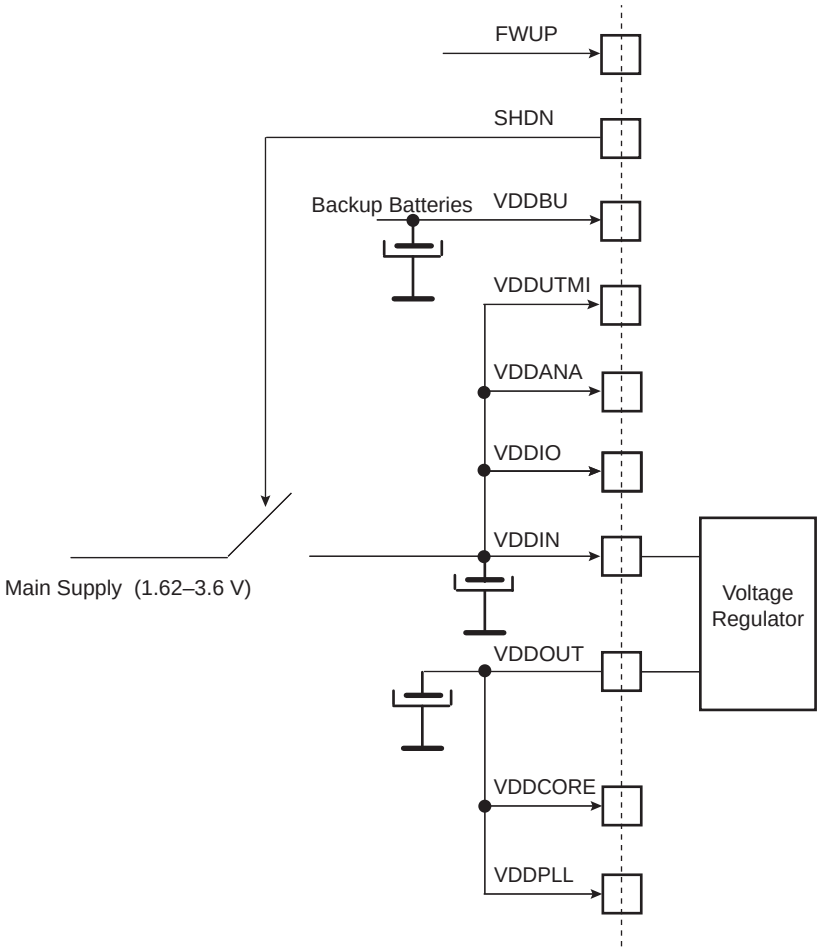
Note: Restrictions
For USB, VDDUTMI needs to be greater than 3.0V.
For ADC, VDDANA needs to be greater than 2.0V.
For DAC, VDDANA needs to be greater than 2.4V.

Figure 5-3. Core Externally Supplied



Note: Restrictions
For USB, VDDUTMI needs to be greater than 3.0V.
For ADC, VDDANA needs to be greater than 2.0V.
For DAC, VDDANA needs to be greater than 2.4V.

Figure 5-4. Backup Batteries Used



Note: 1. Restrictions
For USB, VDDUTMI needs to be greater than 3.0V.
For ADC, VDDANA needs to be greater than 2.0V.
For DAC, VDDANA needs to be greater than 2.4V.
2. VDDUTMI and VDDANA cannot be left unpowered.

5.5 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.

5.6 Low Power Modes

The SAM3X/A devices provide the following low-power modes: Backup, Wait, and Sleep.

5.6.1 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 (< 0.5 ms).

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.

Backup mode is based on the Cortex-M3 deep-sleep mode with the voltage regulator disabled.

The SAM3X/A series can be awakened from this mode through the Force Wake-up pin (FWUP), and Wake-up input pins WKUP0–15, Supply Monitor, RTT or RTC wake-up event. Current consumption is 2.5 µA typical on VDDBU.

Backup mode can be entered by using the WFE instruction.

The procedure to enter Backup mode using the WFE instruction is the following:

1. Write a 1 to the SLEEPDEEP bit in the Cortex-M3 processor System Control Register (SCR) (refer to Section 10.21.7 “System Control Register”).
2. Execute the WFE instruction of the processor.

Exit from Backup mode happens if one of the following enable wake-up events occurs:

- Low level, configurable debouncing on FWUP pin
- Level transition, configurable debouncing on pins WKUPEN0–15
- SM alarm
- RTC alarm
- RTT alarm

5.6.2 Wait Mode

The purpose of the 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.

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 via Wait for Event (WFE) instructions with LPM = 1 (Low Power Mode bit in PMC_FSMR). The Cortex-M3 is able to handle external events or internal events in order to wake-up the core (WFE). This is done by configuring the external lines WKUP0–15 as fast startup wake-up pins (refer to Section 5.8 “Fast Startup”). RTC or RTT Alarm and USB wake-up events can be used to wake up the CPU (exit from WFE).

Current consumption in Wait mode is typically 23 µA for total current consumption if the internal voltage regulator is used or 15 µA if an external regulator is used.

The procedure to enter Wait mode is the following:

1. Select the 4/8/12 MHz Fast RC Oscillator as Main Clock.
2. Set the LPM bit in the PMC Fast Startup Mode Register (PMC_FSMR).
3. Execute the WFE instruction of the processor.

Note: Internal Main clock resynchronization cycles are necessary between the writing of MOSCRCEN bit and the effective entry in Wait mode. Depending on the user application, Waiting for MOSCRCEN bit to be cleared is recommended to ensure that the core will not execute undesired instructions.

5.6.3 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.

This mode is entered via Wait for Interrupt (WFI) or WFE instructions with LPM = 0 in PMC_FSMR.

The processor can be awakened from an interrupt if WFI instruction of the Cortex-M3 is used, or from an event if the WFE instruction is used to enter this mode.

5.6.4 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 shows a summary of the configurations of the low power modes.

Table 5-1. Low Power Mode Configuration Summary

Mode	VDDBU Region (1)	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 (2) (3)	Wake-up Time (4)
Backup Mode	ON	OFF SHDN = 0	OFF (not powered)	WFE + SLEEPDEEP = 1	FWUP pin Pins WKUP0–15 BOD alarm RTC alarm RTT alarm	Reset	Previous state saved	PIOA & PIOB & PIOC & PIOD & PIOE & PIOF Inputs with pull-ups	2.5 µA typ (5)	< 0.5 ms
Wait Mode	ON	ON SHDN = 1	Powered (not clocked)	WFE + SLEEPDEEP = 0 + LPM = 1	Any event from Fast Startup: - through pins WKUP0–15 - RTC alarm - RTT alarm - USB wake-up	Clocked back	Previous state saved	Unchanged	18.4 µA/26.6 µA (6)	< 10 µs
Sleep Mode	ON	ON SHDN = 1	Powered (7) (not 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 Startup: - through pins WKUP0–15 - RTC alarm - RTT alarm - USB wake-up	Clocked back	Previous state saved	Unchanged	(7)	(7)

Notes:
1. SUPC, 32 kHz Oscillator, RTC, RTT, GPBR, POR.
2. The external loads on PIOs are not taken into account in the calculation.
3. BOD current consumption is not included.
4. When considering the 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 startup 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.
5. Current consumption on VDDBU.
6. 18.4 µA on VDDCORE, 26.6 µA for total current consumption (using internal voltage regulator).
7. Depends on MCK frequency. In this mode, the core is supplied and not clocked but some peripherals can be clocked.

5.7 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. See Figure 16-7, "Wake Up Source" on page 277.

5.8 Fast Startup

The SAM3X/A series allows the processor to restart in a few microseconds while the processor is in wait mode. A fast start up can occur upon detection of a low level on one of the 19 wake-up inputs (WKUP0–15 + RTC + RTT + USB).

The fast restart circuitry (shown in Figure 28-4, "Fast Startup Circuitry" on page 530) is fully asynchronous and provides a fast startup signal to the Power Management Controller. As soon as the fast startup 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 by default and reenables the processor clock.

6. Input/Output Lines

The SAM3X/A has different kinds of input/output (I/O) lines, such as general purpose I/Os (GPIO) and system I/Os. GPIOs can have alternate functions thanks to multiplexing capabilities of the PIO controllers. The same PIO line can be used whether in IO mode or by the multiplexed peripheral. System I/Os include pins such as test pins, oscillators, erase or analog inputs.

With a few exceptions, the I/Os have input Schmitt triggers. Refer to the footnotes associated with PIOA to PIOF on page 12, at the end of Table 3-1 “Signal Description List”.

6.1 General Purpose I/O Lines (GPIO)

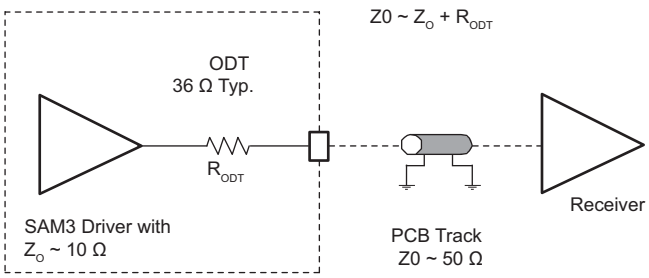
GPIO Lines are managed by PIO Controllers. All I/Os have several input or output modes such as pull-up, input Schmitt triggers, multi-drive (open-drain), glitch filters, debouncing or input change interrupt. Programming of these modes is performed independently for each I/O line through the PIO controller user interface. For more details, refer to Section 31. “Parallel Input/Output Controller (PIO)”.

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

The SAM3X/A embeds high speed pads able to handle up to 65 MHz for HSMCI and SPI clock lines and 45 MHz on other lines. See Section 45.10 “AC Characteristics” for more details. Typical pull-up value is 100 kΩ for all I/Os.

Each I/O line also embeds an ODT (On-Die Termination) (see Figure 6-1). ODT consists of an internal series resistor termination scheme for impedance matching between the driver output (SAM3) and the PCB track 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 reducing signal integrity issues.

Figure 6-1. On-Die Termination



6.2 System I/O Lines

Table 6-1 lists the SAM3X/A system I/O lines shared with PIO lines. These pins are software configurable as general purpose I/O or system pins. At startup, the default function of these pins is always used.

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

CCFG_SYSIO Bit No.	Peripheral	Default Function After Reset	Other Function	Constraints for Normal Start	Configuration
12	–	ERASE	PC0	Low Level at startup(1)	In Matrix User Interface Registers (refer to System I/O Configuration Register in Section 21. "Bus Matrix (MATRIX)")
–	A	TCK/SWCLK	PB28	–	In PIO Controller
–	A	TDI	PB29	–	
–	A	TDO/TRACESWO	PB30	–	
–	A	TMS/SWDIO	PB31	–	

Notes: 1. If PC0 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 PC0 into PIO mode.

6.2.1 Serial Wire JTAG Debug Port (SWJ-DP) Pins

The SWJ-DP pins are TCK/SWCLK, TMS/SWDIO, TDO/TRACESWO, TDI and commonly provided on a standard 20-pin JTAG connector defined by ARM. For more details about voltage reference and reset state, refer to Table 3-1 "Signal Description List".

At startup, SWJ-DP pins are configured in SWJ-DP mode to allow connection with debugging probe. Please refer to Section 11. "Debug and Test Features".

SWJ-DP pins can be used as standard I/Os to provide users with 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 Section 11. "Debug and Test Features".

All JTAG signals are supplied with VDDIO except JTAGSEL, supplied by VDDBU.

6.3 Test Pin

The TST pin is used for JTAG Boundary Scan Manufacturing Test or Fast Flash programming mode of the SAM3X/A series. The TST pin integrates a permanent pull-down resistor of about 15 kΩ to GND, so that it can be left unconnected for normal operations. To enter fast programming mode, see Section 19. "Fast Flash Programming Interface (FFPI)". For more information on the manufacturing and test mode, refer to Section 11. "Debug and Test Features".

6.4 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 kΩ.

6.5 NRSTB Pin

The NRSTB pin is input only and enables asynchronous reset of the SAM3X/A series when asserted low. The NRSTB pin integrates a permanent pull-up resistor of about 15 kΩ. This allows connection of a simple push button on the NRSTB pin as a system-user reset. In all modes, this pin will reset the chip including the Backup region (RTC, RTT and Supply Controller). It reacts as the Power-on reset. It can be used as an external system reset source. In harsh environments, it is recommended to add an external capacitor (10 nF) between NRSTB and VDDBU. (For filtering values, refer to "I/O Characteristics" in Section 45. "Electrical Characteristics".)

It embeds an anti-glitch filter.

6.6 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). The ERASE pin and the ROM code ensure an in-situ reprogrammability of the Flash content without the use of a debug tool. When the security bit is activated, the ERASE pin provides the capability to reprogram the Flash content. It integrates a pull-down resistor of about 100 kΩ to GND, so that it can be left unconnected for normal operations.

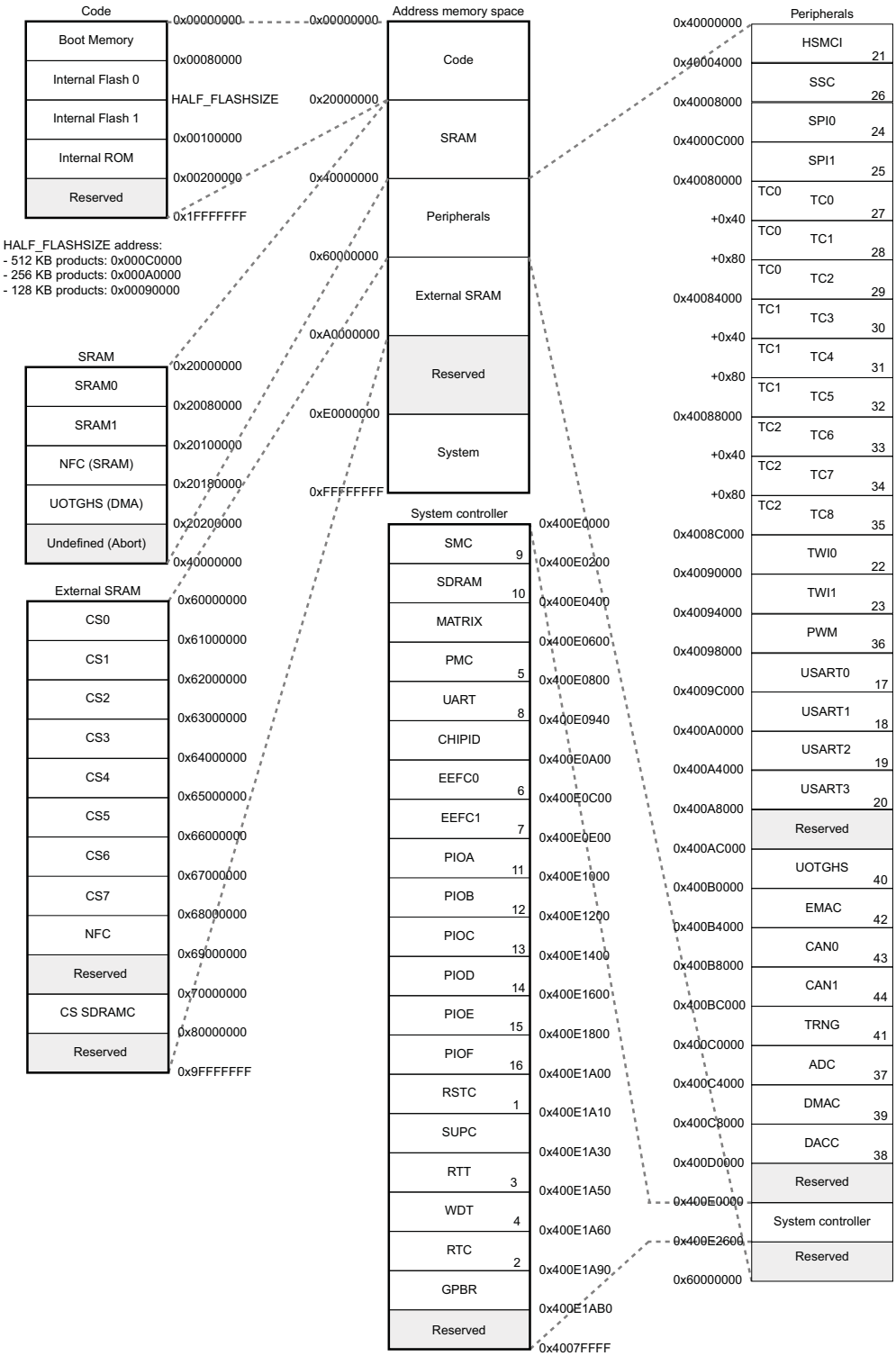
This pin is debounced by SCLK to improve the glitch tolerance. When the ERASE pin is tied high during less than 100 ms, it is not taken into account. The pin must be tied high during more than 220 ms to perform a Flash erase operation.

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, the startup level of this pin must be low to prevent unwanted erasing. Please refer to Section 9.3 "Peripheral Signal Multiplexing on I/O Lines". Also, if the ERASE pin is used as a standard I/O output, asserting the pin to high does not erase the Flash.

7. Memories

7.1 Product Mapping

Figure 7-1. SAM3X/A Product Mapping



7.2 Embedded Memories

7.2.1 Internal SRAM

Table 7-1 shows the embedded high-speed SRAM for the various devices.

Table 7-1. Embedded High-speed SRAM per Device

Device	Pin Count	SRAM0 (KB)	SRAM1 (KB)	NFC SRAM (KB)	Total SRAM (KB)
SAM3X8E	144	64	32	4	100
SAM3X8H(1)					
SAM3A8C	100	64	32	–	96
SAM3X8C					
SAM3X4E	144	32	32	4	68
SAM3A4C	100	32	32	–	64
SAM3X4C					

Note: 1. This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

SRAM0 is accessible over the system Cortex-M3 bus at address 0x2000 0000 and SRAM1 at address 0x2008 0000. The user can see the SRAM as contiguous thanks to mirror effect, giving 0x2007 0000 - 0x2008 7FFF for SAM3X/A8, and 0x2007 8000 - 0x2008 7FFF for SAM3X/A4.

SRAM0 and SRAM1 are in the bit band region. The bit band alias region is mapped from 0x2200 0000 to 0x23FF FFFF.

The NAND Flash Controller (NFC) embeds 4224 bytes of internal SRAM. If the NFC is not used, these 4224 Kbytes can be used as general-purpose SRAM. It can be seen at address 0x2010 0000.

7.2.2 Internal ROM

The SAM3X/A series product embeds an Internal ROM, which contains the SAM-BA® and FFPI program.

At any time, the ROM is mapped at address 0x0010 0000.

7.2.3 Embedded Flash

7.2.3.1 Flash Overview

Table 7-2 shows the Flash organization for the various devices.

Table 7-2. Embedded Flash Memory Organization per Device

Device	Flash Size (Kbytes)	Number of Banks	Number of Pages	Page Size (bytes)	Plane
SAM3A8	512	2	1024	256	Dual
SAM3X8	512	2	1024	256	Dual
SAM3A4	256	2	512	256	Dual
SAM3X4	256	2	512	256	Dual

The Flash contains a 128-byte write buffer, accessible through a 32-bit interface.

7.2.3.2 Flash Power Supply

The Flash is supplied by VDDCORE.

7.2.3.3 Enhanced Embedded Flash Controller

The Enhanced Embedded Flash Controller (EEFC) 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 within the Memory Controller on the APB.

The EEFC ensures the interface of the Flash block with the 32-bit internal bus. Its 128-bit wide memory interface increases performance.

The user can choose between high performance or lower current consumption by selecting either 128-bit or 64-bit access. 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.

7.2.3.4 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 7-3. Number of Lock Bits

Product	Number of Lock Bits	Lock Region Size
SAM3X8	32	16 Kbytes (64 pages)
SAM3A8		
SAM3X4	16	16 Kbytes (64 pages)
SAM3A4		

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 "Set Lock Bit" command enables the protection. The "Clear Lock Bit" command unlocks the lock region.

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

7.2.3.5 Security Bit Feature

The SAM3X/A series 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, either through the ICE interface or through the Fast Flash Programming Interface (FFPI), is forbidden. This ensures the confidentiality of the code programmed in the Flash.

This security bit can only be enabled through the "Set General Purpose NVM Bit 0" command of the EEFC0 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 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.

7.2.3.6 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.

7.2.3.7 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.

7.2.3.8 Fast Flash Programming Interface (FFPI)

The FFPI allows device programming through 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 FFPI is enabled and the Fast Programming Mode is entered when TST, PA0, PA1 are set to high, PA2 and PA3 are set to low and NRST is toggled from 0 to 1.

The table below shows the signal assignment of the PIO lines in FFPI mode

Table 7-4. FFPI PIO Assignment

FFPI Signal	PIO Used
PGMNCMD	PA0
PGMRDY	PA1
PGMNOE	PA2
PGMNVALID	PA3
PGMM[0]	PA4
PGMM[1]	PA5
PGMM[2]	PA6
PGMM[3]	PA7
PGMD[0]	PA8
PGMD[1]	PA9
PGMD[2]	PA10
PGMD[3]	PA11
PGMD[4]	PA12
PGMD[5]	PA13
PGMD[6]	PA14
PGMD[7]	PA15
PGMD[8]	PA16
PGMD[9]	PA17
PGMD[10]	PA18
PGMD[11]	PA19
PGMD[12]	PA20
PGMD[13]	PA21
PGMD[14]	PA22
PGMD[15]	PA23

7.2.3.9 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 and USB.

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.

7.2.3.10 GPNVM Bits

The SAM3X/A series features three GPNVM bits that can be cleared or set respectively through the “Clear GPNVM Bit” and “Set GPNVM Bit” commands of the EEFC0 User Interface.

There is no GPNVM bit on Flash 1.

The GPNVM0 is the security bit.

The GPNVM1 is used to select the boot mode (boot always at 0x00) on ROM or Flash.

The GPNVM2 is used only to swap the Flash 0 and Flash 1. If GPNVM2 is ENABLE, the Flash 1 is mapped at address 0x0008_0000 (Flash 1 and Flash 0 are continuous). If GPNVM2 is DISABLE, the Flash 0 is mapped at address 0x0008_0000 (Flash 0 and Flash 1 are continuous).

Table 7-5. General Purpose Non-volatile Memory Bits

GPNVM Bit[#]	Function
0	Security bit
1	Boot mode selection
2	Flash selection (Flash 0 or Flash 1)

7.2.4 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 (GPNVM1) bit is used to boot either on the ROM (default) or from the Flash.

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

GPNVM2 enables to select if Flash 0 or Flash 1 is used for the boot.

Setting GPNVM bit 2 selects the boot from Flash 1, clearing it selects the boot from Flash 0.

7.3 External Memories

The 144-pin SAM3X and 217-pin SAM3X8H[(1)] feature one External Memory Bus to offer interface to a wide range of external memories and to any parallel peripheral.

Note: 1. This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

7.3.1 External Memory Bus

- Integrates Four External Memory Controllers:
  - Static Memory Controller
  - NAND Flash Controller
  - SLC NAND Flash ECC Controller
  - Single Data Rate Synchronous Dynamic Random Access Memory (SDR-SDRAM)
- Up to 24-bit Address Bus (up to 16 Mbytes linear per chip select)
- Up to 8 chip selects, Configurable Assignment

7.3.2 Static Memory Controller

- 8- or 16-bit Data Bus
- Multiple Access Modes supported
  - Byte Write or Byte Select Lines
  - Asynchronous read in Page Mode supported (4- up to 32-byte page size)

- Multiple device adaptability
  - Control signals programmable setup, pulse and hold time for each Memory Bank
- Multiple Wait State Management
  - Programmable Wait State Generation
  - External Wait Request
  - Programmable Data Float Time
- Slow Clock mode supported

7.3.3 NAND Flash Controller

- Handles automatic Read/write transfer through 4224 bytes SRAM buffer
- DMA support
- Supports SLC NAND Flash technology
- Programmable timing on a per chip select basis
- Programmable Flash Data width 8-bit or 16-bit

7.3.4 NAND Flash Error Corrected Code Controller

- Integrated in the NAND Flash Controller
- Single bit error correction and 2-bit Random detection.
- Automatic Hamming Code Calculation while writing
  - ECC value available in a register
- Automatic Hamming Code Calculation while reading
  - Error Report, including error flag, correctable error flag and word address being detected erroneous
  - Support 8- or 16-bit NAND Flash devices with 512-, 1024-, 2048- or 4096-byte pages

7.3.5 SDR-SDRAM Controller (217-pin SAM3X8H[1] only)

- Numerous configurations supported
  - 2K, 4K, 8K Row Address Memory Parts
  - SDRAM with two or four Internal Banks
  - SDRAM with 16-bit Data Path
- Programming facilities
  - Word, half-word, byte access
  - Automatic page break when Memory Boundary has been reached
  - Multibank Ping-pong Access
  - Timing parameters specified by software
  - Automatic refresh operation, refresh rate is programmable
- Energy-saving capabilities
  - Self-refresh, and Low-power Modes supported
- Error detection
  - Refresh Error Interrupt
- SDRAM Power-up Initialization by software
- Latency is set to two clocks (CAS Latency of 1, 3 Not Supported)
- Auto Precharge Command not used
- Mobile SDRAM supported (except for low-power extended mode and deep power-down mode)

Note: 1. This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

8. System Controller

The System Controller is a set of peripherals which allow handling of key elements of the system such as but not limited to power, resets, clocks, time, interrupts, and watchdog.

The System Controller User Interface also embeds the registers allowing to configure the Matrix and a set of registers configuring the SDR-SDRAM chip select assignment.

8.1 System Controller and Peripherals Mapping

Please refer to Figure 7-1 “SAM3X/A Product Mapping” on page 31 .

All the peripherals are in the bit band region and are mapped in the bit band alias region.

8.2 Power-on-Reset, Brownout and Supply Monitor

The SAM3X/A embeds three features to monitor, warn and/or reset the chip:

- Power-on-Reset on VDDBU
- Brownout Detector on VDDCORE
- Supply Monitor on VDDUTMI

8.2.1 Power-on-Reset on VDDBU

The Power-on-Reset monitors VDDBU. It is always activated and monitors voltage at start up but also during power down. If VDDBU goes below the threshold voltage, the entire chip is reset. For more information, refer to Section 45. “Electrical Characteristics”.

8.2.2 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 Section 16. “Supply Controller (SUPC)” and Section 45. “Electrical Characteristics”.

8.2.3 Supply Monitor on VDDUTMI

The Supply Monitor monitors VDDUTMI. It is not active by default. It can be activated by software and is fully programmable with 16 steps for the threshold (between 1.9V 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 Section 16. “Supply Controller (SUPC)” and Section 45. “Electrical Characteristics”.

9. Peripherals

9.1 Peripheral Identifiers

Table 9-1 defines the Peripheral Identifiers of the SAM3X/A series. A peripheral identifier is required for the control of the peripheral interrupt with the Nested Vectored Interrupt Controller and for the control of the peripheral clock with the Power Management Controller.

Note that some peripherals are always clocked. Please refer to the table below.

Table 9-1. 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	WDG	X		Watchdog Timer
5	PMC	X		Power Management Controller
6	EEFC0	X		Enhanced Embedded Flash Controller 0
7	EEFC1	X		Enhanced Embedded Flash Controller 1
8	UART	X		Universal Asynchronous Receiver Transceiver
9	SMC_SDRAMC	X	X	Static Memory Controller / Synchronous Dynamic RAM Controller
10	SDRAMC	X		Synchronous Dynamic RAM Controller
11	PIOA	X	X	Parallel I/O Controller A
12	PIOB	X	X	Parallel I/O Controller B
13	PIOC	X	X	Parallel I/O Controller C
14	PIOD	X	X	Parallel I/O Controller D
15	PIOE	X	X	Parallel I/O Controller E
16	PIOF	X	X	Parallel I/O Controller F
17	USART0	X	X	Universal Synchronous Asynchronous Receiver Transmitter 0
18	USART1	X	X	Universal Synchronous Asynchronous Receiver Transmitter 1
19	USART2	X	X	Universal Synchronous Asynchronous Receiver Transmitter 2
20	USART3	X	X	Universal Synchronous Asynchronous Receiver Transmitter 3
21	HSMCI	X	X	High Speed Multimedia Card Interface
22	TWI0	X	X	Two-Wire Interface 0
23	TWI1	X	X	Two-Wire Interface 1
24	SPI0	X	X	Serial Peripheral Interface 0
25	SPI1	X	X	Serial Peripheral Interface 1
26	SSC	X	X	Synchronous Serial Controller
27	TC0	X	X	Timer Counter Channel 0
28	TC1	X	X	Timer Counter Channel 1
29	TC2	X	X	Timer Counter Channel 2

Table 9-1. Peripheral Identifiers (Continued)

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
30	TC3	X	X	Timer Counter Channel 3
31	TC4	X	X	Timer Counter Channel 4
32	TC5	X	X	Timer Counter Channel 5
33	TC6	X	X	Timer Counter Channel 6
34	TC7	X	X	Timer Counter Channel 7
35	TC8	X	X	Timer Counter Channel 8
36	PWM	X	X	Pulse Width Modulation Controller
37	ADC	X	X	ADC Controller
38	DACC	X	X	DAC Controller
39	DMAC	X	X	DMA Controller
40	UOTGHS	X	X	USB OTG High Speed
41	TRNG	X	X	True Random Number Generator
42	EMAC	X	X	Ethernet MAC
43	CAN0	X	X	CAN Controller 0
44	CAN1	X	X	CAN Controller 1

9.2 APB/AHB Bridge

The SAM3X/A series product embeds two separate APB/AHB bridges:

- a low speed bridge
- a high speed bridge

This architecture enables a concurrent access on both bridges.

SPI, SSC and HSMCI peripherals are on the high-speed bridge connected to DMAC with the internal FIFO for Channel buffering.

UART, ADC, TWI0–1, USART0–3, PWM, DAC and CAN peripherals are on the low-speed bridge and have dedicated channels for the Peripheral DMA Channels (PDC). Note that USART0–1 can be used with the DMA as well.

The peripherals on the high speed bridge are clocked by MCK. On the low-speed bridge, CAN controllers can be clocked at MCK divided by 2 or 4. Refer to Section 27. “Clock Generator”.

9.3 Peripheral Signal Multiplexing on I/O Lines

The SAM3X/A series product features up to six PIO controllers (PIOA, PIOB, PIOC, PIOD, PIOE, and PIOF) which multiplex the I/O lines of the peripheral set:

- 3 PIO controllers on SAM3A and 100-pin SAM3X
- 4 PIO controllers on 144-pin SAM3X
- 6 PIO controllers on 217-pin SAM3X8H(1)

Each PIO controller controls up to 32 lines. Each line can be assigned to one of two peripheral functions, A or B. The multiplexing tables in the following pages define how the I/O lines of the peripherals A and B are multiplexed on the PIO controllers.

Note that some output-only peripheral functions might be duplicated within the tables.

Note: 1. This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

9.3.1 PIO Controller A Multiplexing

Table 9-2. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Extra Function	Comments
PA0	CANTX0	PWML3		
PA1	CANRX0	PCK0	WKUP0(1)	
PA2	TIOA1	NANDRDY	AD0(2)	
PA3	TIOB1	PWMFI1	AD1/WKUP1(3)	
PA4	TCLK1	NWAIT	AD2(2)	
PA5	TIOA2	PWMFI0	WKUP2(1)	
PA6	TIOB2	NCS0	AD3(2)	
PA7	TCLK2	NCS1	WKUP3(1)	
PA8	URXD	PWMH0	WKUP4(1)	
PA9	UTXD	PWMH3		
PA10	RXD0	DATRG	WKUP5(1)	
PA11	TXD0	ADTRG	WKUP6(1)	
PA12	RXD1	PWML1	WKUP7(1)	
PA13	TXD1	PWMH2		
PA14	RTS1	TK		
PA15	CTS1	TF	WKUP8(1)	
PA16	SPCK1	TD	AD7(2)	
PA17	TWD0	SPCK0		
PA18	TWCK0	A20	WKUP9(1)	
PA19	MCCK	PWMH1		
PA20	MCCDA	PWML2		
PA21	MCDA0	PWML0		
PA22	MCDA1	TCLK3	AD4(2)	
PA23	MCDA2	TCLK4	AD5(2)	
PA24	MCDA3	PCK1	AD6(2)	
PA25	SPI0_MISO	A18		
PA26	SPI0_MOSI	A19		
PA27	SPI0_SPCK	A20	WKUP10(1)	
PA28	SPI0_NPCS0	PCK2	WKUP11(1)	
PA29	SPI0_NPCS1	NRD		
PA30	SPI0_NPCS2	PCK1		217 pins
PA31	SPI0_NPCS3	PCK2		217 pins

Notes: 1. WKUPx can be used if PIO controller defines the I/O line as "input".
2. To select this extra function, refer to Section 43.5.3 “Analog Inputs”.
3. Analog input has priority over WKUPx pin.

9.3.2 PIO Controller B Multiplexing

Table 9-3. Multiplexing on PIO Controller B (PIOB)

I/O Line	Peripheral A	Peripheral B	Extra Function	Comments
PB0	ETXCK/EREFCK(1)	TIOA3(2)		
PB1	ETXEN(1)	TIOB3(2)		
PB2	ETX0(1)	TIOA4(2)		
PB3	ETX1(1)	TIOB4(2)		
PB4	ECRSDV/ERXDV(1)	TIOA5(2)		
PB5	ERX0(1)	TIOB5(2)		
PB6	ERX1(1)	PWML4(2)		
PB7	ERXER(1)	PWML5(2)		
PB8	EMDC(1)	PWML6(2)		
PB9	EMDIO(1)	PWML7(2)		
PB10	UOTGVBOF	A18		
PB11	UOTGID	A19		
PB12	TWD1	PWMH0	AD8(3)	
PB13	TWCK1	PWMH1	AD9(3)	
PB14	CANTX1	PWMH2		
PB15	CANRX1	PWMH3	DAC0/WKUP12(4)	
PB16	TCLK5	PWML0	DAC1(5)	
PB17	RF	PWML1	AD10(3)	
PB18	RD	PWML2	AD11(3)	
PB19	RK	PWML3	AD12(3)	
PB20	TXD2	SPI0_NPCS1	AD13(3)	
PB21	RXD2	SPI0_NPCS2	AD14/WKUP13(4)	
PB22	RTS2	PCK0		
PB23	CTS2	SPI0_NPCS3	WKUP14(6)	
PB24	SCK2	NCS2		
PB25	RTS0	TIOA0		
PB26	CTS0	TCLK0	WKUP15(6)	
PB27	NCS3	TIOB0		
PB28	TCK/SWCLK			TCK after reset
PB29	TDI			TDI after reset
PB30	TDO/TRACESWO			TDO after reset
PB31	TMS/SWDIO			TMS after reset

Notes:
1. SAM3X only
2. SAM3A only
3. To select this extra function, refer to Section 43.5.3 “Analog Inputs”.
4. Analog input has priority over WKUPx pin.

5. DAC0 is selected when DACC_CHER.CH0 is set. DAC1 is selected when DACC_CHER.CH1 is set. See Section 44.7.3 “DACC Channel Enable Register”.
6. WKUPx can be used if PIO controller defines the I/O line as "input".

9.3.3 PIO Controller C Multiplexing

Table 9-4. Multiplexing on PIO Controller C (PIOC)

I/O Line	Peripheral A	Peripheral B	Extra Function	Comments
PC0			ERASE[1]	
PC1				144 and 217 pins
PC2	D0	PWML0		144 and 217 pins
PC3	D1	PWMH0		144 and 217 pins
PC4	D2	PWML1		144 and 217 pins
PC5	D3	PWMH1		144 and 217 pins
PC6	D4	PWML2		144 and 217 pins
PC7	D5	PWMH2		144 and 217 pins
PC8	D6	PWML3		144 and 217 pins
PC9	D7	PWMH3		144 and 217 pins
PC10	D8	ECRS		144 and 217 pins
PC11	D9	ERX2		144 and 217 pins
PC12	D10	ERX3		144 and 217 pins
PC13	D11	ECOL		144 and 217 pins
PC14	D12	ERXCK		144 and 217 pins
PC15	D13	ETX2		144 and 217 pins
PC16	D14	ETX3		144 and 217 pins
PC17	D15	ETXER		144 and 217 pins
PC18	NWR0/NWE	PWMH6		144 and 217 pins
PC19	NANDOE	PWMH5		144 and 217 pins
PC20	NANDWE	PWMH4		144 and 217 pins
PC21	A0/NBS0	PWML4		144 and 217 pins
PC22	A1	PWML5		144 and 217 pins
PC23	A2	PWML6		144 and 217 pins
PC24	A3	PWML7		144 and 217 pins
PC25	A4	TIOA6		144 and 217 pins
PC26	A5	TIOB6		144 and 217 pins
PC27	A6	TCLK6		144 and 217 pins
PC28	A7	TIOA7		144 and 217 pins
PC29	A8	TIOB7		144 and 217 pins
PC30	A9	TCLK7		144 and 217 pins

Notes: 1. To select this extra function, refer to Section 6.2 “System I/O Lines”.

9.3.4 PIO Controller D Multiplexing

Table 9-5. Multiplexing on PIO Controller D (PIOD)

I/O Line	Peripheral A	Peripheral B	Extra Function	Comments
PD0	A10	MCDA4		144 and 217 pins
PD1	A11	MCDA5		144 and 217 pins
PD2	A12	MCDA6		144 and 217 pins
PD3	A13	MCDA7		144 and 217 pins
PD4	A14	TXD3		144 and 217 pins
PD5	A15	RXD3		144 and 217 pins
PD6	A16/BA0	PWMFI2		144 and 217 pins
PD7	A17/BA1	TIOA8		144 and 217 pins
PD8	A21/NANDALE	TIOB8		144 and 217 pins
PD9	A22/NANDCLE	TCLK8		144 and 217 pins
PD10	NWR1/NBS1			144 and 217 pins
PD11	SDA10			217 pins
PD12	SDCS			217 pins
PD13	SDCKE			217 pins
PD14	SDWE			217 pins
PD15	RAS			217 pins
PD16	CAS			217 pins
PD17	A5			217 pins
PD18	A6			217 pins
PD19	A7			217 pins
PD20	A8			217 pins
PD21	A9			217 pins
PD22	A10			217 pins
PD23	A11			217 pins
PD24	A12			217 pins
PD25	A13			217 pins
PD26	A14			217 pins
PD27	A15			217 pins
PD28	A16/BA0			217 pins
PD29	A17/BA1			217 pins
PD30	A18			217 pins

9.3.5 PIO Controller E Multiplexing

Table 9-6. Multiplexing on PIO Controller E (PIOE)

I/O Line	Peripheral A	Peripheral B	Extra Function	Comments
PE0	A19			217 pins
PE1	A20			217 pins
PE2	A21/NANDALE			217 pins
PE3	A22/NANDCLE			217 pins
PE4	A23			217 pins
PE5	NCS4			217 pins
PE6	NCS5			217 pins
PE7				217 pins
PE8				217 pins
PE9	TIOA3			217 pins
PE10	TIOB3			217 pins
PE11	TIOA4			217 pins
PE12	TIOB4			217 pins
PE13	TIOA5			217 pins
PE14	TIOB5			217 pins
PE15	PWMH0			217 pins
PE16	PWMH1	SCK3		217 pins
PE17	PWML2			217 pins
PE18	PWML0	NCS6		217 pins
PE19	PWML4			217 pins
PE20	PWMH4	MCCDB		217 pins
PE21	PWML5			217 pins
PE22	PWMH5	MCDB0		217 pins
PE23	PWML6			217 pins
PE24	PWMH6	MCDB1		217 pins
PE25	PWML7			217 pins
PE26	PWMH7	MCDB2		217 pins
PE27	NCS7	MCDB3		217 pins
PE28	SPI1_MISO			217 pins
PE29	SPI1_MOSI			217 pins
PE30	SPI1_SPCK			217 pins
PE31	SPI1_NPCS0			217 pins

9.3.6 PIO Controller F Multiplexing

Table 9-7. Multiplexing on PIO Controller F (PIOF)

I/O Line	Peripheral A	Peripheral B	Extra Function	Comments
PF0	SPI1_NPCS1			217 pins
PF1	SPI1_NPCS2			217 pins
PF2	SPI1_NPCS3			217 pins
PF3	PWMH3			217 pins
PF4	CTS3			217 pins
PF5	RTS3			217 pins

10. ARM Cortex® M3 Processor

10.1 About this section

This section provides the information required for application and system-level software development. It does not provide information on debug components, features, or operation.

This material is for microcontroller software and hardware engineers, including those who have no experience of ARM products.

Note: The information in this section is reproduced from source material provided to Atmel by ARM Ltd. in terms of Atmel's license for the ARM Cortex™-M3 processor core. This information is copyright ARM Ltd., 2008 - 2009.

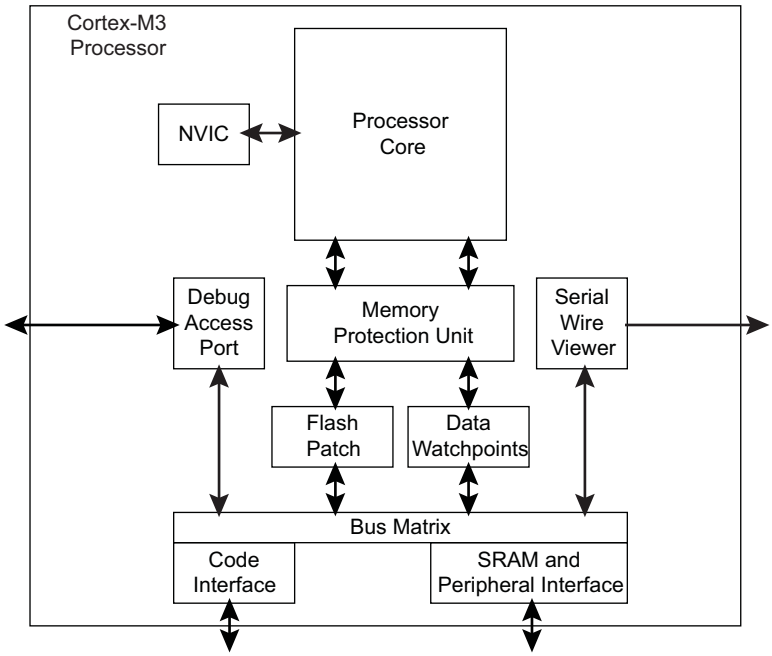
10.2 Embedded Characteristics

- Version 2.0
- Thumb-2 (ISA) subset consisting of all base Thumb-2 instructions, 16-bit and 32-bit.
- Harvard processor architecture enabling simultaneous instruction fetch with data load/store.
- Three-stage pipeline.
- Single cycle 32-bit multiply.
- Hardware divide.
- Thumb and Debug states.
- Handler and Thread modes.
- Low latency ISR entry and exit.
- SysTick Timer
  - 24-bit down counter
  - Self-reload capability
  - Flexible system timer
- Nested Vectored Interrupt Controller
  - Thirty maskable interrupts
  - Sixteen priority levels
  - Dynamic reprioritization of interrupts
  - Priority grouping
    selection of preempting interrupt levels and non preempting interrupt levels.
  - Support for tail-chaining and late arrival of interrupts.
    back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
  - Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead.

10.3 About the Cortex-M3 processor and core peripherals

- The Cortex-M3 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
- platform security, with integrated *memory protection unit* (MPU).

Figure 10-1. Typical Cortex-M3 implementation



The Cortex-M3 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 single-cycle 32x32 multiplication and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M3 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M3 processor implements a version of the Thumb® instruction set, ensuring high code density and reduced program memory requirements. The Cortex-M3 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-M3 processor closely integrates a configurable *nested interrupt controlle*r (NVIC), to deliver industry-leading interrupt performance. The NVIC provides up to 16 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 any assembler stubs, removing any code overhead from the ISRs. Tail-chaining 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.

10.3.1 System level interface

The Cortex-M3 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-M3 processor has a *memory protection unit* (MPU) that provides fine grain memory control, enabling applications to implement security privilege levels, separating code, data and stack on a task-by-task basis. Such requirements are becoming critical in many embedded applications.

10.3.2 Integrated configurable debug

The Cortex-M3 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.

10.3.3 Cortex-M3 processor features and benefits summary

- tight integration of system peripherals reduces area and development costs
- Thumb instruction set combines high code density with 32-bit performance
- 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 multiplier
- 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 and tracing.

10.3.4 Cortex-M3 core peripherals

These are:

10.3.4.1 Nested Vectored Interrupt Controller

The *Nested Vectored Interrupt Controller* (NVIC) is an embedded interrupt controller that supports low latency interrupt processing.

10.3.4.2 System control block

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.

10.3.4.3 System timer

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.

10.3.4.4 Memory protection unit

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.

10.4 Programmers model

This section describes the Cortex-M3 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.

10.4.1 Processor mode and privilege levels for software execution

The processor *modes* are:

10.4.1.1 Thread mode

Used to execute application software. The processor enters Thread mode when it comes out of reset.

10.4.1.2 Handler mode

Used to handle exceptions. The processor returns to Thread mode when it has finished exception processing.

The *privilege levels* for software execution are:

10.4.1.3 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 restricted access to memory or peripherals.

Unprivileged software executes at the unprivileged level.

10.4.1.4 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 software execution is privileged or unprivileged, see “CONTROL register” on page 60. 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.

10.4.2 Stacks

The processor uses a full descending stack. This means the stack pointer indicates the last stacked item on the stack 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 independent copies of the stack pointer, see “Stack Pointer” on page 52.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see “CONTROL register” on page 60. In Handler mode, the processor always uses the main stack. The options for processor operations are:

Table 10-1. 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 (1)	Main stack or process stack(1)
Handler	Exception handlers	Always privileged	Main stack

1. See “CONTROL register” on page 60.

10.4.3 Core registers

The processor core registers are:

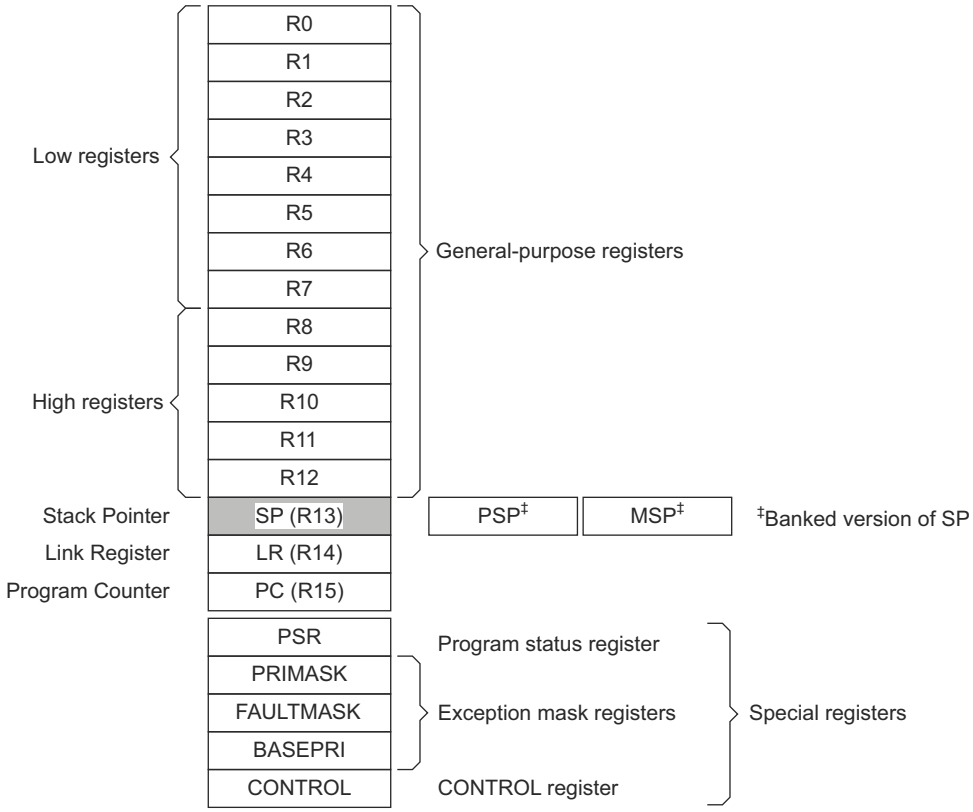


Table 10-2. Core register set summary

Name	Type [1]	Required privilege [2]	Reset value	Description
R0-R12	RW	Either	Unknown	"General-purpose registers" on page 52
MSP	RW	Privileged	See description	"Stack Pointer" on page 52
PSP	RW	Either	Unknown	"Stack Pointer" on page 52
LR	RW	Either	0xFFFFFFFF	"Link Register" on page 52
PC	RW	Either	See description	"Program Counter" on page 52
PSR	RW	Privileged	0x01000000	"Program Status Register" on page 52
ASPR	RW	Either	0x00000000	"Application Program Status Register" on page 54
IPSR	RO	Privileged	0x00000000	"Interrupt Program Status Register" on page 54
EPSR	RO	Privileged	0x01000000	"Execution Program Status Register" on page 55
PRIMASK	RW	Privileged	0x00000000	"Priority Mask Register" on page 57
FAULTMASK	RW	Privileged	0x00000000	"Fault Mask Register" on page 58
BASEPRI	RW	Privileged	0x00000000	"Base Priority Mask Register" on page 59
CONTROL	RW	Privileged	0x00000000	"CONTROL register" on page 60

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.

10.4.3.1 General-purpose registers

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

10.4.3.2 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.

10.4.3.3 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.

10.4.3.4 Program Counter

The *Program Counter* (PC) is register R15. It contains the current program address. Bit[0] is always 0 because instruction fetches must be halfword aligned. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004.

10.4.3.5 Program Status Register

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 bit assignments are:

- **APSR:**

31	30	29	28	27	26	25	24
N	Z	C	V	Q	Reserved		

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved							

- **IPSR:**

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							ISR_NUMBER

7	6	5	4	3	2	1	0
ISR_NUMBER							

- **EPSR:**

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

The PSR bit assignments are:

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

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

- read all of the registers using PSR with the MRS instruction
- write to the APSR using APSR with the MSR instruction.

The PSR combinations and attributes are:

Table 10-3. PSR register combinations

Register	Type	Combination
PSR	RW (1), (2)	APSR, EPSR, and IPSR
IEPSR	RO	EPSR and IPSR
IAPSR	RW(1)	APSR and IPSR
EAPSR	RW(2)	APSR and EPSR

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

See the instruction descriptions “MRS” on page 146 and “MSR” on page 147 for more information about how to access the program status registers.

10.4.3.6 Application Program Status Register

The APSR contains the current state of the condition flags from previous instruction executions. See the register summary in Table 10-2 on page 51 for its attributes. The bit assignments are:

- **N**

Negative or less than 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:

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**

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.

10.4.3.7 Interrupt Program Status Register

The IPSR contains the exception type number of the current *Interrupt Service Routine* (ISR). See the register summary in Table 10-2 on page 51 for its attributes. The bit assignments are:

- **ISR_NUMBER**

This is the 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

45 = IRQ29

see “Exception types” on page 71 for more information.

10.4.3.8 Execution Program Status Register

The EPSR contains the Thumb state bit, and the execution state bits for either the:

- *If-Then* (IT) instruction
- *Interruptible-Continuable Instruction* (ICI) field for an interrupted load multiple or store multiple instruction.

See the register summary in Table 10-2 on page 51 for the EPSR attributes. The bit assignments are:

- **ICI**

Interruptible-continuable instruction bits, see “Interruptible-continuable instructions” on page 55.

- **IT**

Indicates the execution state bits of the IT instruction, see “IT” on page 137.

- **T**

Always set to 1.

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 application software are ignored. Fault handlers can examine EPSR value in the stacked PSR to indicate the operation that is at fault. See “Exception entry and return” on page 75.

10.4.3.9 Interruptible-continuable instructions

When an interrupt occurs during the execution of an LDM or STM 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 execution of the multiple load or store instruction.

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

10.4.3.10 If-Then block

The If-Then block contains up to four instructions following a 16-bit 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” on page 137 for more information.

10.4.3.11 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” on page 146, “MSR” on page 147, and “CPS” on page 143 for more information.

10.4.3.12 Priority Mask Register

The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in Table 10-2 on page 51 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved							PRIMASK

- **PRIMASK**

0: no effect

1: prevents the activation of all exceptions with configurable priority.

10.4.3.13 Fault Mask Register

The FAULTMASK register prevents activation of all exceptions. See the register summary in Table 10-2 on page 51 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved							FAULTMASK

- **FAULTMASK**

0: no effect

1: prevents the activation of all exceptions.

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

10.4.3.14 Base Priority Mask Register

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. See the register summary in Table 10-2 on page 51 for its attributes. The bit assignments are:

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

- **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” on page 160 for more information. Remember that higher priority field values correspond to lower exception priorities.

10.4.3.15 CONTROL register

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode. See the register summary in Table 10-2 on page 51 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved						Active Stack Pointer	Thread Mode Privilege Level

- **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.

- **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.

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, Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, use the MSR instruction to set the Active stack pointer bit to 1, see “MSR” on page 147.

When changing the stack pointer, 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” on page 145

10.4.4 Exceptions and interrupts

The Cortex-M3 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 handler mode to handle all exceptions except for reset. See “Exception entry” on page 76 and “Exception return” on page 76 for more information.

The NVIC registers control interrupt handling. See “Nested Vectored Interrupt Controller” on page 152 for more information.

10.4.5 Data types

The processor:

- supports the following data types:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- supports 64-bit data transfer instructions.
- 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” on page 62 for more information.

10.4.6 The Cortex Microcontroller Software Interface Standard

For a Cortex-M3 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-M3 processor. It also includes optional interfaces for middleware components comprising a TCP/IP stack and a Flash file system.

CMSIS simplifies 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.

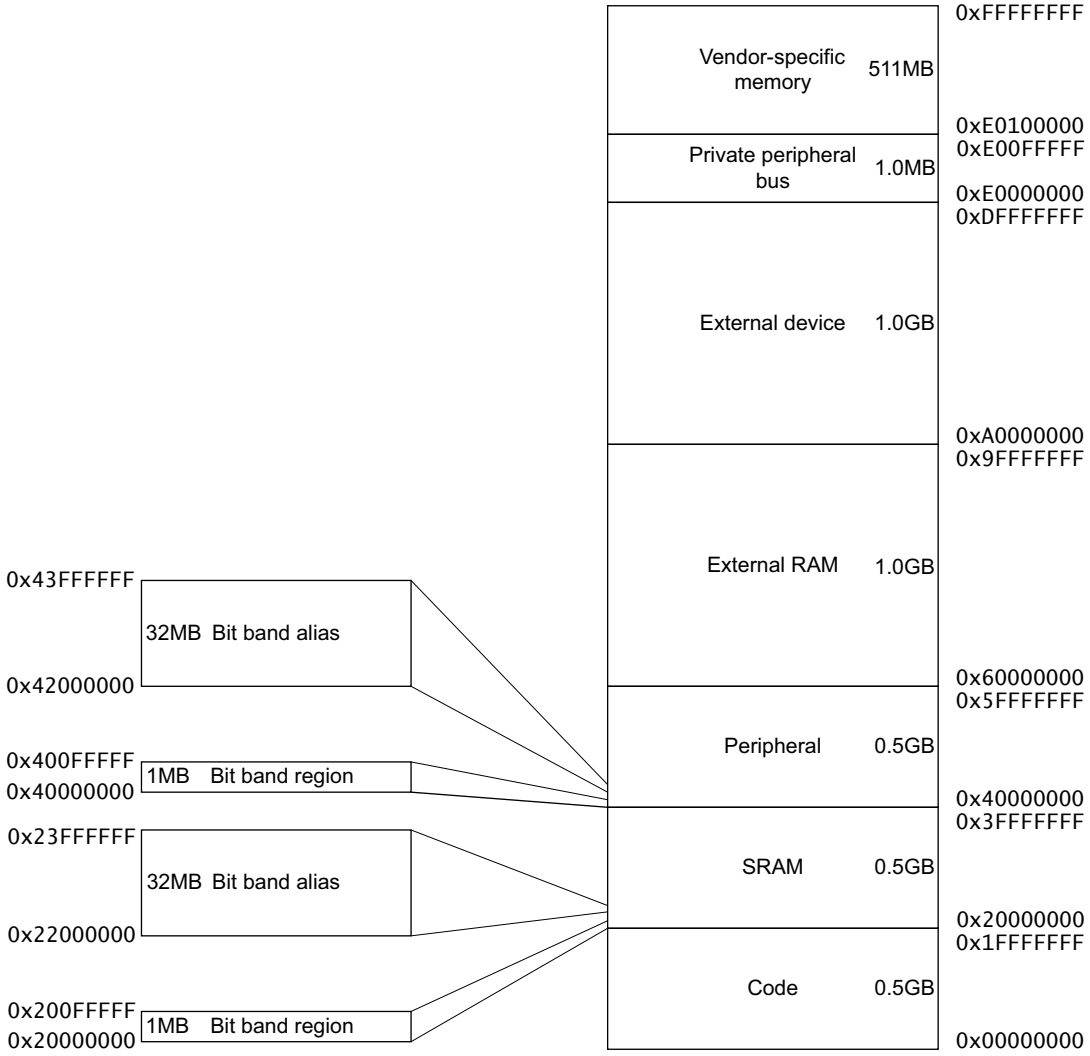
This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

The following sections give more information about the CMSIS:

- “Power management programming hints” on page 81
- “Intrinsic functions” on page 85
- “The CMSIS mapping of the Cortex-M3 NVIC registers” on page 153
- “NVIC programming hints” on page 164.

10.5 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. The memory map is:



The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data, see “Bit-banding” on page 66.

The processor reserves regions of the *Private peripheral bus* (PPB) address range for core peripheral registers, see “About the Cortex-M3 peripherals” on page 152.

This memory mapping is generic to ARM Cortex-M3 products. To get the specific memory mapping of this product, refer to the Memories section of the datasheet.

10.5.1 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.

The memory types are:

10.5.1.1 Normal

The processor can re-order transactions for efficiency, or perform speculative reads.

10.5.1.2 Device

The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.

10.5.1.3 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.

The additional memory attributes include.

10.5.1.4 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, software must ensure data coherency between the bus masters.

10.5.1.5 Execute Never (XN)

Means the processor prevents instruction accesses. Any attempt to fetch an instruction from an XN region causes a memory management fault exception.

10.5.2 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, software must insert a memory barrier instruction between the memory access instructions, see “Software ordering of memory accesses” on page 65.

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 caused by two instructions is:

A1 \ A2	Normal access	Device access		Strongly-ordered access
		Non-shareable	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.

10.5.3 Behavior of memory accesses

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

Table 10-4. Memory access behavior

Address range	Memory region	Memory type	XN	Description
0x00000000-0x1FFFFFFF	Code	Normal (1)	-	Executable region for program code. You can also put data here.
0x20000000-0x3FFFFFFF	SRAM	Normal (1)	-	Executable region for data. You can also put code here. This region includes bit band and bit band alias areas, see Table 10-6 on page 67.
0x40000000-0x5FFFFFFF	Peripheral	Device (1)	XN	This region includes bit band and bit band alias areas, see Table 10-6 on page 67.
0x60000000-0x9FFFFFFF	External RAM	Normal (1)	-	Executable region for data.
0xA0000000-0xDFFFFFFF	External device	Device (1)	XN	External Device memory
0xE0000000-0xE00FFFFF	Private Peripheral Bus	Strongly-ordered (1)	XN	This region includes the NVIC, System timer, and system control block.
0xE0100000-0xFFFFFFFF	Reserved	Device (1)	XN	Reserved

1. See "Memory regions, types and attributes" on page 62 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" on page 197.

10.5.3.1 Additional memory access constraints for shared memory

When a system includes shared memory, some memory regions have additional access constraints, and some regions are subdivided, as Table 10-5 shows:

Table 10-5. Memory region share ability policies

Address range	Memory region	Memory type	Shareability	
0x00000000-0x1FFFFFFF	Code	Normal (1)	-	
0x20000000-0x3FFFFFFF	SRAM	Normal (1)	-	
0x40000000-0x5FFFFFFF	Peripheral (2)	Device (1)	-	
0x60000000-0x7FFFFFFF	External RAM	Normal (1)	-	WBWA (2)
0x80000000-0x9FFFFFFF				WT (2)
0xA0000000-0xBFFFFFFF	External device	Device (1)	Shareable (1)	-
0xC0000000-0xDFFFFFFF			Non-shareable (1)	
0xE0000000-0xE00FFFFF	Private Peripheral Bus	Strongly-ordered (1)	Shareable (1)	-
0xE0100000-0xFFFFFFFF	Vendor-specific device (2)	Device (1)	-	-

1. See "Memory regions, types and attributes" on page 62 for more information.
2. The Peripheral and Vendor-specific device regions have no additional access constraints.

10.5.4 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" on page 63 describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

10.5.4.1 DMB

The *Data Memory Barrier* (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions. See "DMB" on page 144.

10.5.4.2 DSB

The *Data Synchronization Barrier* (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See “DSB” on page 144.

10.5.4.3 ISB

The *Instruction Synchronization Barrier* (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions. See “ISB” on page 145.

Use memory barrier instructions in, for example:

- MPU programming:
  - Use a DSB instruction to ensure the effect of the MPU takes place immediately at the end of context switching.
  - Use an ISB instruction to ensure the new MPU setting takes effect immediately after programming the MPU region or regions, if the MPU configuration code was accessed using a branch or call. If the MPU configuration code is entered using exception mechanisms, then an ISB instruction is not required.
- Vector table. If the program changes an entry in the vector table, and then enables the corresponding exception, use a DMB instruction between the operations. This ensures that if the exception is taken immediately after being enabled the processor uses the new exception vector.
- Self-modifying code. If a program contains self-modifying code, use an ISB instruction immediately after the code modification in the program. This ensures subsequent instruction execution uses the updated program.
- Memory map switching. If the system contains a memory map switching mechanism, use a DSB instruction after switching the memory map in the program. This ensures subsequent instruction execution uses the updated memory map.
- Dynamic exception priority change. When an exception priority has to change when the exception is pending or active, use DSB instructions after the change. This ensures the change takes effect on completion of the DSB instruction.
- Using a semaphore in multi-master system. If the system contains more than one bus master, for example, if another processor is present in the system, each processor must use a DMB instruction after any semaphore instructions, to ensure other bus masters see the memory transactions in the order in which they were executed.

Memory accesses to Strongly-ordered memory, such as the system control block, do not require the use of DMB instructions.

10.5.5 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 1MB of the SRAM and peripheral memory regions.

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

- accesses to the 32MB SRAM alias region map to the 1MB SRAM bit-band region, as shown in Table 10-6
- accesses to the 32MB peripheral alias region map to the 1MB peripheral bit-band region, as shown in Table 10-7.

Table 10-6. 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 10-7. 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.

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.

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
```

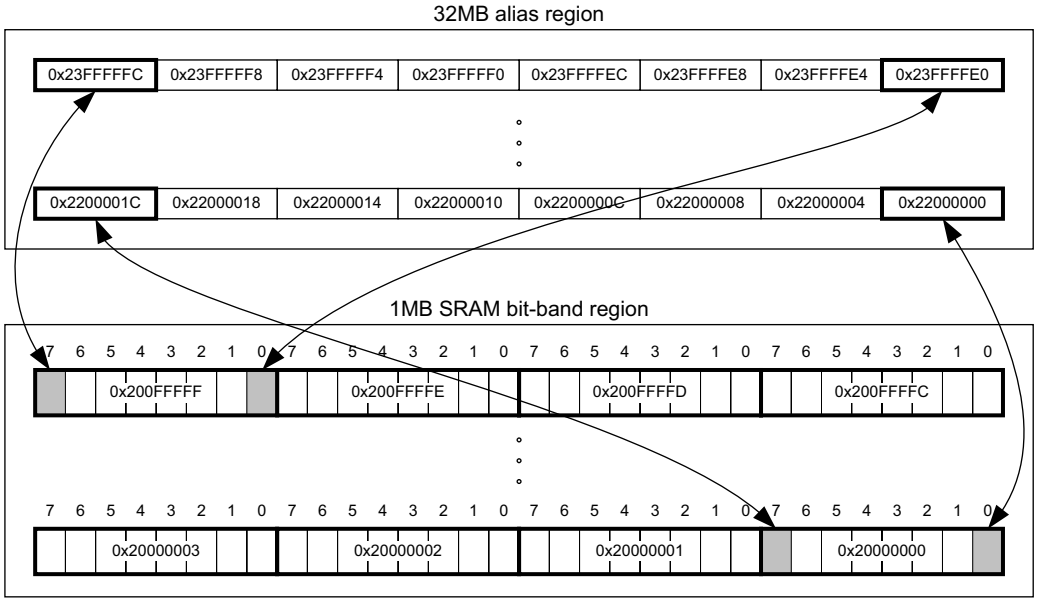
where:

- `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 10-2 shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 0x23FFFFE0 maps to bit[0] of the bit-band byte at 0x200FFFFF: 0x23FFFFE0 = 0x22000000 + (0xFFFFF*32) + (0*4).
- The alias word at 0x23FFFFFC maps to bit[7] of the bit-band byte at 0x200FFFFF: 0x23FFFFFC = 0x22000000 + (0xFFFFF*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 = 0x22000000+ (0*32) + (7*4).

Figure 10-2. Bit-band mapping



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

10.5.5.2 Directly accessing a bit-band region

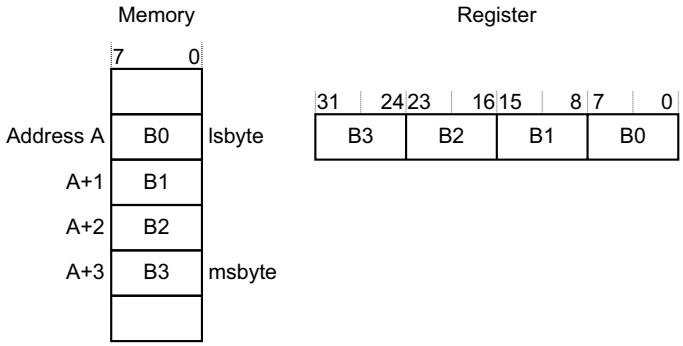
"Behavior of memory accesses" on page 64 describes the behavior of direct byte, halfword, or word accesses to the bit-band regions.

10.5.6 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. or "Little-endian format" describes how words of data are stored in memory.

10.5.6.1 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:



10.5.7 Synchronization primitives

The Cortex-M3 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. 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:

10.5.7.1 A Load-Exclusive instruction

Used to read the value of a memory location, requesting exclusive access to that location.

10.5.7.2 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.

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

To perform a guaranteed read-modify-write of a memory location, software must:

- Use a Load-Exclusive instruction to read the value of the location.
- Update the value, as required.
- Use a Store-Exclusive instruction to attempt to write the new value back to the memory location, and tests 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 the first step might be out of date. The software must retry the read-modify-write sequence,

Software can use the synchronization primitives to implement a semaphores as follows:

- Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.
- If the semaphore is free, use a Store-Exclusive to write the claim value to the semaphore address.

- If the returned status bit from the second step indicates that the Store-Exclusive succeeded then the software has claimed the semaphore. However, if the Store-Exclusive failed, another process might have claimed the semaphore after the software performed the first step.

The Cortex-M3 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 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” on page 106 and “CLREX” on page 108.

10.5.8 Programming hints for the synchronization primitives

ANSI C cannot directly generate the exclusive access instructions. Some C compilers provide intrinsic functions for generation of these instructions:

Table 10-8. C compiler intrinsic functions for exclusive access instructions

Instruction	Intrinsic function
LDREX, LDREXH, or LDREXB	unsigned int __ldrex(volatile void *ptr)
STREX, STREXH, or STREXB	int __strex(unsigned int val, volatile void *ptr)
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 require LDREXB operation:

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

10.6 Exception model

This section describes the exception model.

10.6.1 Exception states

Each exception is in one of the following states:

10.6.1.1 Inactive

The exception is not active and not pending.

10.6.1.2 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.

10.6.1.3 Active

An exception that 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.

10.6.1.4 Active and pending

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

10.6.2 Exception types

The exception types are:

10.6.2.1 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.

10.6.2.2 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.

10.6.2.3 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.

10.6.2.4 Memory management fault

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.

10.6.2.5 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.

10.6.2.6 Usage fault

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

- an undefined instruction
- an illegal unaligned access
- 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
- division by zero.

10.6.2.7 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.

10.6.2.8 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.

10.6.2.9 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.

10.6.2.10 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 10-9. Properties of the different exception types

Exception number (1)	IRQ number (1)	Exception type	Priority	Vector address or offset (2)	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 (3)	0x00000010	Synchronous
5	-11	Bus fault	Configurable (3)	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	Usage fault	Configurable (3)	0x00000018	Synchronous
7-10	-	-	-	Reserved	-
11	-5	SVCall	Configurable (3)	0x0000002C	Synchronous
12-13	-	-	-	Reserved	-
14	-2	PendSV	Configurable (3)	0x00000038	Asynchronous
15	-1	SysTick	Configurable (3)	0x0000003C	Asynchronous
16 and above	0 and above (4)	Interrupt (IRQ)	Configurable (5)	0x00000040 and above (6)	Asynchronous

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” on page 54.
2. See “Vector table” on page 74 for more information.
3. See “System Handler Priority Registers” on page 177.
4. See the “Peripheral Identifiers” section of the datasheet.
5. See “Interrupt Priority Registers” on page 160.
6. 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 10-9 on page 72 shows as having configurable priority, see:

- “System Handler Control and State Register” on page 180
- “Interrupt Clear-enable Registers” on page 156.

For more information about hard faults, memory management faults, bus faults, and usage faults, see “Fault handling” on page 78.

10.6.3 Exception handlers

The processor handles exceptions using:

10.6.3.1 Interrupt Service Routines (ISRs)

Interrupts IRQ0 to IRQ29 are the exceptions handled by ISRs.

10.6.3.2 Fault handlers

Hard fault, memory management fault, usage fault, bus fault are fault exceptions handled by the fault handlers.

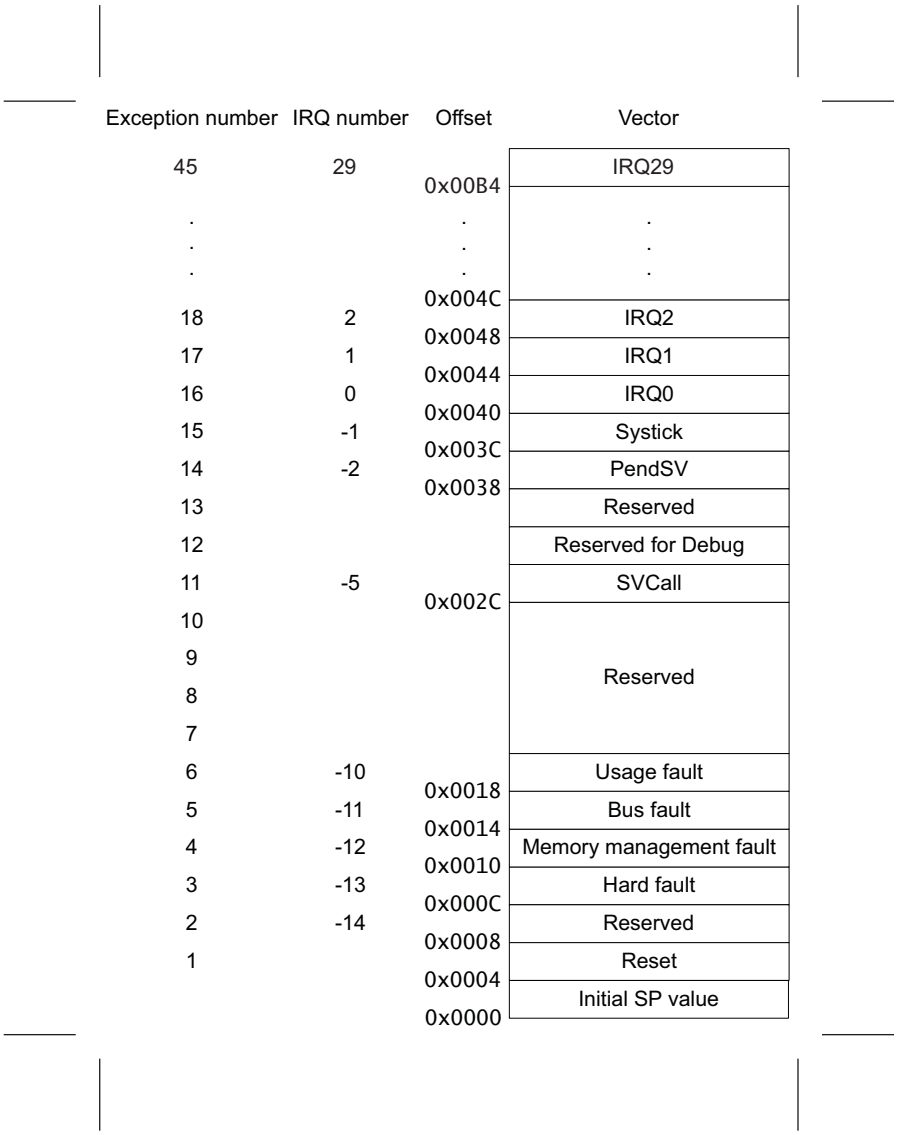
10.6.3.3 System handlers

NMI, PendSV, SVCall SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

10.6.4 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 10-3 on page 74 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 10-3. Vector table



On system reset, the vector table is fixed at address 0x00000000. Privileged software can write to the VTOR to relocate the vector table start address to a different memory location, in the range 0x00000080 to 0x3FFFFF80, see “Vector Table Offset Register” on page 171.

10.6.5 Exception priorities

As Table 10-9 on page 72 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.

If 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” on page 177
- “Interrupt Priority Registers” on page 160.

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.

10.6.6 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” on page 172.

10.6.7 Exception entry and return

Descriptions of exception handling use the following terms:

10.6.7.1 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” on page 75 for more information about preemption by an interrupt.

When one exception preempts another, the exceptions are called nested exceptions. See “Exception entry” on page 76 more information.

10.6.7.2 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” on page 76 for more information.

10.6.7.3 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.

10.6.7.4 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.

10.6.7.5 Exception entry

Exception entry occurs when there is a pending exception with sufficient priority and either:

- the processor is in Thread mode
- the new exception is of 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 the exception has more priority than any limits set by the mask registers, see “Exception mask registers” on page 56. 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 as *stack frame*. The stack frame contains the following information:

- R0-R3, R12
- Return address
- PSR
- LR.

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. Unless stack alignment is disabled, the stack frame is aligned to a double-word address. If the STKALIGN bit of the *Configuration Control Register* (CCR) is set to 1, stack align adjustment is performed during stacking.

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 was processor was in before the entry occurred.

If no higher priority exception occurs during 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 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.

10.6.7.6 Exception return

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:

- a POP instruction that includes the PC
- a BX instruction with any register.
- an LDR or LDM instruction with the PC as the destination.

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 four bits of this value provide information on the return stack and processor mode. Table 10-10 shows the EXC_RETURN[3:0] values with a description of the exception return behavior.

The processor sets EXC_RETURN bits[31:4] to 0xFFFFFFF. When this value is loaded into the PC it indicates to the processor that the exception is complete, and the processor initiates the exception return sequence.

Table 10-10. Exception return behavior

EXC_RETURN[3:0]	Description
bXXX0	Reserved.
b0001	Return to Handler mode. Exception return gets state from MSP. Execution uses MSP after return.
b0011	Reserved.
b01X1	Reserved.
b1001	Return to Thread mode. Exception return gets state from MSP. Execution uses MSP after return.
b1101	Return to Thread mode. Exception return gets state from PSP. Execution uses PSP after return.
b1X11	Reserved.

10.7 Fault handling

Faults are a subset of the exceptions, see “Exception model” on page 70. 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 or an attempt to change state with a BX instruction
- attempting to execute an instruction from a memory region marked as *Non-Executable* (XN).
- an MPU fault because of a privilege violation or an attempt to access an unmanaged region.

10.7.1 Fault types

Table 10-11 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” on page 182 for more information about the fault status registers.

Table 10-11. Faults

Fault	Handler	Bit name	Fault status register
Bus error on a vector read	Hard fault	VECTTBL	“Hard Fault Status Register” on page 188
Fault escalated to a hard fault		FORCED	
MPU mismatch:	Memory management fault	-	-
on instruction access		IACCVIOL (1)	“Memory Management Fault Address Register” on page 189
on data access		DACCVIOL	
during exception stacking		MSTKERR	
during exception unstacking		MUNSKERR	
Bus error:	Bus fault	-	-
during exception stacking		STKERR	“Bus Fault Status Register” on page 184
during exception unstacking		UNSTKERR	
during instruction prefetch		IBUSERR	
Precise data bus error		PRECISERR	
Imprecise data bus error		IMPRECISERR	
Attempt to access a coprocessor	Usage fault	NOCP	“Usage Fault Status Register” on page 186
Undefined instruction		UNDEFINSTR	
Attempt to enter an invalid instruction set state (2)		INVSTATE	
Invalid EXC_RETURN value		INVPC	
Illegal unaligned load or store		UNALIGNED	
Divide By 0		DIVBYZERO	

1. Occurs on an access to an XN region even if the MPU is disabled.
2. Attempting to use an instruction set other than the Thumb instruction set.

10.7.2 Fault escalation and hard faults

All faults exceptions except for hard fault have configurable exception priority, see “System Handler Priority Registers” on page 177. Software can disable execution of the handlers for these faults, see “System Handler Control and State Register” on page 180.

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” on page 70.

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 because 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.

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.

10.7.3 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 10-12.

Table 10-12. Fault status and fault address registers

Handler	Status register name	Address register name	Register description
Hard fault	HFSR	-	“Hard Fault Status Register” on page 188
Memory management fault	MMFSR	MMFAR	“Memory Management Fault Status Register” on page 183 “Memory Management Fault Address Register” on page 189
Bus fault	BFSR	BFAR	“Bus Fault Status Register” on page 184 “Bus Fault Address Register” on page 190
Usage fault	UFSR	-	“Usage Fault Status Register” on page 186

10.7.4 Lockup

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

- it is reset

10.8 Power management

The Cortex-M3 processor sleep modes reduce power consumption:

- Backup Mode
- Wait Mode
- Sleep Mode

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see “System Control Register” on page 174. For more information about the behavior of the sleep modes see “Low Power Modes” in the PMC section of the datasheet.

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

10.8.1 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 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.

10.8.1.1 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” on page 151 for more information.

10.8.1.2 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” on page 150 for more information.

10.8.1.3 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.

10.8.2 Wakeup from sleep mode

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

10.8.2.1 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 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” on page 56.

10.8.2.2 Wakeup from WFE

The processor wakes up if:

- it detects an exception with sufficient priority to cause exception entry

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 exception entry. For more information about the SCR see “System Control Register” on page 174.

10.8.3 Power management programming hints

ANSI C cannot directly generate the WFI and WFE instructions. The CMSIS provides the following intrinsic functions for these instructions:

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

10.9 Instruction set summary

The processor implements a version of the Thumb instruction set. Table 10-13 lists the supported instructions.

In Table 10-13:

- 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 10-13. Cortex-M3 instructions

Mnemonic	Operands	Brief description	Flags	Page
ADC, ADCS	{Rd,} Rn, Op2	Add with Carry	N,Z,C,V	page 110
ADD, ADDS	{Rd,} Rn, Op2	Add	N,Z,C,V	page 110
ADD, ADDW	{Rd,} Rn, #imm12	Add	N,Z,C,V	page 110
ADR	Rd, label	Load PC-relative address	-	page 95
AND, ANDS	{Rd,} Rn, Op2	Logical AND	N,Z,C	page 113
ASR, ASRS	Rd, Rm, <Rs|#n>	Arithmetic Shift Right	N,Z,C	page 114
B	label	Branch	-	page 134
BFC	Rd, #lsb, #width	Bit Field Clear	-	page 130
BFI	Rd, Rn, #lsb, #width	Bit Field Insert	-	page 130
BIC, BICS	{Rd,} Rn, Op2	Bit Clear	N,Z,C	page 113
BKPT	#imm	Breakpoint	-	page 142
BL	label	Branch with Link	-	page 134
BLX	Rm	Branch indirect with Link	-	page 134
BX	Rm	Branch indirect	-	page 134
CBNZ	Rn, label	Compare and Branch if Non Zero	-	page 136
CBZ	Rn, label	Compare and Branch if Zero	-	page 136
CLREX	-	Clear Exclusive	-	page 108

Table 10-13. Cortex-M3 instructions (Continued)

Mnemonic	Operands	Brief description	Flags	Page
CLZ	Rd, Rm	Count leading zeros	-	page 116
CMN, CMNS	Rn, Op2	Compare Negative	N,Z,C,V	page 117
CMP, CMPS	Rn, Op2	Compare	N,Z,C,V	page 117
CPSID	iflags	Change Processor State, Disable Interrupts	-	page 143
CPSIE	iflags	Change Processor State, Enable Interrupts	-	page 143
DMB	-	Data Memory Barrier	-	page 144
DSB	-	Data Synchronization Barrier	-	page 144
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR	N,Z,C	page 113
ISB	-	Instruction Synchronization Barrier	-	page 145
IT	-	If-Then condition block	-	page 137
LDM	Rn{!}, reglist	Load Multiple registers, increment after	-	page 103
LDMDB, LDMEA	Rn{!}, reglist	Load Multiple registers, decrement before	-	page 103
LDMFD, LDMIA	Rn{!}, reglist	Load Multiple registers, increment after	-	page 103
LDR	Rt, [Rn, #offset]	Load Register with word	-	page 98
LDRB, LDRBT	Rt, [Rn, #offset]	Load Register with byte	-	page 98
LDRD	Rt, Rt2, [Rn, #offset]	Load Register with two bytes	-	page 98
LDREX	Rt, [Rn, #offset]	Load Register Exclusive	-	page 98
LDREXB	Rt, [Rn]	Load Register Exclusive with byte	-	page 98
LDREXH	Rt, [Rn]	Load Register Exclusive with halfword	-	page 98
LDRH, LDRHT	Rt, [Rn, #offset]	Load Register with halfword	-	page 98
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load Register with signed byte	-	page 98
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load Register with signed halfword	-	page 98
LDRT	Rt, [Rn, #offset]	Load Register with word	-	page 98
LSL, LSLS	Rd, Rm, <Rs|#n>	Logical Shift Left	N,Z,C	page 114
LSR, LSRS	Rd, Rm, <Rs|#n>	Logical Shift Right	N,Z,C	page 114
MLA	Rd, Rn, Rm, Ra	Multiply with Accumulate, 32-bit result	-	page 124
MLS	Rd, Rn, Rm, Ra	Multiply and Subtract, 32-bit result	-	page 124
MOV, MOVS	Rd, Op2	Move	N,Z,C	page 118
MOVT	Rd, #imm16	Move Top	-	page 120
MOVW, MOV	Rd, #imm16	Move 16-bit constant	N,Z,C	page 118
MRS	Rd, spec_reg	Move from special register to general register	-	page 146
MSR	spec_reg, Rm	Move from general register to special register	N,Z,C,V	page 147
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result	N,Z	page 124
MVN, MVNS	Rd, Op2	Move NOT	N,Z,C	page 118
NOP	-	No Operation	-	page 148
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT	N,Z,C	page 113
ORR, ORRS	{Rd,} Rn, Op2	Logical OR	N,Z,C	page 113

Table 10-13. Cortex-M3 instructions (Continued)

Mnemonic	Operands	Brief description	Flags	Page
POP	reglist	Pop registers from stack	-	page 105
PUSH	reglist	Push registers onto stack	-	page 105
RBIT	Rd, Rn	Reverse Bits	-	page 121
REV	Rd, Rn	Reverse byte order in a word	-	page 121
REV16	Rd, Rn	Reverse byte order in each halfword	-	page 121
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend	-	page 121
ROR, RORS	Rd, Rm, <Rs|#n>	Rotate Right	N,Z,C	page 114
RRX, RRXS	Rd, Rm	Rotate Right with Extend	N,Z,C	page 114
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract	N,Z,C,V	page 110
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry	N,Z,C,V	page 110
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract	-	page 131
SDIV	{Rd,} Rn, Rm	Signed Divide	-	page 126
SEV	-	Send Event	-	page 148
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply with Accumulate (32 x 32 + 64), 64-bit result	-	page 125
SMULL	RdLo, RdHi, Rn, Rm	Signed Multiply (32 x 32), 64-bit result	-	page 125
SSAT	Rd, #n, Rm {,shift #s}	Signed Saturate	Q	page 127
STM	Rn{!}, reglist	Store Multiple registers, increment after	-	page 103
STMDB, STMEA	Rn{!}, reglist	Store Multiple registers, decrement before	-	page 103
STMFD, STMIA	Rn{!}, reglist	Store Multiple registers, increment after	-	page 103
STR	Rt, [Rn, #offset]	Store Register word	-	page 98
STRB, STRBT	Rt, [Rn, #offset]	Store Register byte	-	page 98
STRD	Rt, Rt2, [Rn, #offset]	Store Register two words	-	page 98
STREX	Rd, Rt, [Rn, #offset]	Store Register Exclusive	-	page 106
STREXB	Rd, Rt, [Rn]	Store Register Exclusive byte	-	page 106
STREXH	Rd, Rt, [Rn]	Store Register Exclusive halfword	-	page 106
STRH, STRHT	Rt, [Rn, #offset]	Store Register halfword	-	page 98
STRT	Rt, [Rn, #offset]	Store Register word	-	page 98
SUB, SUBS	{Rd,} Rn, Op2	Subtract	N,Z,C,V	page 110
SUB, SUBW	{Rd,} Rn, #imm12	Subtract	N,Z,C,V	page 110
SVC	#imm	Supervisor Call	-	page 149
SXTB	{Rd,} Rm {,ROR #n}	Sign extend a byte	-	page 132
SXTH	{Rd,} Rm {,ROR #n}	Sign extend a halfword	-	page 132
TBB	[Rn, Rm]	Table Branch Byte	-	page 139
TBH	[Rn, Rm, LSL #1]	Table Branch Halfword	-	page 139
TEQ	Rn, Op2	Test Equivalence	N,Z,C	page 122
TST	Rn, Op2	Test	N,Z,C	page 122

Table 10-13. Cortex-M3 instructions (Continued)

Mnemonic	Operands	Brief description	Flags	Page
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract	-	page 131
UDIV	{Rd,} Rn, Rm	Unsigned Divide	-	page 126
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result	-	page 125
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32 x 32), 64-bit result	-	page 125
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate	Q	page 127
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a byte	-	page 132
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a halfword	-	page 132
WFE	-	Wait For Event	-	page 150
WFI	-	Wait For Interrupt	-	page 151

10.10 Intrinsic functions

ANSI cannot directly access some Cortex-M3 instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, you might have to use inline assembler to access some instructions.

The CMSIS provides the following intrinsic functions to generate instructions that ANSI cannot directly access:

Table 10-14. CMSIS intrinsic functions to generate some Cortex-M3 instructions

Instruction	CMSIS intrinsic 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 10-15. 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)

10.11 About the instruction descriptions

The following sections give more information about using the instructions:

- “Operands” on page 86
- “Restrictions when using PC or SP” on page 86
- “Flexible second operand” on page 86
- “Shift Operations” on page 87
- “Address alignment” on page 90
- “PC-relative expressions” on page 90
- “Conditional execution” on page 91
- “Instruction width selection” on page 92.

10.11.1 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 in that they can either be a register or a constant. See “Flexible second operand” .

10.11.2 Restrictions when using PC or SP

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

Bit[0] of any address you write 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-M3 processor only supports Thumb instructions.

10.11.3 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” on page 87

10.11.3.1 Constant

You 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.

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.

10.11.3.2 Instruction substitution

Your assembler might be able to produce an equivalent instruction in cases where you specify a constant that is not permitted. For example, an assembler might assemble the instruction CMP *Rd*, #0xFFFFFFFE as the equivalent instruction CMN *Rd*, #0x2.

10.11.3.3 Register with optional shift

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

If you omit the shift, or specify LSL #0, the instruction uses the value in *Rm*.

If you specify 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 “Shift Operations”

10.11.4 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” on page 86. The result is used by the instruction.

The permitted shift lengths depend on the shift type and the instruction, see the individual instruction description or “Flexible second operand” on page 86. 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.

10.11.4.1 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 10-4 on page 88.

You can use the ASR #n operation 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 10-4. ASR #3

10.11.4.2 LSR

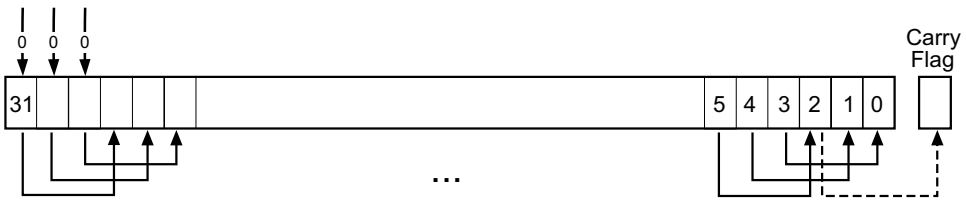
Logical shift right by n bits moves the left-hand 32-n bits of the register Rm, to the right by *n* places, into the right-hand 32-n bits of the result. And it sets the left-hand n bits of the result to 0. See Figure 10-5.

You can use the LSR #n operation 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 10-5. LSR #3



10.11.4.3 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 10-6 on page 89.

You can use he LSL #n operation 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 10-6. LSL #3

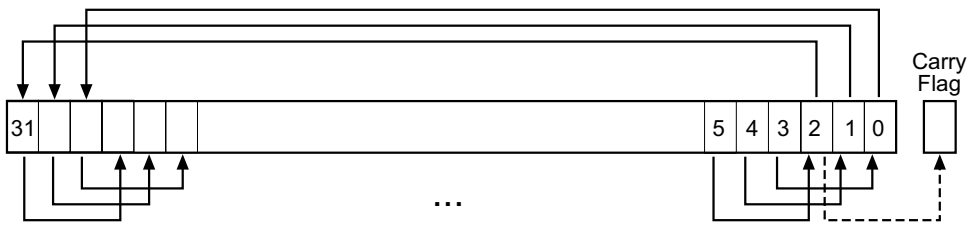
10.11.4.4 ROR

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 10-7.

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 10-7. ROR #3

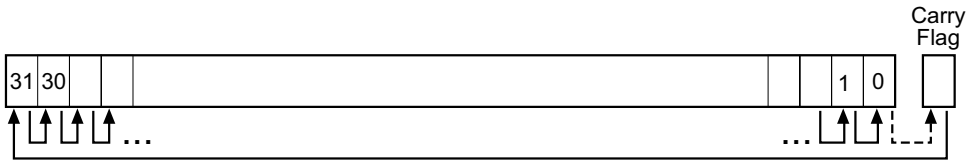


10.11.4.5 RRX

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 10-8 on page 90.

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 10-8. RRX



10.11.5 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-M3 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” on page 78.

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” on page 175.

10.11.6 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].

10.11.7 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” on page 54. 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.

You can execute an instruction 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 10-16 on page 92 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” on page 137 for more information and restrictions when using the IT instruction. Depending on the vendor, the assembler might automatically insert an IT instruction if you have conditional instructions outside the IT block.

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

This section describes:

- “The condition flags”
- “Condition code suffixes” .

10.11.7.1 The 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” on page 52.

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.

Overflow occurs if the result of an add, subtract, or compare is greater than or equal to 2^{31}, or less than -2^{31}.

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

10.11.7.2 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 10-16 shows the condition codes to use.

You can use conditional execution with the IT instruction to reduce the number of branch instructions in code.

Table 10-16 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

Table 10-16. Condition code suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned ≥
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 ≤
GE	N = V	Greater than or equal, signed ≥
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 ≤
AL	Can have any value	Always. This is the default when no suffix is specified.

10.11.7.3 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
```

10.11.7.4 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            ; IT 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
```

10.11.8 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, you 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 you specify an instruction width suffix and the assembler cannot generate an instruction encoding of the requested width, it generates an error.

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.

10.11.8.1 Instruction width selection

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
```

10.12 Memory access instructions

Table 10-17 shows the memory access instructions:

Table 10-17. Memory access instructions

Mnemonic	Brief description	See
ADR	Load PC-relative address	"ADR" on page 95
CLREX	Clear Exclusive	"CLREX" on page 108
LDM{mode}	Load Multiple registers	"LDM and STM" on page 103
LDR{type}	Load Register using immediate offset	"LDR and STR, immediate offset" on page 96
LDR{type}	Load Register using register offset	"LDR and STR, register offset" on page 98
LDR{type}T	Load Register with unprivileged access	"LDR and STR, unprivileged" on page 100
LDR	Load Register using PC-relative address	"LDR, PC-relative" on page 101
LDREX{type}	Load Register Exclusive	"LDREX and STREX" on page 106
POP	Pop registers from stack	"PUSH and POP" on page 105
PUSH	Push registers onto stack	"PUSH and POP" on page 105
STM{mode}	Store Multiple registers	"LDM and STM" on page 103
STR{type}	Store Register using immediate offset	"LDR and STR, immediate offset" on page 96
STR{type}	Store Register using register offset	"LDR and STR, register offset" on page 98
STR{type}T	Store Register with unprivileged access	"LDR and STR, unprivileged" on page 100
STREX{type}	Store Register Exclusive	"LDREX and STREX" on page 106

10.12.1 ADR

Load PC-relative address.

10.12.1.1 Syntax

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

where:

cond — is an optional condition code, see “Conditional execution” on page 91.

Rd — is the destination register.

label — is a PC-relative expression. See “PC-relative expressions” on page 90.

10.12.1.2 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 you use ADR to generate a target address for a BX or BLX instruction, you must ensure that bit[0] of the address you generate is set to1 for correct execution.

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

You 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” on page 92.

10.12.1.3 Restrictions

Rd must not be SP and must not be PC.

10.12.1.4 Condition flags

This instruction does not change the flags.

10.12.1.5 Examples

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

10.12.2 LDR and STR, immediate offset

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

10.12.2.1 Syntax

```
op{type}{cond} Rt, [Rn {, #offset}]        ; immediate offset
op{type}{cond} Rt, [Rn, #offset]!          ; pre-indexed
op{type}{cond} Rt, [Rn], #offset           ; post-indexed
opD{cond} Rt, Rt2, [Rn {, #offset}]        ; immediate offset, two words
opD{cond} Rt, Rt2, [Rn, #offset]!          ; pre-indexed, two words
opD{cond} Rt, Rt2, [Rn], #offset           ; 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” on page 91.

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.

10.12.2.2 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:

10.12.2.3 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]
```

10.12.2.4 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]!
```

10.12.2.5 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” on page 90.

Table 10-18 shows the ranges of offset for immediate, pre-indexed and post-indexed forms.

Table 10-18. 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

10.12.2.6 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.

10.12.2.7 Condition flags

These instructions do not change the flags.

10.12.2.8 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.
```

10.12.3 LDR and STR, register offset

Load and Store with register offset.

10.12.3.1 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” on page 91.

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.

10.12.3.2 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” on page 90.

10.12.3.3 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.

10.12.3.4 Condition flags

These instructions do not change the flags.

10.12.3.5 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
```

10.12.4 LDR and STR, unprivileged

Load and Store with unprivileged access.

10.12.4.1 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” on page 91.

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*.

10.12.4.2 Operation

These load and store instructions perform the same function as the memory access instructions with immediate offset, see “LDR and STR, immediate offset” on page 96. 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.

10.12.4.3 Restrictions

In these instructions:

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

10.12.4.4 Condition flags

These instructions do not change the flags.

10.12.4.5 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
```

10.12.5 LDR, PC-relative

Load register from memory.

10.12.5.1 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” on page 91.

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” on page 90.

10.12.5.2 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” on page 90.

label must be within a limited range of the current instruction. Table 10-19 shows the possible offsets between *label* and the PC.

Table 10-19. Offset ranges

Instruction type	Offset range
Word, halfword, signed halfword, byte, signed byte	–4095 to 4095
Two words	–1020 to 1020

You might have to use the .W suffix to get the maximum offset range. See “Instruction width selection” on page 92.

10.12.5.3 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.

10.12.5.4 Condition flags

These instructions do not change the flags.

10.12.5.5 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
```

10.12.6 LDM and STM

Load and Store Multiple registers.

10.12.6.1 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” on page 91.

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” on page 104.

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.

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

10.12.6.2 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 STMFD 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” on page 105 for details.

10.12.6.3 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 you specify the writeback suffix.

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.

10.12.6.4 Condition flags

These instructions do not change the flags.

10.12.6.5 Examples

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

10.12.6.6 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
```

10.12.7 PUSH and POP

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

10.12.7.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 91.

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.

10.12.7.2 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” on page 103 for more information.

10.12.7.3 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.

10.12.7.4 Condition flags

These instructions do not change the flags.

10.12.7.5 Examples

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

10.12.8 LDREX and STREX

Load and Store Register Exclusive.

10.12.8.1 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” on page 91.

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*.

10.12.8.2 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” on page 69

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.

10.12.8.3 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.

10.12.8.4 Condition flags

These instructions do not change the flags.

10.12.8.5 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
```

10.12.9 CLREX

Clear Exclusive.

10.12.9.1 Syntax

`CLREX{`*cond*`}`

where:

cond is an optional condition code, see “Conditional execution” on page 91.

10.12.9.2 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” on page 69 for more information.

10.12.9.3 Condition flags

These instructions do not change the flags.

10.12.9.4 Examples

```
CLREX
```

10.13 General data processing instructions

Table 10-20 shows the data processing instructions:

Table 10-20. Data processing instructions

Mnemonic	Brief description	See
ADC	Add with Carry	"ADD, ADC, SUB, SBC, and RSB" on page 110
ADD	Add	"ADD, ADC, SUB, SBC, and RSB" on page 110
ADDW	Add	"ADD, ADC, SUB, SBC, and RSB" on page 110
AND	Logical AND	"AND, ORR, EOR, BIC, and ORN" on page 113
ASR	Arithmetic Shift Right	"ASR, LSL, LSR, ROR, and RRX" on page 114
BIC	Bit Clear	"AND, ORR, EOR, BIC, and ORN" on page 113
CLZ	Count leading zeros	"CLZ" on page 116
CMN	Compare Negative	"CMP and CMN" on page 117
CMP	Compare	"CMP and CMN" on page 117
EOR	Exclusive OR	"AND, ORR, EOR, BIC, and ORN" on page 113
LSL	Logical Shift Left	"ASR, LSL, LSR, ROR, and RRX" on page 114
LSR	Logical Shift Right	"ASR, LSL, LSR, ROR, and RRX" on page 114
MOV	Move	"MOV and MVN" on page 118
MOVT	Move Top	"MOVT" on page 120
MOVW	Move 16-bit constant	"MOV and MVN" on page 118
MVN	Move NOT	"MOV and MVN" on page 118
ORN	Logical OR NOT	"AND, ORR, EOR, BIC, and ORN" on page 113
ORR	Logical OR	"AND, ORR, EOR, BIC, and ORN" on page 113
RBIT	Reverse Bits	"REV, REV16, REVSH, and RBIT" on page 121
REV	Reverse byte order in a word	"REV, REV16, REVSH, and RBIT" on page 121
REV16	Reverse byte order in each halfword	"REV, REV16, REVSH, and RBIT" on page 121
REVSH	Reverse byte order in bottom halfword and sign extend	"REV, REV16, REVSH, and RBIT" on page 121
ROR	Rotate Right	"ASR, LSL, LSR, ROR, and RRX" on page 114
RRX	Rotate Right with Extend	"ASR, LSL, LSR, ROR, and RRX" on page 114
RSB	Reverse Subtract	"ADD, ADC, SUB, SBC, and RSB" on page 110
SBC	Subtract with Carry	"ADD, ADC, SUB, SBC, and RSB" on page 110
SUB	Subtract	"ADD, ADC, SUB, SBC, and RSB" on page 110
SUBW	Subtract	"ADD, ADC, SUB, SBC, and RSB" on page 110
TEQ	Test Equivalence	"TST and TEQ" on page 122
TST	Test	"TST and TEQ" on page 122

10.13.1 ADD, ADC, SUB, SBC, and RSB

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

10.13.1.1 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” on page 91.

cond is an optional condition code, see “Conditional execution” on page 91.

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” on page 86 for details of the options.

imm12 is any value in the range 0-4095.

10.13.1.2 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 page 112.

See also “ADR” on page 95.

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

10.13.1.3 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:

- you 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:
  - you must not specify the S suffix
  - the second operand must be a constant in the range 0 to 4095.
  -
  - When using the PC for an addition or a subtraction, bits[1:0] of the PC are rounded to b00 before performing the calculation, making the base address for the calculation word-aligned.
  - If you want to generate the address of an instruction, you have to adjust the constant based on the value of the PC. ARM recommends that you use the ADR instruction instead of ADD or SUB with *Rn* equal to the PC, because your 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.

10.13.1.4 Condition flags

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

10.13.1.5 Examples

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

10.13.1.6 Multiword arithmetic examples

10.13.1.7 64-bit addition

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.

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

10.13.1.8 96-bit subtraction

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.

```
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
```

10.13.2 AND, ORR, EOR, BIC, and ORN

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

10.13.2.1 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” on page 91.

cond is an optional condition code, see See “Conditional execution” on page 91..

Rd is the destination register.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See “Flexible second operand” on page 86 for details of the options.

10.13.2.2 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*.

10.13.2.3 Restrictions

Do not use SP and do not use PC.

10.13.2.4 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” on page 86
- do not affect the V flag.

10.13.2.5 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
```

10.13.3 ASR, LSL, LSR, ROR, and RRX

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

10.13.3.1 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” on page 91.

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 1 to 31.

MOV{S}{cond} Rd, Rm is the preferred syntax for LSL{S}{cond} Rd, Rm, #0.

10.13.3.2 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” on page 87.

10.13.3.3 Restrictions

Do not use SP and do not use PC.

10.13.3.4 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” on page 87.

10.13.3.5 Examples

```
ASR    R7, R8, #9  ; Arithmetic shift right by 9 bits
LSLS   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
```

10.13.4 CLZ

Count Leading Zeros.

10.13.4.1 Syntax

```
CLZ{cond} Rd, Rm
```

where:

cond is an optional condition code, see “Conditional execution” on page 91.

Rd is the destination register.

Rm is the operand register.

10.13.4.2 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 in the source register, and zero if bit[31] is set.

10.13.4.3 Restrictions

Do not use SP and do not use PC.

10.13.4.4 Condition flags

This instruction does not change the flags.

10.13.4.5 Examples

```
CLZ     R4,R9
CLZNE   R2,R3
```

10.13.5 CMP and CMN

Compare and Compare Negative.

10.13.5.1 Syntax

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

where:

cond — is an optional condition code, see “Conditional execution” on page 91.

Rn — is the register holding the first operand.

Operand2 — is a flexible second operand. See “Flexible second operand” on page 86 for details of the options.

10.13.5.2 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.

10.13.5.3 Restrictions

In these instructions:

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

10.13.5.4 Condition flags

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

10.13.5.5 Examples

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

10.13.6 MOV and MVN

Move and Move NOT.

10.13.6.1 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” on page 91.

cond is an optional condition code, see “Conditional execution” on page 91.

Rd is the destination register.

Operand2 is a flexible second operand. See “Flexible second operand” on page 86 for details of the options.

imm16 is any value in the range 0-65535.

10.13.6.2 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” on page 114.

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.

10.13.6.3 Restrictions

You can use SP and PC only in the MOV instruction, with the following restrictions:

- the second operand must be a register without shift
- you must not specify the S suffix.

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.

10.13.6.4 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” on page 86
- do not affect the V flag.

10.13.6.5 Example

```
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 0xFFFFFFF0 (bitwise inverse of 0xF)
                      ; to the R2 and update flags
```

10.13.7 MOVT

Move Top.

10.13.7.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 91.

Rd is the destination register.

imm16 is a 16-bit immediate constant.

10.13.7.2 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 you to generate any 32-bit constant.

10.13.7.3 Restrictions

Rd must not be SP and must not be PC.

10.13.7.4 Condition flags

This instruction does not change the flags.

10.13.7.5 Examples

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

10.13.8 REV, REV16, REVSH, and RBIT

Reverse bytes and Reverse bits.

10.13.8.1 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” on page 91.

Rd is the destination register.

Rn is the register holding the operand.

10.13.8.2 Operation

Use these instructions to change endianness of data:

REV converts 32-bit big-endian data into little-endian data or 32-bit little-endian data into big-endian data.

REV16 converts 16-bit big-endian data into little-endian data or 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.

10.13.8.3 Restrictions

Do not use SP and do not use PC.

10.13.8.4 Condition flags

These instructions do not change the flags.

10.13.8.5 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
```

10.13.9 TST and TEQ

Test bits and Test Equivalence.

10.13.9.1 Syntax

```
TST{cond} Rn, Operand2
TEQ{cond} Rn, Operand2
```

where:

cond — is an optional condition code, see “Conditional execution” on page 91.

Rn — is the register holding the first operand.

Operand2 — is a flexible second operand. See “Flexible second operand” on page 86 for details of the options.

10.13.9.2 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.

10.13.9.3 Restrictions

Do not use SP and do not use PC.

10.13.9.4 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” on page 86
- do not affect the V flag.

10.13.9.5 Examples

```
TST     R0, #0x3F8  ; Perform bitwise AND of R0 value to 0x3F8,
                    ; APSR is updated but result is discarded
TEQEQ   R10, R9     ; Conditionally test if value in R10 is equal to
                    ; value in R9, APSR is updated but result is discarded
```

10.14 Multiply and divide instructions

Table 10-21 shows the multiply and divide instructions:

Table 10-21. Multiply and divide instructions

Mnemonic	Brief description	See
MLA	Multiply with Accumulate, 32-bit result	"MUL, MLA, and MLS" on page 124
MLS	Multiply and Subtract, 32-bit result	"MUL, MLA, and MLS" on page 124
MUL	Multiply, 32-bit result	"MUL, MLA, and MLS" on page 124
SDIV	Signed Divide	"SDIV and UDIV" on page 126
SMLAL	Signed Multiply with Accumulate (32x32+64), 64-bit result	"UMULL, UMLAL, SMULL, and SMLAL" on page 125
SMULL	Signed Multiply (32x32), 64-bit result	"UMULL, UMLAL, SMULL, and SMLAL" on page 125
UDIV	Unsigned Divide	"SDIV and UDIV" on page 126
UMLAL	Unsigned Multiply with Accumulate (32x32+64), 64-bit result	"UMULL, UMLAL, SMULL, and SMLAL" on page 125
UMULL	Unsigned Multiply (32x32), 64-bit result	"UMULL, UMLAL, SMULL, and SMLAL" on page 125

10.14.1 MUL, MLA, and MLS

Multiply, Multiply with Accumulate, and Multiply with Subtract, using 32-bit operands, and producing a 32-bit result.

10.14.1.1 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” on page 91.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “Conditional execution” on page 91.

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.

10.14.1.2 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.

10.14.1.3 Restrictions

In these instructions, do not use SP and do not use PC.

If you use the S suffix with the MUL instruction:

- *Rd*, *Rn*, and *Rm* must all be in the range R0 to R7
- *Rd* must be the same as *Rm*
- you must not use the *cond* suffix.

10.14.1.4 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.

10.14.1.5 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)
```

10.14.2 UMULL, UMLAL, SMULL, and SMLAL

Signed and Unsigned Long Multiply, with optional Accumulate, using 32-bit operands and producing a 64-bit result.

10.14.2.1 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” on page 91.

RdHi, *RdLo* are the destination registers.

For UMLAL and SMLAL they also hold the accumulating value.

Rn, Rm are registers holding the operands.

10.14.2.2 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*.

10.14.2.3 Restrictions

In these instructions:

- do not use SP and do not use PC
- *RdHi* and *RdLo* must be different registers.

10.14.2.4 Condition flags

These instructions do not affect the condition code flags.

10.14.2.5 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
```

10.14.3 SDIV and UDIV

Signed Divide and Unsigned Divide.

10.14.3.1 Syntax

```
SDIV{cond} {Rd,} Rn, Rm
UDIV{cond} {Rd,} Rn, Rm
```

where:

cond is an optional condition code, see “Conditional execution” on page 91.

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.

10.14.3.2 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.

10.14.3.3 Restrictions

Do not use SP and do not use PC.

10.14.3.4 Condition flags

These instructions do not change the flags.

10.14.3.5 Examples

```
SDIV  R0, R2, R4  ; Signed divide, R0 = R2/R4
UDIV  R8, R8, R1  ; Unsigned divide, R8 = R8/R1
```

10.15 Saturating instructions

This section describes the saturating instructions, SSAT and USAT.

10.15.1 SSAT and USAT

Signed Saturate and Unsigned Saturate to any bit position, with optional shift before saturating.

10.15.1.1 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” on page 91.

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.

10.15.1.2 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 $-2^{n-1} \le x \le 2^{n-1}-1$.

The USAT instruction applies the specified shift, then saturates to the unsigned range $0 \le x \le 2^{n}-1$.

For signed *n*-bit saturation using SSAT, 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 using USAT, 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, you must use the MSR instruction, see “MSR” on page 147.

To read the state of the Q flag, use the MRS instruction, see “MRS” on page 146.

10.15.1.3 Restrictions

Do not use SP and do not use PC.

10.15.1.4 Condition flags

These instructions do not affect the condition code flags.

If saturation occurs, these instructions set the Q flag to 1.

10.15.1.5 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
```

10.16 Bitfield instructions

Table 10-22 shows the instructions that operate on adjacent sets of bits in registers or bitfields:

Table 10-22. Packing and unpacking instructions

Mnemonic	Brief description	See
BFC	Bit Field Clear	"BFC and BFI" on page 130
BFI	Bit Field Insert	"BFC and BFI" on page 130
SBFX	Signed Bit Field Extract	"SBFX and UBFX" on page 131
SXTB	Sign extend a byte	"SXT and UXT" on page 132
SXTH	Sign extend a halfword	"SXT and UXT" on page 132
UBFX	Unsigned Bit Field Extract	"SBFX and UBFX" on page 131
UXTB	Zero extend a byte	"SXT and UXT" on page 132
UXTH	Zero extend a halfword	"SXT and UXT" on page 132

10.16.1 BFC and BFI

Bit Field Clear and Bit Field Insert.

10.16.1.1 Syntax

```
BFC{cond} Rd, #lsb, #width
BFI{cond} Rd, Rn, #lsb, #width
```

where:

cond — is an optional condition code, see “Conditional execution” on page 91.

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*.

10.16.1.2 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.

10.16.1.3 Restrictions

Do not use SP and do not use PC.

10.16.1.4 Condition flags

These instructions do not affect the flags.

10.16.1.5 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
```

10.16.2 SBFX and UBFX

Signed Bit Field Extract and Unsigned Bit Field Extract.

10.16.2.1 Syntax

```
SBFX{cond} Rd, Rn, #lsb, #width
UBFX{cond} Rd, Rn, #lsb, #width
```

where:

cond — is an optional condition code, see “Conditional execution” on page 91.

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*.

10.16.2.2 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.

10.16.2.3 Restrictions

Do not use SP and do not use PC.

10.16.2.4 Condition flags

These instructions do not affect the flags.

10.16.2.5 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
```

10.16.3 SXT and UXT

Sign extend and Zero extend.

10.16.3.1 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” on page 91.

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.

10.16.3.2 Operation

These instructions do the following:

- Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
- 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.

10.16.3.3 Restrictions

Do not use SP and do not use PC.

10.16.3.4 Condition flags

These instructions do not affect the flags.

10.16.3.5 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
```

10.17 Branch and control instructions

Table 10-23 shows the branch and control instructions:

Table 10-23. Branch and control instructions

Mnemonic	Brief description	See
B	Branch	“B, BL, BX, and BLX” on page 134
BL	Branch with Link	“B, BL, BX, and BLX” on page 134
BLX	Branch indirect with Link	“B, BL, BX, and BLX” on page 134
BX	Branch indirect	“B, BL, BX, and BLX” on page 134
CBNZ	Compare and Branch if Non Zero	“CBZ and CBNZ” on page 136
CBZ	Compare and Branch if Non Zero	“CBZ and CBNZ” on page 136
IT	If-Then	“IT” on page 137
TBB	Table Branch Byte	“TBB and TBH” on page 139
TBH	Table Branch Halfword	“TBB and TBH” on page 139

10.17.1 B, BL, BX, and BLX

Branch instructions.

10.17.1.1 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 “Conditional execution” on page 91.

label is a PC-relative expression. See “PC-relative expressions” on page 90.

Rm is a register that indicates an address to branch to. Bit[0] of the value in *Rm* must be 1, but the address to branch to is created by changing bit[0] to 0.

10.17.1.2 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 cause a UsageFault exception if bit[0] of *Rm* is 0.

B*cond* 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” on page 137.

Table 10-24 shows the ranges for the various branch instructions.

Table 10-24. Branch ranges

Instruction	Branch range
B label	–16 MB to +16 MB
B*cond* label (outside IT block)	–1 MB to +1 MB
B*cond* 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

You might have to use the .W suffix to get the maximum branch range. See “Instruction width selection” on page 92.

10.17.1.3 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.

B*cond* 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.

10.17.1.4 Condition flags

These instructions do not change the flags.

10.17.1.5 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
```

10.17.2 CBZ and CBNZ

Compare and Branch on Zero, Compare and Branch on Non-Zero.

10.17.2.1 Syntax

```
CBZ Rn, label
CBNZ Rn, label
```

where:

Rn is the register holding the operand.

label is the branch destination.

10.17.2.2 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
```

10.17.2.3 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.

10.17.2.4 Condition flags

These instructions do not change the flags.

10.17.2.5 Examples

```
CBZ    R5, target ; Forward branch if R5 is zero
CBNZ   R0, target ; Forward branch if R0 is not zero
```

10.17.3 IT

If-Then condition instruction.

10.17.3.1 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.

10.17.3.2 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.

Your assembler might be able to generate the required IT instructions for conditional instructions automatically, so that you do not need to write them yourself. See your 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.

10.17.3.3 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.

10.17.3.4 Condition flags

This instruction does not change the flags.

10.17.3.5 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
```

10.17.4 TBB and TBH

Table Branch Byte and Table Branch Halfword.

10.17.4.1 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.

10.17.4.2 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.

10.17.4.3 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.

10.17.4.4 Condition flags

These instructions do not change the flags.

10.17.4.5 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
```

10.18 Miscellaneous instructions

Table 10-25 shows the remaining Cortex-M3 instructions:

Table 10-25. Miscellaneous instructions

Mnemonic	Brief description	See
BKPT	Breakpoint	"BKPT" on page 142
CPSID	Change Processor State, Disable Interrupts	"CPS" on page 143
CPSIE	Change Processor State, Enable Interrupts	"CPS" on page 143
DMB	Data Memory Barrier	"DMB" on page 144
DSB	Data Synchronization Barrier	"DSB" on page 144
ISB	Instruction Synchronization Barrier	"ISB" on page 145
MRS	Move from special register to register	"MRS" on page 146
MSR	Move from register to special register	"MSR" on page 147
NOP	No Operation	"NOP" on page 148
SEV	Send Event	"SEV" on page 148
SVC	Supervisor Call	"SVC" on page 149
WFE	Wait For Event	"WFE" on page 150
WFI	Wait For Interrupt	"WFI" on page 151

10.18.1 BKPT

Breakpoint.

10.18.1.1 Syntax

```
BKPT #imm
```

where:

imm is an expression evaluating to an integer in the range 0-255 (8-bit value).

10.18.1.2 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.

10.18.1.3 Condition flags

This instruction does not change the flags.

10.18.1.4 Examples

```
BKPT 0xAB   ; Breakpoint with immediate value set to 0xAB (debugger can
            ; extract the immediate value by locating it using the PC)
```

10.18.2 CPS

Change Processor State.

10.18.2.1 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.

10.18.2.2 Operation

CPS changes the PRIMASK and FAULTMASK special register values. See “Exception mask registers” on page 56 for more information about these registers.

10.18.2.3 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.

10.18.2.4 Condition flags

This instruction does not change the condition flags.

10.18.2.5 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)
```

10.18.3 DMB

Data Memory Barrier.

10.18.3.1 Syntax

`DMB{`*cond*`}`

where:

cond is an optional condition code, see “Conditional execution” on page 91.

10.18.3.2 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.

10.18.3.3 Condition flags

This instruction does not change the flags.

10.18.3.4 Examples

```
DMB  ; Data Memory Barrier
```

10.18.4 DSB

Data Synchronization Barrier.

10.18.4.1 Syntax

`DSB{`*cond*`}`

where:

cond is an optional condition code, see “Conditional execution” on page 91.

10.18.4.2 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.

10.18.4.3 Condition flags

This instruction does not change the flags.

10.18.4.4 Examples

```
DSB ; Data Synchronisation Barrier
```

10.18.5 ISB

Instruction Synchronization Barrier.

10.18.5.1 Syntax

```
ISB{cond}
```

where:

cond is an optional condition code, see “Conditional execution” on page 91.

10.18.5.2 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 memory again, after the ISB instruction has been completed.

10.18.5.3 Condition flags

This instruction does not change the flags.

10.18.5.4 Examples

```
ISB  ; Instruction Synchronisation Barrier
```

10.18.6 MRS

Move the contents of a special register to a general-purpose register.

10.18.6.1 Syntax

```
MRS{cond} Rd, spec_reg
```

where:

cond is an optional condition code, see “Conditional execution” on page 91.

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.

10.18.6.2 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.

BASEPRI_MAX is an alias of BASEPRI when used with the MRS instruction.

See “MSR” on page 147.

10.18.6.3 Restrictions

Rd must not be SP and must not be PC.

10.18.6.4 Condition flags

This instruction does not change the flags.

10.18.6.5 Examples

```
MRS  R0, PRIMASK ; Read PRIMASK value and write it to R0
```

10.18.7 MSR

Move the contents of a general-purpose register into the specified special register.

10.18.7.1 Syntax

```
MSR{cond} spec_reg, Rn
```

where:

cond is an optional condition code, see “Conditional execution” on page 91.

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.

10.18.7.2 Operation

The register access operation in MSR depends on the privilege level. Unprivileged software can only access the APSR, see “Application Program Status Register” on page 54. Privileged software can access all special registers.

In unprivileged software writes to unallocated or execution state bits in the PSR are ignored.

When you write 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” on page 146.

10.18.7.3 Restrictions

Rn must not be SP and must not be PC.

10.18.7.4 Condition flags

This instruction updates the flags explicitly based on the value in *Rn*.

10.18.7.5 Examples

```
MSR  CONTROL, R1 ; Read R1 value and write it to the CONTROL register
```

10.18.8 NOP

No Operation.

10.18.8.1 Syntax

NOP{*cond*}

where:

cond is an optional condition code, see “Conditional execution” on page 91.

10.18.8.2 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.

10.18.8.3 Condition flags

This instruction does not change the flags.

10.18.8.4 Examples

```
NOP  ; No operation
```

10.18.9 SEV

Send Event.

10.18.9.1 Syntax

SEV{*cond*}

where:

cond is an optional condition code, see “Conditional execution” on page 91.

10.18.9.2 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” on page 80.

10.18.9.3 Condition flags

This instruction does not change the flags.

10.18.9.4 Examples

```
SEV ; Send Event
```

10.18.10 SVC

Supervisor Call.

10.18.10.1 Syntax

```
SVC{cond} #imm
```

where:

cond is an optional condition code, see “Conditional execution” on page 91.

imm is an expression evaluating to an integer in the range 0-255 (8-bit value).

10.18.10.2 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.

10.18.10.3 Condition flags

This instruction does not change the flags.

10.18.10.4 Examples

```
SVC  0x32  ; Supervisor Call (SVC handler can extract the immediate value
           ; by locating it via the stacked PC)
```

10.18.11 WFE

Wait For Event.

10.18.11.1 Syntax

```
WFE{cond}
```

where:

cond is an optional condition code, see “Conditional execution” on page 91.

10.18.11.2 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” on page 80.

10.18.11.3 Condition flags

This instruction does not change the flags.

10.18.11.4 Examples

```
WFE  ; Wait for event
```

10.18.12 WFI

Wait for Interrupt.

10.18.12.1 Syntax

```
WFI{cond}
```

where:

cond is an optional condition code, see “Conditional execution” on page 91.

10.18.12.2 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.

10.18.12.3 Condition flags

This instruction does not change the flags.

10.18.12.4 Examples

```
WFI ; Wait for interrupt
```

10.19 About the Cortex-M3 peripherals

The address map of the *Private peripheral bus* (PPB) is:

Table 10-26. Core peripheral register regions

Address	Core peripheral	Description
0xE000E008-0xE000E00F	System control block	Table 10-30 on page 165
0xE000E010-0xE000E01F	System timer	Table 10-33 on page 192
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	Table 10-27 on page 153
0xE000ED00-0xE000ED3F	System control block	Table 10-30 on page 165
0xE000ED90-0xE000EDB8	Memory protection unit	Table 10-35 on page 198
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	Table 10-27 on page 153

In register descriptions:

- the register *type* is described as follows:

  RW Read and write.

  RO Read-only.

  WO Write-only.

- 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.

10.20 Nested Vectored Interrupt Controller

This section describes the Nested Vectored Interrupt Controller (NVIC) and the registers it uses. The NVIC supports:

- 1 to 30 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.

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. The hardware implementation of the NVIC registers is:

Table 10-27. NVIC register summary

Address	Name	Type	Required privilege	Reset value	Description
0xE000E100-0xE000E104	ISER0-ISER1	RW	Privileged	0x00000000	"Interrupt Set-enable Registers" on page 155
0xE000E180-0xE000E184	ICER0-ICER1	RW	Privileged	0x00000000	"Interrupt Clear-enable Registers" on page 156
0xE000E200-0xE000E204	ISPR0-ISPR1	RW	Privileged	0x00000000	"Interrupt Set-pending Registers" on page 157
0xE000E280-0xE000E284	ICPR0-ICPR1	RW	Privileged	0x00000000	"Interrupt Clear-pending Registers" on page 158
0xE000E300-0xE000E304	IABR0-IABR1	RO	Privileged	0x00000000	"Interrupt Active Bit Registers" on page 159
0xE000E400-0xE000E41C	IPR0-IPR7	RW	Privileged	0x00000000	"Interrupt Priority Registers" on page 160
0xE000EF00	STIR	WO	Configurable (1)	0x00000000	"Software Trigger Interrupt Register" on page 162

1. See the register description for more information.

10.20.1 The CMSIS mapping of the Cortex-M3 NVIC registers

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] to ISER[1] corresponds to the registers ISER0-ISER1
  - the array ICER[0] to ICER[1] corresponds to the registers ICER0-ICER1
  - the array ISPR[0] to ISPR[1] corresponds to the registers ISPR0-ISPR1
  - the array ICPR[0] to ICPR[1] corresponds to the registers ICPR0-ICPR1
  - the array IABR[0] to IABR[1] corresponds to the registers IABR0-IABR1
- 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[29] corresponds to the registers IPR0-IPR7, 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. For more information see the description of the NVIC_SetPriority function in "NVIC programming hints" on page 164. Table 10-28 shows how the interrupts, or IRQ numbers, map onto the interrupt registers and corresponding CMSIS variables that have one bit per interrupt.

Table 10-28. Mapping of interrupts to the interrupt variables

Interrupts	CMSIS array elements (1)				
	Set-enable	**Clear-enable**	**Set-pending**	**Clear-pending**	**Active Bit**
0-29	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
30-63	ISER[1]	ICER[1]	ISPR[1]	ICPR[1]	IABR[1]

1. Each array element corresponds to a single NVIC register, for example the element ICER[0] corresponds to the ICER0 register.

10.20.2 Interrupt Set-enable Registers

The ISER0-ISER1 register enables interrupts, and show which interrupts are enabled. See:

- the register summary in Table 10-27 on page 153 for the register attributes
- Table 10-28 on page 154 for which interrupts are controlled by each register.

The bit assignments are:

31	30	29	28	27	26	25	24
SETENA bits							

23	22	21	20	19	18	17	16
SETENA bits							

15	14	13	12	11	10	9	8
SETENA bits							

7	6	5	4	3	2	1	0
SETENA bits							

- **SETENA**

Interrupt set-enable bits.

Write:

0: no effect

1: enable interrupt.

Read:

0: interrupt disabled

1: interrupt enabled.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

10.20.3 Interrupt Clear-enable Registers

The ICER0-ICER1 register disables interrupts, and shows which interrupts are enabled. See:

- the register summary in Table 10-27 on page 153 for the register attributes
- Table 10-28 on page 154 for which interrupts are controlled by each register

The bit assignments are:

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							

- **CLRENA**

Interrupt clear-enable bits.

Write:

0: no effect

1: disable interrupt.

Read:

0: interrupt disabled

1: interrupt enabled.

10.20.4 Interrupt Set-pending Registers

The ISPR0-ISPR1 register forces interrupts into the pending state, and shows which interrupts are pending. See:

- the register summary in Table 10-27 on page 153 for the register attributes
- Table 10-28 on page 154 for which interrupts are controlled by each register.

The bit assignments are:

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

- **SETPEND**

Interrupt set-pending bits.

Write:

0: no effect.

1: changes interrupt state to pending.

Read:

0: interrupt is not pending.

1: interrupt is pending.

Writing 1 to the ISPR bit corresponding to:

- an interrupt that is pending has no effect
- a disabled interrupt sets the state of that interrupt to pending

10.20.5 Interrupt Clear-pending Registers

The ICPR0-ICPR1 register removes the pending state from interrupts, and show which interrupts are pending. See:

- the register summary in Table 10-27 on page 153 for the register attributes
- Table 10-28 on page 154 for which interrupts are controlled by each register.

The bit assignments are:

31	30	29	28	27	26	25	24
CLRPEND							

23	22	21	20	19	18	17	16
CLRPEND							

15	14	13	12	11	10	9	8
CLRPEND							

7	6	5	4	3	2	1	0
CLRPEND							

- **CLRPEND**

Interrupt clear-pending bits.

Write:

0: no effect.

1: removes pending state an interrupt.

Read:

0: interrupt is not pending.

1: interrupt is pending.

Writing 1 to an ICPR bit does not affect the active state of the corresponding interrupt.

10.20.6 Interrupt Active Bit Registers

The IABR0-IABR1 register indicates which interrupts are active. See:

- the register summary in Table 10-27 on page 153 for the register attributes
- Table 10-28 on page 154 for which interrupts are controlled by each register.

The bit assignments are:

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							

- **ACTIVE**

Interrupt active flags:

0: interrupt not active

1: interrupt active.

A bit reads as one if the status of the corresponding interrupt is active or active and pending.

10.20.7 Interrupt Priority Registers

The IPR0-IPR7 registers provide a 4-bit priority field for each interrupt (See the “Peripheral Identifiers” section of the datasheet for more details). These registers are byte-accessible. See the register summary in Table 10-27 on page 153 for their attributes. Each register holds four priority fields, that map up to four elements in the CMSIS interrupt priority array IP[0] to IP[29], as shown:

10.20.7.1 IPRm

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

10.20.7.2 IPR4

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

10.20.7.3 IPR3

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

10.20.7.4 IPR2

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

10.20.7.5 IPR1

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

10.20.7.6 IPR0

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

- **Priority, byte offset 3**
- **Priority, byte offset 2**
- **Priority, byte offset 1**
- **Priority, byte offset 0**

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.

See “The CMSIS mapping of the Cortex-M3 NVIC registers” on page 153 for more information about the IP[0] to IP[29] interrupt priority array, that provides the software view of the interrupt priorities.

Find the IPR number and byte offset for interrupt *N* as follows:

- the corresponding IPR number, *M*, is given by *M* = *N* DIV 4
- the byte offset of the required Priority field in this register is *N* MOD 4, where:
  - byte offset 0 refers to register bits[7:0]
  - byte offset 1 refers to register bits[15:8]
  - byte offset 2 refers to register bits[23:16]
  - byte offset 3 refers to register bits[31:24].

10.20.8 Software Trigger Interrupt Register

Write to the STIR to generate a *Software Generated Interrupt* (SGI). See the register summary in Table 10-27 on page 153 for the STIR attributes.

When the USERSETMPEND bit in the SCR is set to 1, unprivileged software can access the STIR, see "System Control Register" on page 174.

Only privileged software can enable unprivileged access to the STIR.

The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							INTID

7	6	5	4	3	2	1	0
INTID							

- **INTID**

Interrupt ID of the required SGI, in the range 0-239. For example, a value of b000000011 specifies interrupt IRQ3.

10.20.9 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 needs servicing.

10.20.9.1 Hardware and software control of interrupts

The Cortex-M3 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
- software writes to the corresponding interrupt set-pending register bit, see “Interrupt Set-pending Registers” on page 157, or to the STIR to make an SGI pending, see “Software Trigger Interrupt Register” on page 162.

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.
  - If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR 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.

10.20.10 NVIC design hints and tips

Ensure 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 pending state even it is disabled.

Before programming VTOR to relocate the vector table, ensure the vector table entries of the new vector table are setup for fault handlers and all enabled exception like interrupts. For more information see “Vector Table Offset Register” on page 171.

10.20.10.1 NVIC programming hints

Software uses the CPSIE I and CPSID I instructions to enable and disable 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 10-29. 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 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

For more information about these functions see the CMSIS documentation.

10.21 System control block

The *System control block* (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The system control block registers are:

Table 10-30. Summary of the system control block registers

Address	Name	Type	Required privilege	Reset value	Description
0xE000E008	ACTLR	RW	Privileged	0x00000000	"Auxiliary Control Register" on page 166
0xE000ED00	CPUID	RO	Privileged	0x412FC230	"CPUID Base Register" on page 167
0xE000ED04	ICSR	RW(1)	Privileged	0x00000000	"Interrupt Control and State Register" on page 168
0xE000ED08	VTOR	RW	Privileged	0x00000000	"Vector Table Offset Register" on page 171
0xE000ED0C	AIRCR	RW(1)	Privileged	0xFA050000	"Application Interrupt and Reset Control Register" on page 172
0xE000ED10	SCR	RW	Privileged	0x00000000	"System Control Register" on page 174
0xE000ED14	CCR	RW	Privileged	0x00000200	"Configuration and Control Register" on page 175
0xE000ED18	SHPR1	RW	Privileged	0x00000000	"System Handler Priority Register 1" on page 178
0xE000ED1C	SHPR2	RW	Privileged	0x00000000	"System Handler Priority Register 2" on page 179
0xE000ED20	SHPR3	RW	Privileged	0x00000000	"System Handler Priority Register 3" on page 179
0xE000ED24	SHCRS	RW	Privileged	0x00000000	"System Handler Control and State Register" on page 180
0xE000ED28	CFSR	RW	Privileged	0x00000000	"Configurable Fault Status Register" on page 182
0xE000ED28	MMSR(2)	RW	Privileged	0x00	"Memory Management Fault Address Register" on page 189
0xE000ED29	BFSR(2)	RW	Privileged	0x00	"Bus Fault Status Register" on page 184
0xE000ED2A	UFSR(2)	RW	Privileged	0x0000	"Usage Fault Status Register" on page 186
0xE000ED2C	HFSR	RW	Privileged	0x00000000	"Hard Fault Status Register" on page 188
0xE000ED34	MMAR	RW	Privileged	Unknown	"Memory Management Fault Address Register" on page 189
0xE000ED38	BFAR	RW	Privileged	Unknown	"Bus Fault Address Register" on page 190

Notes: 1. See the register description for more information.
2. A subregister of the CFSR.

10.21.1 The CMSIS mapping of the Cortex-M3 SCB registers

To improve software efficiency, the CMSIS simplifies the SCB register presentation. In the CMSIS, the byte array SHP[0] to SHP[12] corresponds to the registers SHPR1-SHPR3.

10.21.2 Auxiliary Control Register

The ACTLR 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.

See the register summary in Table 10-30 on page 165 for the ACTLR attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved					DISFOLD	DISDEFWBUF	DISMCYCINT

- **DISFOLD**

When set to 1, disables IT folding. see "About IT folding" on page 166 for more information.

- **DISDEFWBUF**

When set to 1, disables write buffer use during default memory map accesses. This causes all bus faults to be precise bus faults but decreases performance because any store to memory must complete before the processor can execute the next instruction.

This bit only affects write buffers implemented in the Cortex-M3 processor.

- **DISMCYCINT**

When set to 1, disables interruption of load multiple and store multiple instructions. This increases the interrupt latency of the processor because any LDM or STM must complete before the processor can stack the current state and enter the interrupt handler.

10.21.2.1 About IT folding

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 improves 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 IT folding.

10.21.3 CPUID Base Register

The CPUID register contains the processor part number, version, and implementation information. See the register summary in Table 10-30 on page 165 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Implementer							

23	22	21	20	19	18	17	16
Variant				Constant			

15	14	13	12	11	10	9	8
PartNo							

7	6	5	4	3	2	1	0
PartNo				Revision			

- **Implementer**

Implementer code:

0x41 = ARM

- **Variant**

Variant number, the r value in the r*n*p*n* product revision identifier:

0x2 = r2p0

- **Constant**

Reads as 0xF

- **PartNo**

Part number of the processor:

0xC23 = Cortex-M3

- **Revision**

Revision number, the p value in the r*n*p*n* product revision identifier:

0x0 = r2p0

10.21.4 Interrupt Control and State Register

The ICSR:

- provides:
  - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- indicates:
  - the exception number of the exception being processed
  - whether there are preempted active exceptions
  - the exception number of the highest priority pending exception
  - whether any interrupts are pending.

See the register summary in Table 10-30 on page 165, and the Type descriptions in Table 10-33 on page 192, for the ICSR attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved	Reserved		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	Reserved

23	22	21	20	19	18	17	16
Reserved for Debug	ISRPENDING	VECTPENDING					

15	14	13	12	11	10	9	8
VECTPENDING				RETTOBASE	Reserved		VECTACTIVE

7	6	5	4	3	2	1	0
VECTACTIVE							

- **PENDSVSET**

RW

PendSV set-pending bit.

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**

WO

PendSV clear-pending bit.

Write:

0: no effect

1: removes the pending state from the PendSV exception.

- **PENDSTSET**

RW

SysTick exception set-pending bit.

Write:

0: no effect

1: changes SysTick exception state to pending.

Read:

0: SysTick exception is not pending

1: SysTick exception is pending.

- **PENDSTCLR**

WO

SysTick exception clear-pending bit.

Write:

0: no effect

1: removes the pending state from the SysTick exception.

This bit is WO. On a register read its value is Unknown.

- **Reserved for Debug use**

RO

This bit is reserved for Debug use and reads-as-zero when the processor is not in Debug.

- **ISRPENDING**

RO

Interrupt pending flag, excluding Faults:

0: interrupt not pending

1: interrupt pending.

- **VECTPENDING**

RO

Indicates the 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**

RO

Indicates whether there are preempted active exceptions:

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**

RO

Contains the active exception number:

0: Thread mode

Nonzero = The exception number [(1)] of the currently active exception.

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” on page 54.

When you write to the ICSR, the effect is Unpredictable if you:

- write 1 to the PENDSVSET bit and write 1 to the PENDSVCLR bit
- write 1 to the PENDSTSET bit and write 1 to the PENDSTCLR bit.

Note: 1. This is the same value as IPSR bits [8:0] see “Interrupt Program Status Register” on page 54.

10.21.5 Vector Table Offset Register

The VTOR indicates the offset of the vector table base address from memory address 0x00000000. See the register summary in Table 10-30 on page 165 for its attributes.

The bit assignments are:

31	30	29	28	27	26	25	24
Reserved		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	Reserved						

- **TBLOFF**

Vector table base offset field. 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.

Bit[29] is sometimes called the TBLBASE bit.

When setting TBLOFF, you must align the offset to the number of exception entries in the vector table. 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 you require 21 interrupts, 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.

10.21.6 Application Interrupt and Reset Control Register

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. See the register summary in Table 10-30 on page 165 and Table 10-33 on page 192 for its attributes.

To write to this register, you must write 0x05FA to the VECTKEY field, otherwise the processor ignores the write.

The bit assignments are:

31	30	29	28	27	26	25	24
On Read: VECTKEYSTAT, On Write: VECTKEY							

23	22	21	20	19	18	17	16
On Read: VECTKEYSTAT, On Write: VECTKEY							

15	14	13	12	11	10	9	8
ENDIANESS	Reserved				PRIGROUP		

7	6	5	4	3	2	1	0
Reserved					SYSRESETREQ	VECTCLR-ACTIVE	VECTRESET

- **VECTKEYSTAT**

Register Key:

Reads as 0xFA05

- **VECTKEY**

Register key:

On writes, write 0x5FA to VECTKEY, otherwise the write is ignored.

- **ENDIANESS**

RO

Data endianness bit:

0: Little-endian

ENDIANESS is set from the **BIGEND** configuration signal during reset.

- **PRIGROUP**

R/W

Interrupt priority grouping field. This field determines the split of group priority from subpriority, see “Binary point” on page 173.

- **SYSRESETREQ**

WO

System reset request:

0: no effect

1: asserts a proc_reset_signal.

This is intended to force a large system reset of all major components except for debug.

This bit reads as 0.

- **VECTCLRACTIVE**

WO

Reserved for Debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.

- **VECTRESET**

WO

Reserved for Debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.

10.21.6.1 Binary point

The PRIGROUP field indicates the position of the binary point that splits the PRI_*n* fields in the Interrupt Priority Registers into separate *group priority* and *subpriority* fields. Table 10-31 shows how the PRIGROUP value controls this split.

Table 10-31. Priority grouping

	Interrupt priority level value, PRI_*N*[7:0]			Number of	
PRIGROUP	**Binary point [1]**	**Group priority bits**	**Subpriority bits**	**Group priorities**	**Subpriorities**
b011	bxxxx.0000	[7:4]	None	16	1
b100	bxxx.y0000	[7:5]	[4]	8	2
b101	bxx.yy0000	[7:6]	[5:4]	4	4
b110	bx.yyy0000	[7]	[6:4]	2	8
b111	b.yyyy0000	None	[7:4]	1	16

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, see “Interrupt priority grouping” on page 75.

10.21.7 System Control Register

The SCR controls features of entry to and exit from low power state. See the register summary in Table 10-30 on page 165 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved			SEVONPEND	Reserved	SLEEPDEEP	SLEEONEXIT	Reserved

- **SEVONPEND**

Send Event on Pending bit:

0: only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded

1: enabled events and all interrupts, including disabled interrupts, can wakeup the processor.

When an event or interrupt enters 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**

Controls whether the processor uses sleep or deep sleep as its low power mode:

0: sleep

1: deep sleep.

- **SLEEPONEXIT**

Indicates sleep-on-exit when returning from Handler mode to 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.

10.21.8 Configuration and Control Register

The CCR controls entry to Thread mode and enables:

- the handlers for hard fault and faults escalated by FAULTMASK to ignore bus faults
- trapping of divide by zero and unaligned accesses
- access to the STIR by unprivileged software, see “Software Trigger Interrupt Register” on page 162.

See the register summary in Table 10-30 on page 165 for the CCR attributes.

The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved						STKALIGN	BFHFNMIGN

7	6	5	4	3	2	1	0
Reserved			DIV_0_TRP	UNALIGN_TRP	Reserved	USERSETMPEND	NONBASETHRDENA

- **STKALIGN**

Indicates 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.

- **BFHFNMIGN**

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**

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**

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 STIR, see “Software Trigger Interrupt Register” on page 162:

0: disable

1: enable.

- **NONEBASETHRDENA**

Indicates how the processor enters Thread mode:

0: processor can enter Thread mode only when no exception is active.

1: processor can enter Thread mode from any level under the control of an EXC_RETURN value, see “Exception return” on page 76.

10.21.9 System Handler Priority Registers

The SHPR1-SHPR3 registers set the priority level, 0 to 15 of the exception handlers that have configurable priority.

SHPR1-SHPR3 are byte accessible. See the register summary in Table 10-30 on page 165 for their attributes.

The system fault handlers and the priority field and register for each handler are:

Table 10-32. System fault handler priority fields

Handler	Field	Register description
Memory management fault	PRI_4	"System Handler Priority Register 1" on page 178
Bus fault	PRI_5	
Usage fault	PRI_6	
SVCall	PRI_11	"System Handler Priority Register 2" on page 179
PendSV	PRI_14	"System Handler Priority Register 3" on page 179
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.

10.21.9.1 System Handler Priority Register 1

The bit assignments are:

31	30	29	28	27	26	25	24
PRI_7: Reserved							
23	22	21	20	19	18	17	16
PRI_6							
15	14	13	12	11	10	9	8
PRI_5							
7	6	5	4	3	2	1	0
PRI_4							

- **PRI_7**

Reserved

- **PRI_6**

Priority of system handler 6, usage fault

- **PRI_5**

Priority of system handler 5, bus fault

- **PRI_4**

Priority of system handler 4, memory management fault

10.21.9.2 System Handler Priority Register 2

The bit assignments are:

31	30	29	28	27	26	25	24
PRI_11							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
Reserved							

- **PRI_11**

Priority of system handler 11, SVCall

10.21.9.3 System Handler Priority Register 3

The bit assignments are:

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
Reserved							
7	6	5	4	3	2	1	0
Reserved							

- **PRI_15**

Priority of system handler 15, SysTick exception

- **PRI_14**

Priority of system handler 14, PendSV

10.21.10 System Handler Control and State Register

The SHCSR enables the system handlers, and indicates:

- the pending status of the bus fault, memory management fault, and SVC exceptions
- the active status of the system handlers.

See the register summary in Table 10-30 on page 165 for the SHCSR attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved					USGFAULTENA	BUSFAULTENA	MEMFAULTENA

15	14	13	12	11	10	9	8
SVCALLPENDED	BUSFAULTPENDED	MEMFAULTPENDED	USGFAULTPENDED	SYSTICKACT	PENDSVACT	Reserved	MONITORACT

7	6	5	4	3	2	1	0
SVCALLAVCT	Reserved			USGFAULTACT	Reserved	BUSFAULTACT	MEMFAULTACT

- **USGFAULTENA**

Usage fault enable bit, set to 1 to enable (1)

- **BUSFAULTENA**

Bus fault enable bit, set to 1 to enable (3)

- **MEMFAULTENA**

Memory management fault enable bit, set to 1 to enable (3)

- **SVCALLPENDED**

SVC call pending bit, reads as 1 if exception is pending (2)

- **BUSFAULTPENDED**

Bus fault exception pending bit, reads as 1 if exception is pending (2)

- **MEMFAULTPENDED**

Memory management fault exception pending bit, reads as 1 if exception is pending (2)

- **USGFAULTPENDED**

Usage fault exception pending bit, reads as 1 if exception is pending (2)

- **SYSTICKACT**

SysTick exception active bit, reads as 1 if exception is active (3)

- **PENDSVACT**

PendSV exception active bit, reads as 1 if exception is active

1. Enable bits, set to 1 to enable the exception, or set to 0 to disable the exception.
2. Pending bits, read as 1 if the exception is pending, or as 0 if it is not pending. You can write to these bits to change the pending status of the exceptions.
3. Active bits, read as 1 if the exception is active, or as 0 if it is not active. You can write to these bits to change the active status of the exceptions, but see the Caution in this section.

- **MONITORACT**

Debug monitor active bit, reads as 1 if Debug monitor is active

- **SVCALLACT**

SVC call active bit, reads as 1 if SVC call is active

- **USGFAULTACT**

Usage fault exception active bit, reads as 1 if exception is active

- **BUSFAULTACT**

Bus fault exception active bit, reads as 1 if exception is active

- **MEMFAULTACT**

Memory management fault exception active bit, reads as 1 if exception is active

If you disable a system handler and the corresponding fault occurs, the processor treats the fault as a hard fault.

You 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.

- Software that changes the value of an active bit in this register without correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure software that writes to this register retains and subsequently restores the current active status.
- After you have enabled the system handlers, if you have to change the value of a bit in this register you must use a read-modify-write procedure to ensure that you change only the required bit.

10.21.11 Configurable Fault Status Register

The CFSR indicates the cause of a memory management fault, bus fault, or usage fault. See the register summary in Table 10-30 on page 165 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Usage Fault Status Register: UFSR							

23	22	21	20	19	18	17	16
Usage Fault Status Register: UFSR							

15	14	13	12	11	10	9	8
Bus Fault Status Register: BFSR							

7	6	5	4	3	2	1	0
Memory Management Fault Status Register: MMFSR							

The following subsections describe the subregisters that make up the CFSR:

- “Memory Management Fault Status Register” on page 183
- “Bus Fault Status Register” on page 184
- “Usage Fault Status Register” on page 186.

The CFSR is byte accessible. You can access the CFSR or its subregisters as follows:

- access the complete CFSR with a word access to 0xE000ED28
- access the MMFSR with a byte access to 0xE000ED28
- access the MMFSR and BFSR with a halfword access to 0xE000ED28
- access the BFSR with a byte access to 0xE000ED29
- access the UFSR with a halfword access to 0xE000ED2A.

10.21.11.1 Memory Management Fault Status Register

The flags in the MMFSR indicate the cause of memory access faults. The bit assignments are:

7	6	5	4	3	2	1	0
MMARVALID	Reserved		MSTKERR	MUNSTKERR	Reserved	DACCVIOL	IACCVIOL

- **MMARVALID**

Memory Management Fault Address Register (MMAR) valid flag:

0: value in MMAR is not a valid fault address

1: MMAR 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 MMAR value has been overwritten.

- **MSTKERR**

Memory manager fault on stacking for exception entry:

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 the MMAR.

- **MUNSTKERR**

Memory manager fault on unstacking for a return from exception:

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 MMAR.

- **DACCVIOL**

Data access violation flag:

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 MMAR with the address of the attempted access.

- **IACCVIOL**

Instruction access violation flag:

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 MMAR.

10.21.11.2 Bus Fault Status Register

The flags in the BFSR indicate the cause of a bus access fault. The bit assignments are:

7	6	5	4	3	2	1	0
BFRVALID	Reserved		STKERR	UNSTKERR	IMPRECISERR	PRECISERR	IBUSERR

- **BFARVALID**

Bus Fault Address Register (BFAR) valid flag:

0: value in BFAR is not a valid fault address

1: 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 BFAR value has been overwritten.

- **STKERR**

Bus fault on stacking for exception entry:

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 BFAR.

- **UNSTKERR**

Bus fault on unstacking for a return from exception:

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 performed a new save, and does not write a fault address to the BFAR.

- **IMPRECISERR**

Imprecise data bus error:

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 BFAR.

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 both IMPRECISERR set to 1 and one of the precise fault status bits set to 1.

- **PRECISERR**

Precise data bus error:

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 is 1, it writes the faulting address to the BFAR.

- **IBUSERR**

Instruction bus error:

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 is 1, it does not write a fault address to the BFAR.

10.21.11.3 Usage Fault Status Register

The UFSR indicates the cause of a usage fault. The bit assignments are:

15	14	13	12	11	10	9	8
Reserved						DIVBYZERO	UNALIGNED

7	6	5	4	3	2	1	0
Reserved				NOCP	INVPC	INVSTATE	UNDEFINSTR

- **DIVBYZERO**

Divide by zero usage fault:

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 CCR to 1, see “Configuration and Control Register” on page 175.

- **UNALIGNED**

Unaligned access usage fault:

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 CCR to 1, see “Configuration and Control Register” on page 175.

Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of the setting of UNALIGN_TRP.

- **NOCP**

No coprocessor usage fault. 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.

- **INVPC**

Invalid PC load usage fault, 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.

- **INVSTATE**

Invalid state usage fault:

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.

- **UNDEFINSTR**

Undefined instruction usage fault:

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.

The UFSR bits are sticky. This means 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.

10.21.12 Hard Fault Status Register

The HFSR gives information about events that activate the hard fault handler. See the register summary in Table 10-30 on page 165 for its attributes.

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. The bit assignments are:

31	30	29	28	27	26	25	24
DEBUGEVT	FORCED	Reserved					

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved						VECTTBL	Reserved

- **DEBUGEVT**

Reserved for Debug use. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.

- **FORCED**

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**

Indicates a bus fault on a vector table read during 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.

The HFSR bits are sticky. This means 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.

10.21.13 Memory Management Fault Address Register

The MMFAR contains the address of the location that generated a memory management fault. See the register summary in Table 10-30 on page 165 for its attributes. The bit assignments are:

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

- **ADDRESS**

When the MMARVALID bit of the MMFSR is set to 1, this field holds the address of the location that generated the memory management fault

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.

Flags in the MMFSR indicate the cause of the fault, and whether the value in the MMFAR is valid. See “Memory Management Fault Status Register” on page 183.

10.21.14 Bus Fault Address Register

The BFAR contains the address of the location that generated a bus fault. See the register summary in Table 10-30 on page 165 for its attributes. The bit assignments are:

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

- **ADDRESS**

When the BFARVALID bit of the BFSR is set to 1, this field holds the address of the location that generated the bus fault

When an unaligned access faults the address in the BFAR is the one requested by the instruction, even if it is not the address of the fault.

Flags in the BFSR indicate the cause of the fault, and whether the value in the BFAR is valid. See "Bus Fault Status Register" on page 184.

10.21.15 System control block design hints and tips

Ensure software uses aligned accesses of the correct size to access the system control block registers:

- except for the CFSR and SHPR1-SHPR3, it must use aligned word accesses
- for the CFSR and SHPR1-SHPR3 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:

- Read and save the MMFAR or BFAR value.
- Read the MMARVALID bit in the MMFSR, or the BFARVALID bit in the BFSR. The MMFAR or BFAR address is valid only if this bit is 1.

Software must follow this sequence because another higher priority exception might change the MMFAR or BFAR value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the MMFAR or BFAR value.

10.22 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 LOAD 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 system timer registers are:

Table 10-33. System timer registers summary

Address	Name	Type	Required privilege	Reset value	Description
0xE000E010	CTRL	RW	Privileged	0x00000004	“SysTick Control and Status Register” on page 193
0xE000E014	LOAD	RW	Privileged	0x00000000	“SysTick Reload Value Register” on page 194
0xE000E018	VAL	RW	Privileged	0x00000000	“SysTick Current Value Register” on page 195
0xE000E01C	CALIB	RO	Privileged	0x0002904 (1)	“SysTick Calibration Value Register” on page 196

1. SysTick calibration value.

10.22.1 SysTick Control and Status Register

The SysTick CTRL register enables the SysTick features. See the register summary in Table 10-33 on page 192 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							COUNTFLAG

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved					CLKSOURCE	TICKINT	ENABLE

- **COUNTFLAG**

Returns 1 if timer counted to 0 since last time this was read.

- **CLKSOURCE**

Indicates the clock source:

0: MCK/8

1: MCK

- **TICKINT**

Enables SysTick exception request:

0: counting down to zero does not assert the SysTick exception request

1: counting down to zero to asserts the SysTick exception request.

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 LOAD 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.

10.22.2 SysTick Reload Value Register

The LOAD register specifies the start value to load into the VAL register. See the register summary in Table 10-33 on page 192 for its attributes. The bit assignments are:

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

- **RELOAD**

Value to load into the VAL register when the counter is enabled and when it reaches 0, see "Calculating the RELOAD value" .

10.22.2.1 Calculating the RELOAD value

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:

- To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.
- To deliver a single SysTick interrupt after a delay of N processor clock cycles, use a RELOAD of value N. For example, if a SysTick interrupt is required after 400 clock pulses, set RELOAD to 400.

10.22.3 SysTick Current Value Register

The VAL register contains the current value of the SysTick counter. See the register summary in Table 10-33 on page 192 for its attributes. The bit assignments are:

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

- **CURRENT**

Reads return the current value of the SysTick counter.

A write of any value clears the field to 0, and also clears the SysTick CTRL.COUNTFLAG bit to 0.

10.22.4 SysTick Calibration Value Register

The CALIB register indicates the SysTick calibration properties. See the register summary in Table 10-33 on page 192 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
NOREF	SKEW	Reserved					

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							

- **NOREF**

Reads as zero.

- **SKEW**

Reads as zero

- **TENMS**

Read as 0x0002904. The SysTick calibration value is fixed at 0x0002904 (10500), which allows the generation of a time base of 1 ms with SysTick clock at 10.5 MHz (84/8 = 10.5 MHz)

10.22.5 SysTick design hints and tips

The SysTick counter runs on the processor clock. If this clock signal is stopped for low power mode, the SysTick counter stops.

Ensure software uses aligned word accesses to access the SysTick registers.

10.23 Memory protection unit

This section describes the *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-M3 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-M3 MPU memory map is unified. This means instruction accesses and data accesses have 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 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.

Configuration of MPU regions is based on memory types, see “Memory regions, types and attributes” on page 62.

Table 10-34 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” on page 209 for guidelines for programming such an implementation.

Table 10-34. 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		Normal memory that is shared between several processors.
	Non-shared		Normal memory that only a single processor uses.

Use the MPU registers to define the MPU regions and their attributes. The MPU registers are:

Table 10-35. MPU registers summary

Address	Name	Type	Required privilege	Reset value	Description
0xE000ED90	TYPE	RO	Privileged	0x00000800	“MPU Type Register” on page 199
0xE000ED94	CTRL	RW	Privileged	0x00000000	“MPU Control Register” on page 200
0xE000ED98	RNR	RW	Privileged	0x00000000	“MPU Region Number Register” on page 202
0xE000ED9C	RBAR	RW	Privileged	0x00000000	“MPU Region Base Address Register” on page 203
0xE000EDA0	RASR	RW	Privileged	0x00000000	“MPU Region Attribute and Size Register” on page 204
0xE000EDA4	RBAR_A1	RW	Privileged	0x00000000	Alias of RBAR, see “MPU Region Base Address Register” on page 203
0xE000EDA8	RASR_A1	RW	Privileged	0x00000000	Alias of RASR, see “MPU Region Attribute and Size Register” on page 204
0xE000EDAC	RBAR_A2	RW	Privileged	0x00000000	Alias of RBAR, see “MPU Region Base Address Register” on page 203
0xE000EDB0	RASR_A2	RW	Privileged	0x00000000	Alias of RASR, see “MPU Region Attribute and Size Register” on page 204
0xE000EDB4	RBAR_A3	RW	Privileged	0x00000000	Alias of RBAR, see “MPU Region Base Address Register” on page 203
0xE000EDB8	RASR_A3	RW	Privileged	0x00000000	Alias of RASR, see “MPU Region Attribute and Size Register” on page 204

10.23.1 MPU Type Register

The TYPE register indicates whether the MPU is present, and if so, how many regions it supports. See the register summary in Table 10-35 on page 198 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

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
Reserved							SEPARATE

- **IREGION**

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**

Indicates the number of supported MPU data regions:

0x08 = Eight MPU regions.

- **SEPARATE**

Indicates support for unified or separate instruction and date memory maps:

0: unified.

10.23.2 MPU Control Register

The MPU CTRL register:

- enables the MPU
- enables the default memory map background region
- enables use of the MPU when in the hard fault, *Non-maskable Interrupt* (NMI), and FAULTMASK escalated handlers.

See the register summary in Table 10-35 on page 198 for the MPU CTRL attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved					PRIVDEFENA	HFNMIENA	ENABLE

- **PRIVDEFENA**

Enables privileged software access to the default memory map:

0: If the MPU is enabled, disables 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 use of the default memory map as a background region for privileged software accesses.

When enabled, the background region acts as if it is 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**

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" on page 62. 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, see Table 10-34 on page 197. 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 HFNMIENA 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 HFNMIENA bit to 1 enables the MPU when operating with these two priorities.

10.23.3 MPU Region Number Register

The RNR selects which memory region is referenced by the RBAR and RASR registers. See the register summary in Table 10-35 on page 198 for its attributes. The bit assignments are:

31	30	29	28	27	26	25	24
Reserved							

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
REGION							

- **REGION**

Indicates the MPU region referenced by the RBAR and RASR registers.

The MPU supports 8 memory regions, so the permitted values of this field are 0-7.

Normally, you write the required region number to this register before accessing the RBAR or RASR. However you can change the region number by writing to the RBAR with the VALID bit set to 1, see "MPU Region Base Address Register" on page 203. This write updates the value of the REGION field.

10.23.4 MPU Region Base Address Register

The RBAR defines the base address of the MPU region selected by the RNR, and can update the value of the RNR. See the register summary in Table 10-35 on page 198 for its attributes.

Write RBAR with the VALID bit set to 1 to change the current region number and update the RNR. The bit assignments are:

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
Reserved			VALID	REGION			

- **ADDR**

Region base address field. The value of N depends on the region size. For more information see “The ADDR field” .

- **VALID**

MPU Region Number valid bit:

Write:

0: RNR not changed, and the processor:

updates the base address for the region specified in the RNR

ignores the value of the REGION field

1: the processor:

updates the value of the RNR to the value of the REGION field

updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION**

MPU region field:

For the behavior on writes, see the description of the VALID field.

On reads, returns the current region number, as specified by the RNR.

10.23.4.1 The ADDR field

The ADDR field is bits[31:N] of the RBAR. The region size, as specified by the SIZE field in the RASR, defines the value of N:

$$N = Log_2(\text{Region size in bytes}),$$

If the region size is configured to 4GB, in the 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 64KB region must be aligned on a multiple of 64KB, for example, at 0x00010000 or 0x00020000.

10.23.5 MPU Region Attribute and Size Register

The RASR defines the region size and memory attributes of the MPU region specified by the RNR, and enables that region and any subregions. See the register summary in Table 10-35 on page 198 for its attributes.

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.

The bit assignments are:

31	30	29	28	27	26	25	24
Reserved			XN	Reserved	AP		

23	22	21	20	19	18	17	16
Reserved		TEX			S	C	B

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
Reserved		SIZE					ENABLE

- **XN**

Instruction access disable bit:

0: instruction fetches enabled

1: instruction fetches disabled.

- **AP**

Access permission field, see Table 10-39 on page 207.

- **TEX, C, B**

Memory access attributes, see Table 10-37 on page 206.

- **S**

Shareable bit, see Table 10-36 on page 205.

- **SRD**

Subregion disable bits. For each bit in this field:

0: corresponding sub-region is enabled

1: corresponding sub-region is disabled

See “Subregions” on page 208 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**

Specifies the size of the MPU protection region. The minimum permitted value is 3 (b00010), see See “SIZE field values” on page 205 for more information.

- **ENABLE**

Region enable bit.

For information about access permission, see “MPU access permission attributes” .

10.23.5.1 SIZE field values

The SIZE field defines the size of the MPU memory region specified by the 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. Table 10-36 gives example SIZE values, with the corresponding region size and value of N in the RBAR.

Table 10-36. Example SIZE field values

SIZE value	Region size	Value of N (1)	Note
b00100 (4)	32B	5	Minimum permitted size
b01001 (9)	1KB	10	-
b10011 (19)	1MB	20	-
b11101 (29)	1GB	30	-
b11111 (31)	4GB	b01100	Maximum possible size

1. In the RBAR, see “MPU Region Base Address Register” on page 203.

10.23.6 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 RASR, control 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.

Table 10-37 shows the encodings for the TEX, C, B, and S access permission bits.

Table 10-37. TEX, C, B, and S encoding

TEX	C	B	S	Memory type	Shareability	Other attributes
b000	0	0	x (1)	Strongly-ordered	Shareable	-
		1	x (1)	Device	Shareable	-
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner write-back. No write allocate.
			1		Shareable	
b001	0	0	0	Normal	Not shareable	
			1		Shareable	
		1	x (1)	Reserved encoding		-
	1	0	x (1)	Implementation defined attributes.		-
		1	0	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
			1		Shareable	
b010	0	0	x (1)	Device	Not shareable	Nonshared Device.
		1	x (1)	Reserved encoding		-
	1	x (1)	x (1)	Reserved encoding		-
b1BB	A	A	0	Normal	Not shareable	
			1		Shareable	

1. The MPU ignores the value of this bit.

Table 10-38 shows the cache policy for memory attribute encodings with a TEX value is in the range 4-7.

Table 10-38. 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 10-39 shows the AP encodings that define the access permissions for privileged and unprivileged software.

Table 10-39. 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

10.23.7 MPU mismatch

When an access violates the MPU permissions, the processor generates a memory management fault, see “Exceptions and interrupts” on page 61. The MMFSR indicates the cause of the fault. See “Memory Management Fault Status Register” on page 183 for more information.

10.23.8 Updating an MPU region

To update the attributes for an MPU region, update the RNR, RBAR and RASR registers. You can program each register separately, or use a multiple-word write to program all of these registers. You can use the RBAR and RASR aliases to program up to four regions simultaneously using an STM instruction.

10.23.8.1 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 you have previously enabled the region being changed. 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
```

Software must use memory barrier instructions:

- before MPU setup if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings
- after 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 mechanism cause memory barrier behavior.

Software does not need any memory barrier instructions during MPU setup, because it accesses the MPU through the PPB, which is a Strongly-Ordered memory region.

For example, if you want all of the memory access behavior to take effect immediately after the programming sequence, use a DSB instruction and an ISB instruction. A DSB is required after changing MPU settings, such as at the end of 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 you do not require an ISB.

10.23.8.2 Updating an MPU region using multi-word writes

You 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
```

You can do this in two words for pre-packed information. This means that the RBAR contains the required region number and had the VALID bit set to 1, see “MPU Region Base Address Register” on page 203. 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
```

10.23.8.3 Subregions

Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the RASR to disable a subregion, see “MPU Region Attribute and Size Register” on page 204. The least significant bit of SRD controls the first subregion, and the most significant bit controls the last subregion. Disabling

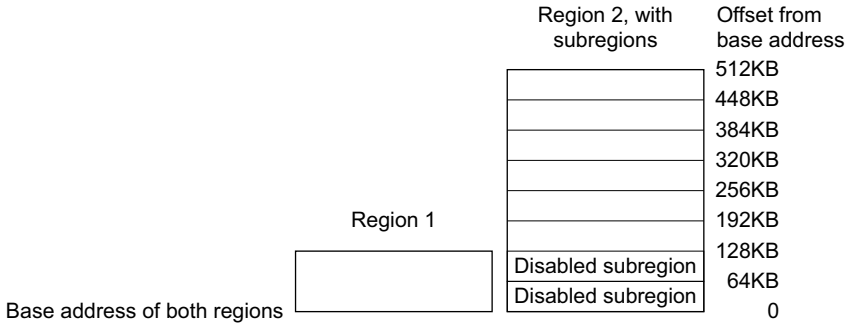
a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions, With regions of these sizes, you must set the SRD field to 0x00, otherwise the MPU behavior is Unpredictable.

10.23.8.4 Example of SRD use

Two regions with the same base address overlap. Region one is 128KB, and region two is 512KB. To ensure the attributes from region one apply to the first128KB region, set the SRD field for region two to b00000011 to disable the first two subregions, as Figure 10-9 shows

Figure 10-9. SRD use



10.23.9 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 software uses aligned accesses of the correct size to access MPU registers:

- except for the RASR, it must use aligned word accesses
- for the RASR 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.

10.23.9.1 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 10-40. 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 share ability 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 share ability attribute might be important. In these cases refer to the recommendations of the memory device manufacturer.

10.24 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.

See also "Index register"

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.

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-M3 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.

See also "Little-endian (LE)"

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 "Little-endian memory" , "Endianness"

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.

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.

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.

Should Be One (SBO)

Write as 1, or all 1s for bit fields, by software. Writing as 0 produces Unpredictable results.

Should Be Zero (SBZ)

Write as 0, or all 0s for bit fields, by software. Writing as 1 produces Unpredictable results.

Should Be Zero or Preserved (SBZP)

Write as 0, or all 0s for bit fields, by software, or preserved by writing the same value back that has been previously read from the same field on the same processor.

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 (UNP)

You 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 you are using the debugging features of a processor.

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.

11. Debug and Test Features

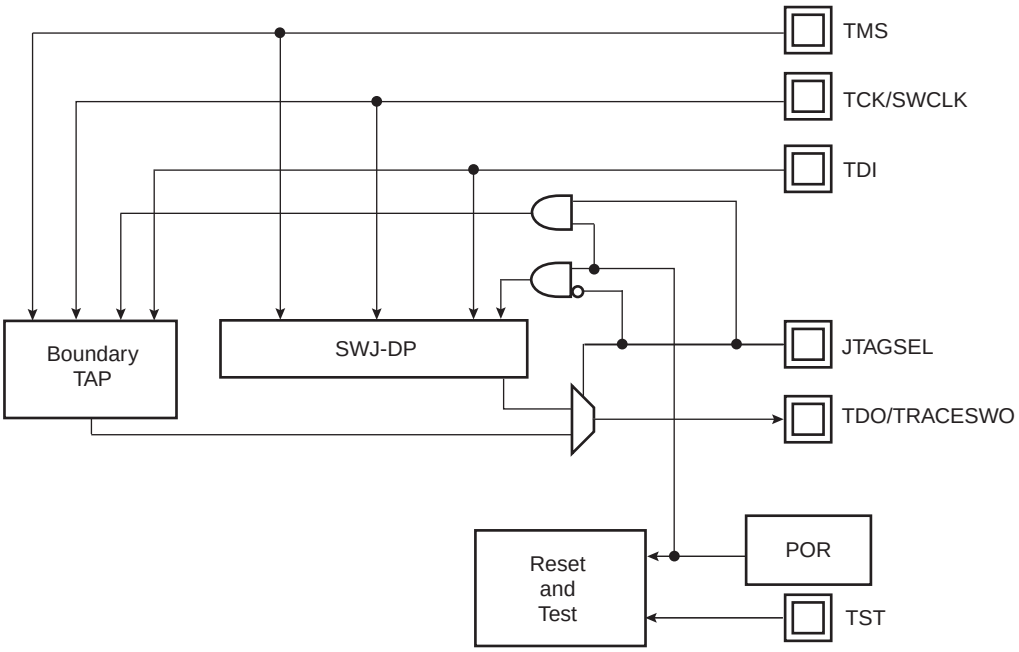
11.1 Description

The SAM3 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.

11.2 Embedded Characteristics

- Debug access to all memory and registers in the system, including Cortex-M3 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-scan on all digital pins

Figure 11-1. Debug and Test Block Diagram

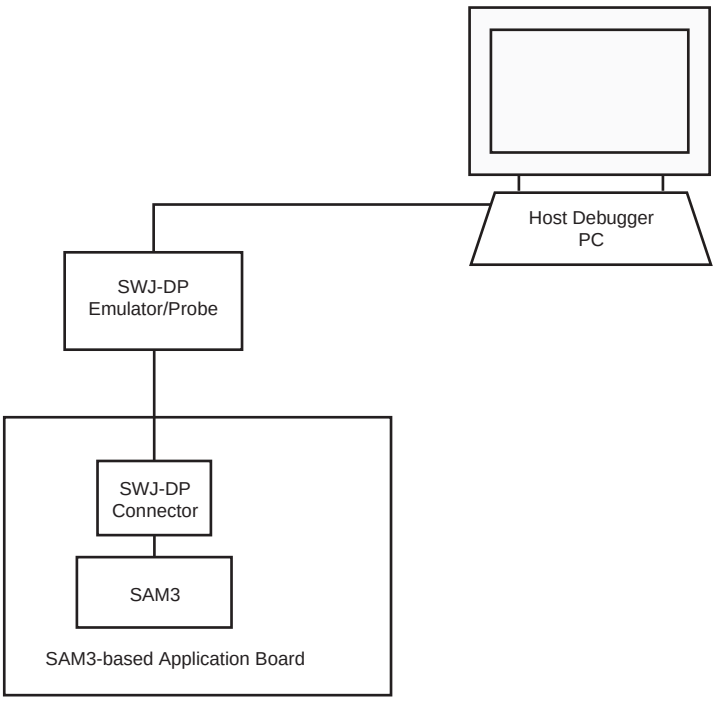


11.3 Application Examples

11.3.1 Debug Environment

Figure 11-2 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.

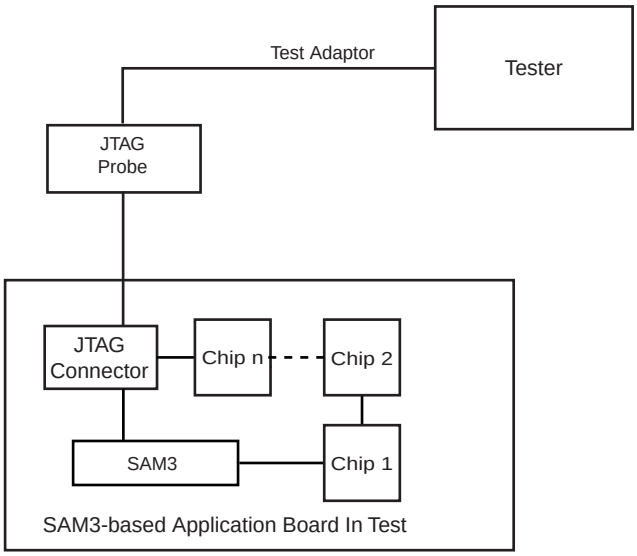
Figure 11-2. Application Debug Environment Example



11.3.2 Test Environment

Figure 11-3 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 11-3. Application Test Environment Example



11.4 Debug and Test Pin Description

Table 11-1. Debug and Test Signal List

Signal Name	Function	Type	Active Level
Reset/Test			
NRST	Microcontroller Reset	Input/Output	Low
TST	Test Select	Input	
SWD/JTAG			
TCK/SWCLK	Test Clock/Serial Wire Clock	Input	
TDI	Test Data In	Input	
TDO/TRACESWO	Test Data Out/Trace Asynchronous Data Out	Output (1)	
TMS/SWDIO	Test Mode Select/Serial Wire Input/Output	Input	
JTAGSEL	JTAG Selection	Input	High

Note: 1. TDO pin is set in input mode when the Cortex-M3 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.

11.5 Functional Description

11.5.1 Test Pin

One dedicated pin, TST, is used to define the device operating mode. When this pin is at low level during power-up, the device is in normal operating mode. When at high level, the device is in test mode or FFPI mode. The TST pin integrates a permanent pull-down resistor of about 15 kΩ,so that it can be left unconnected for normal operation. Note that when setting the TST pin to low or high level at power up, it must remain in the same state during the duration of the whole operation.

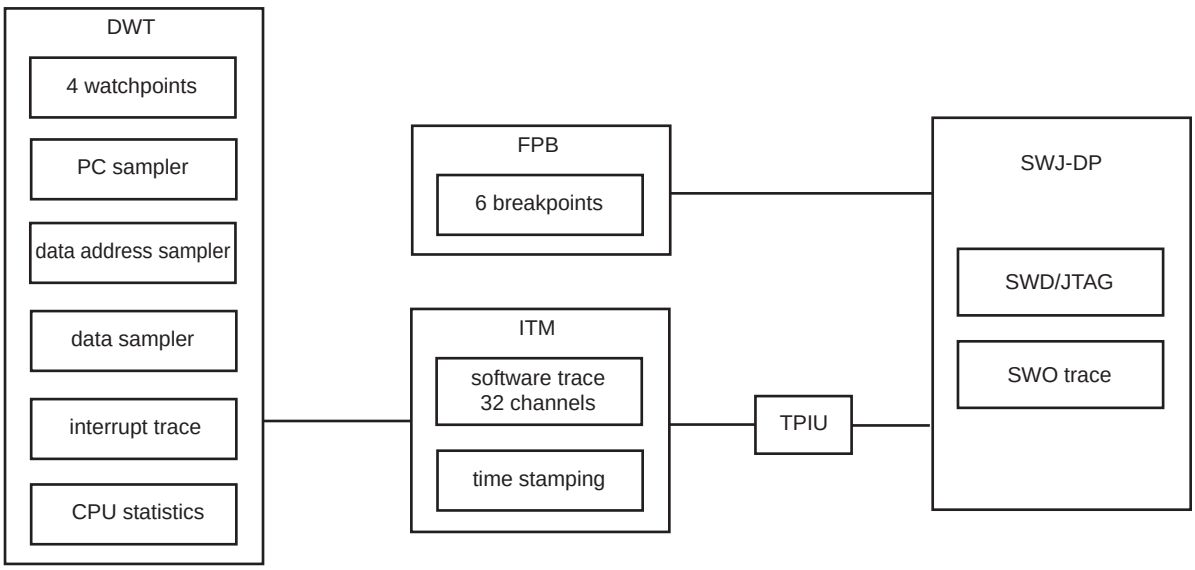
11.5.2 Debug Architecture

Figure 11-4 shows the SAM3 debug architecture. The Cortex-M3 embeds five 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 M3-based microcontrollers. For further details on SWJ-DP see the Cortex M3 technical reference manual.

Figure 11-4. Debug Architecture



11.5.3 Serial Wire/JTAG Debug Port (SWJ-DP)

The Cortex-M3 embeds a SWJ-DP Debug port which is the standard CoreSight™ debug port. It combines Serial Wire Debug Port (SW-DP), from 2 to 3 pins and JTAG debug Port (JTAG-DP), 5 pins.

By default, the JTAG Debug Port is active. If the host debugger wants to switch to the Serial Wire Debug Port, it must provide a dedicated JTAG sequence on TMS/SWDIO and TCK/SWCLK which disables JTAG-DP and enables 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.

Table 11-2. 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.

11.5.3.1 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

11.5.4 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.

11.5.5 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 following items:

- Clock cycle (CYCCNT)
- Folded instructions
- Load Store Unit (LSU) operations
- Sleep Cycles
- CPI (all instruction cycles except for the first cycle)
- Interrupt overhead

11.5.6 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 “printf” function. For more information, refer to Section 11.5.6.1 “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.

11.5.6.1 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 11.5.6.3 “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.

11.5.6.2 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

11.5.6.3 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).

11.5.7 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 FWUP, NRSTB and JTAGSEL are high while TST is tied low during power-up and must be kept in this state during the whole boundary scan operation. VDDCORE must be externally supplied between 1.8V and 1.95V. 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 www.atmel.com.

11.5.7.1 JTAG Boundary-scan Register

The Boundary-scan Register (BSR) contains a number of bits which correspond to active pins and associated control signals.

Each SAM3 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 SAM3 Series.

11.5.8 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
SAM3X	0x05B2B

- **MANUFACTURER IDENTITY[11:1]**

Set to 0x01F.

- **Bit[0] Required by IEEE Std. 1149.1.**

Set to 0x1.

Chip Name	JTAG ID Code
SAM3X	0x05B2B03F

12. Reset Controller (RSTC)

12.1 Description

The Reset Controller (RSTC), based on power-on reset cells, handles all the resets of the system without any external components. It reports which reset occurred last.

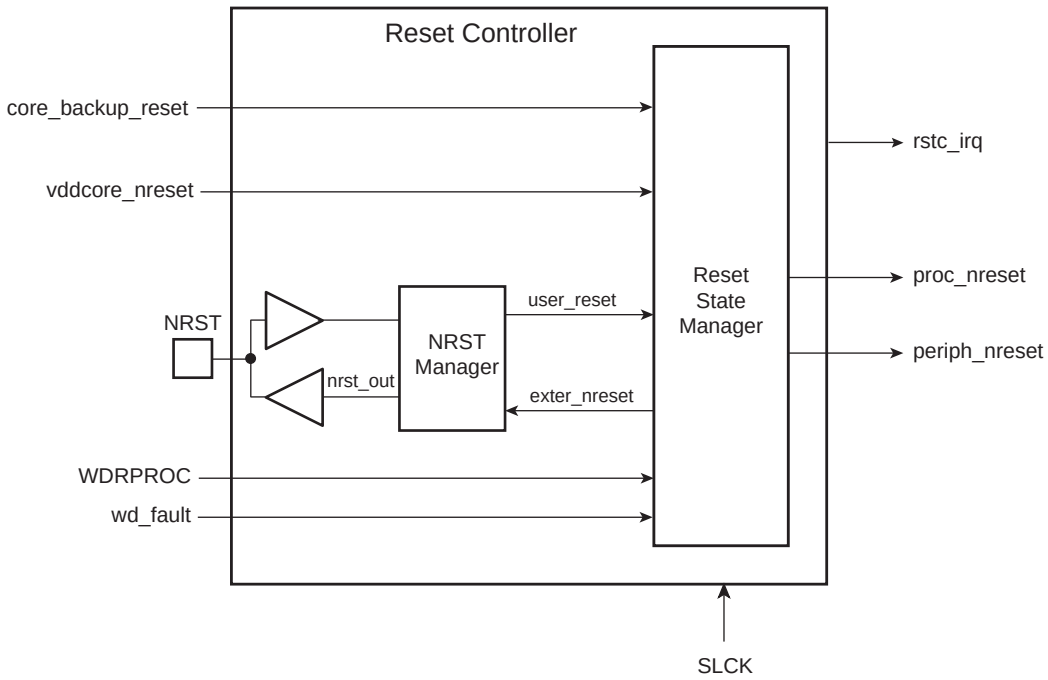
The Reset Controller also drives independently or simultaneously the external reset and the peripheral and processor resets.

12.2 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
- AMBA™-compliant Interface
  - Interface to the ARM® Advanced Peripheral Bus

12.3 Block Diagram

Figure 12-1. Reset Controller Block Diagram



12.4 Functional Description

12.4.1 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.

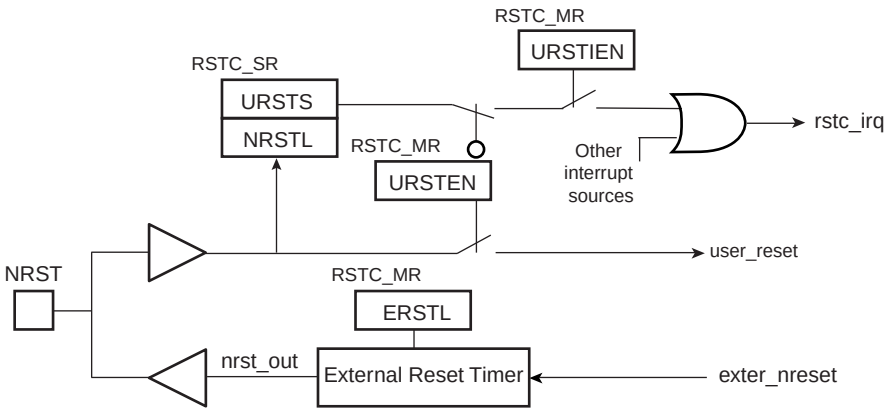
The Reset Controller Mode Register (RSTC_MR), allowing the configuration of the Reset Controller, is powered with VDDIO, so that its configuration is saved as long as VDDIO is on.

12.4.2 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 12-2 shows the block diagram of the NRST Manager.

Figure 12-2. NRST Manager



12.4.2.1 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.

12.4.2.2 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.

12.4.3 Brownout Manager

The Brownout manager is embedded within the Supply Controller, please refer to the product Supply Controller section for a detailed description.

12.4.4 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.

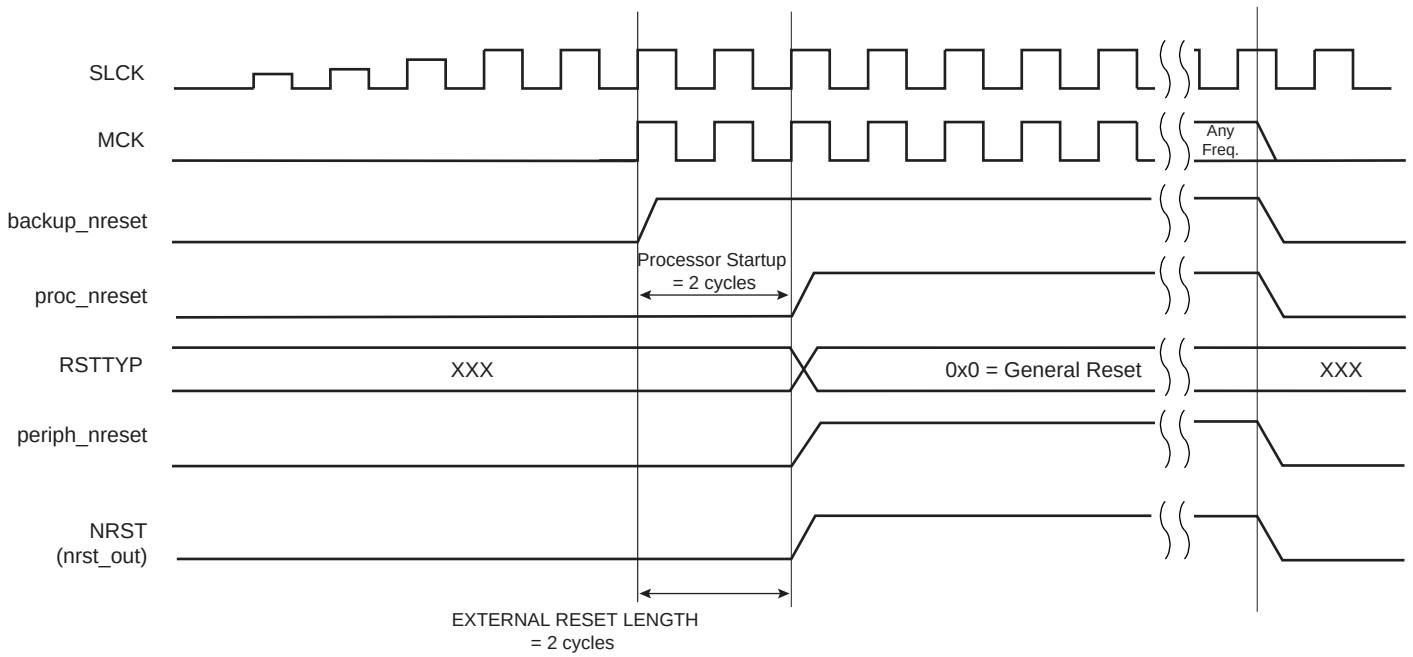
12.4.4.1 General Reset

A general reset occurs when a Power-on-reset is detected, an Asynchronous Master Reset (NRSTB pin) is requested, 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 12-3 shows how the General Reset affects the reset signals.

Figure 12-3. General Reset State



12.4.4.2 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.

12.4.4.3 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.

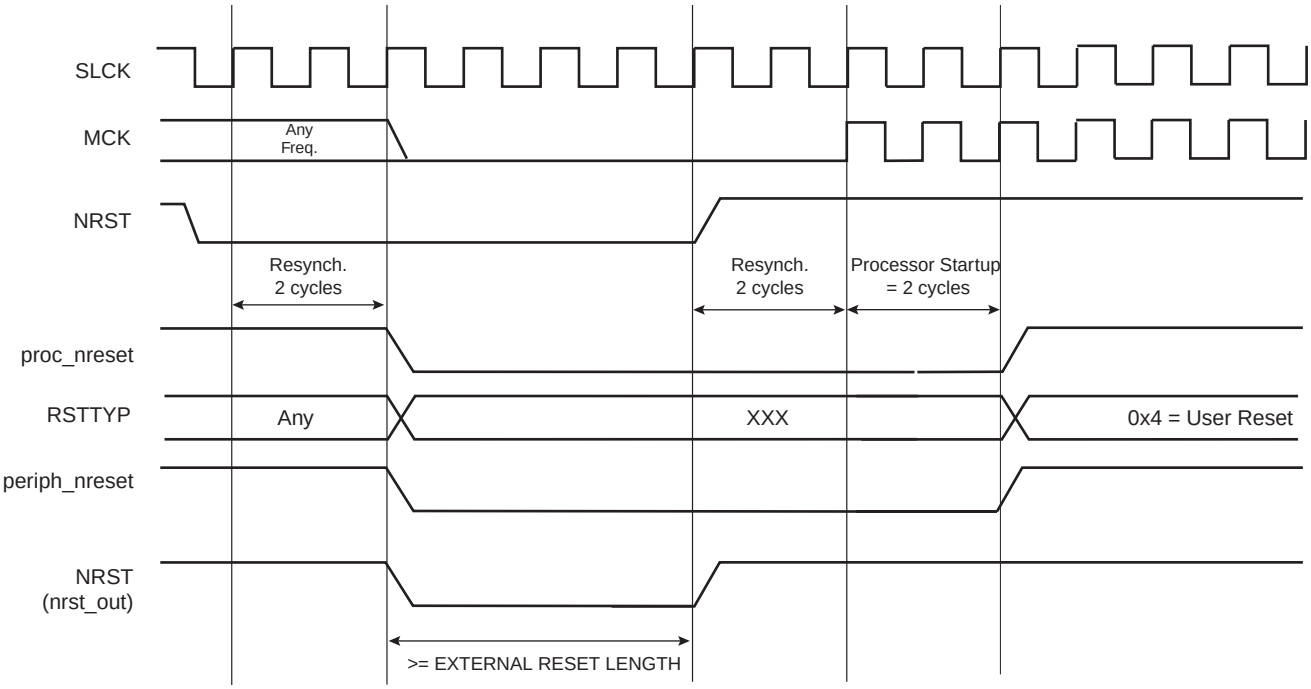
The User Reset is entered as soon as a low level is detected on NRST. The Processor Reset and the Peripheral Reset are asserted.

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 12-4. User Reset State



12.4.4.4 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.

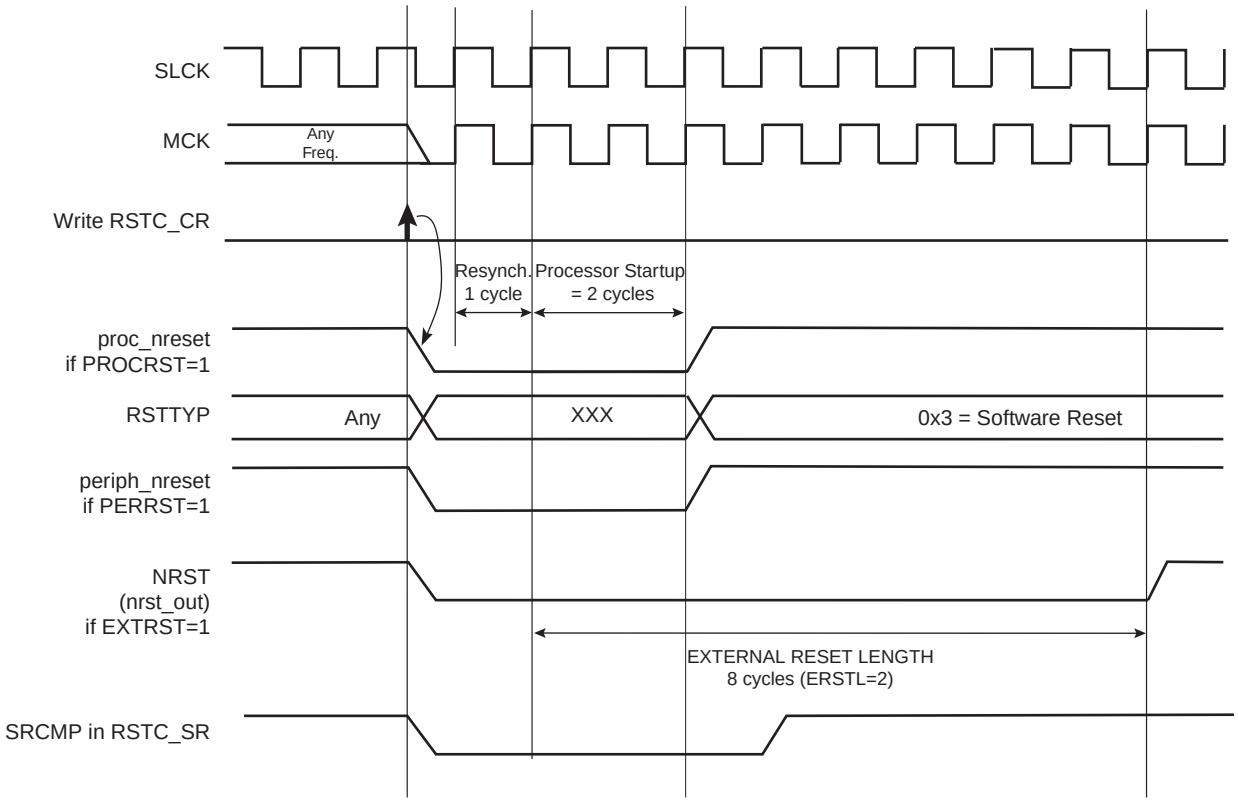
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset is left, i.e.; synchronously to SLCK.

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 12-5. Software Reset



12.4.4.5 Watchdog Reset

The Watchdog Reset is entered when a watchdog fault occurs. This state lasts 3 Slow Clock cycles.

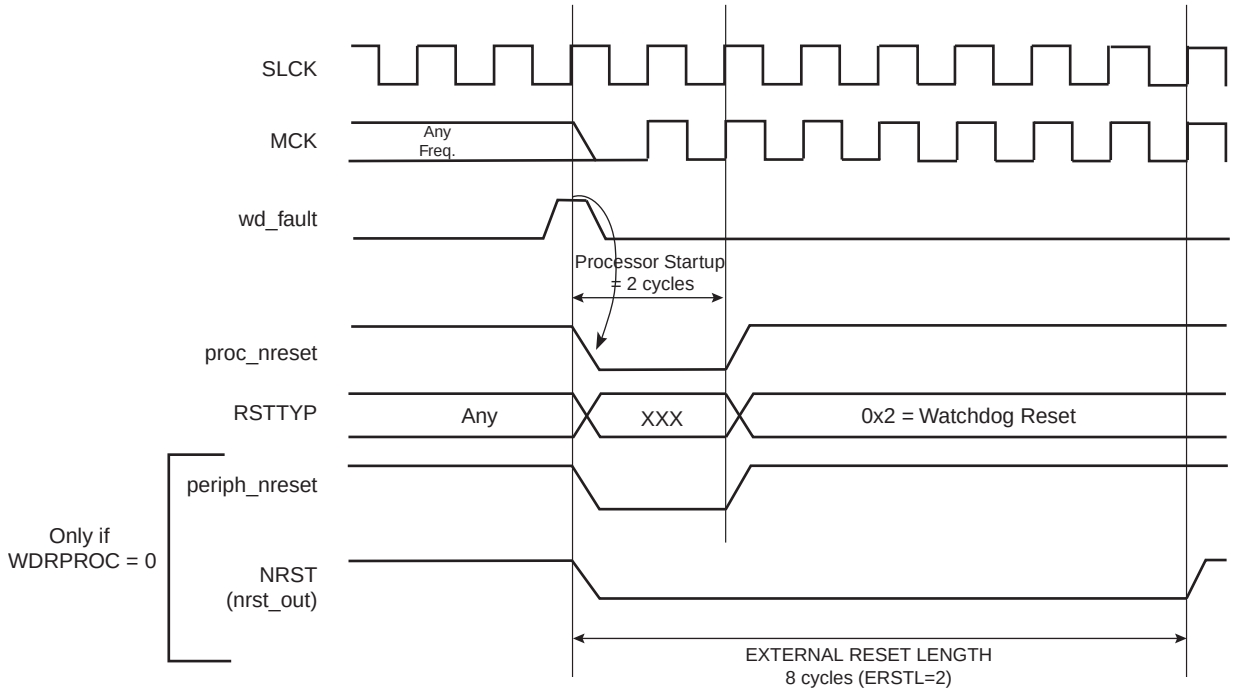
When in Watchdog Reset, assertion of the reset signals depends on the WDRPROC bit in WDT_MR:

- 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 12-6. Watchdog Reset



12.4.5 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:

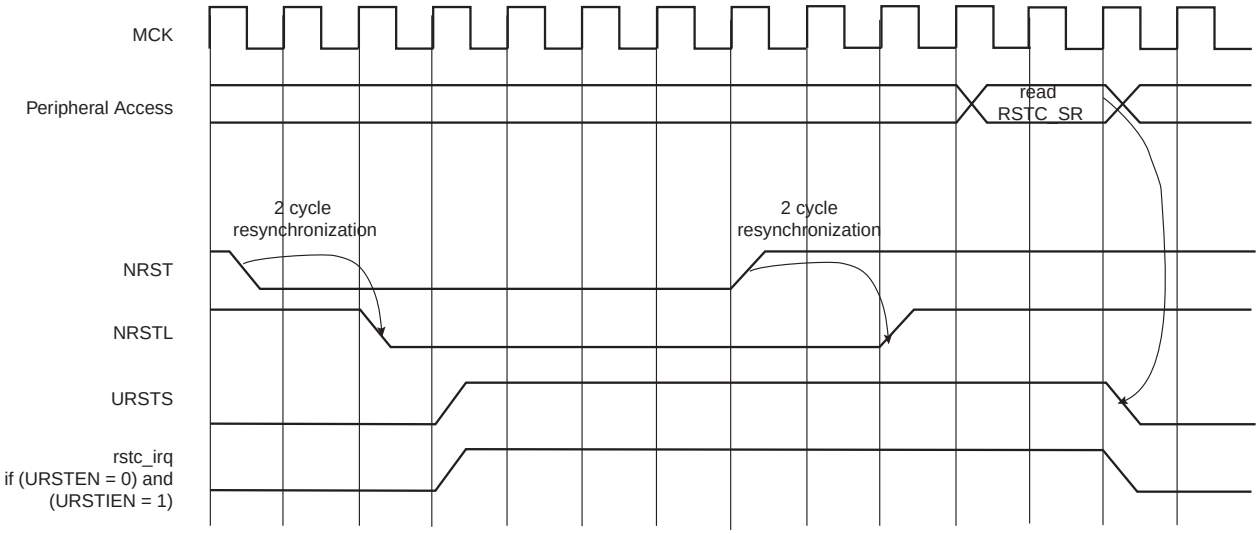
- When in User Reset:
  - A watchdog event is impossible because the Watchdog Timer is being reset by the proc_nreset signal.
  - A software reset is impossible, since the processor reset is being activated.
- When in Software Reset:
  - A watchdog event has priority over the current state.
  - The NRST 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.

12.4.6 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 12-7). 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 12-7. Reset Controller Status and Interrupt



12.5 Reset Controller (RSTC) User Interface

Table 12-1. 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

12.5.1 Reset Controller Control Register

Name: RSTC_CR

Address: 0x400E1A00

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.

- **KEY: Password**

Should be written at value 0xA5. Writing any other value in this field aborts the write operation.

12.5.2 Reset Controller Status Register

Name: RSTC_SR

Address: 0x400E1A04

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**

Reports the cause of the last processor reset. Reading this RSTC_SR does not reset this field.

RSTTYP			Reset Type	Comments
0	0	0	General Reset	First power-up Reset
0	0	1	Backup Reset	Return from Backup mode
0	1	0	Watchdog Reset	Watchdog fault occurred
0	1	1	Software Reset	Processor reset required by the software
1	0	0	User Reset	NRST pin detected low

- **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.

12.5.3 Reset Controller Mode Register

Name: RSTC_MR

Address: 0x400E1A08

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: Password**

Should be written at value 0xA5. Writing any other value in this field aborts the write operation.

13. Real-time Timer (RTT)

13.1 Description

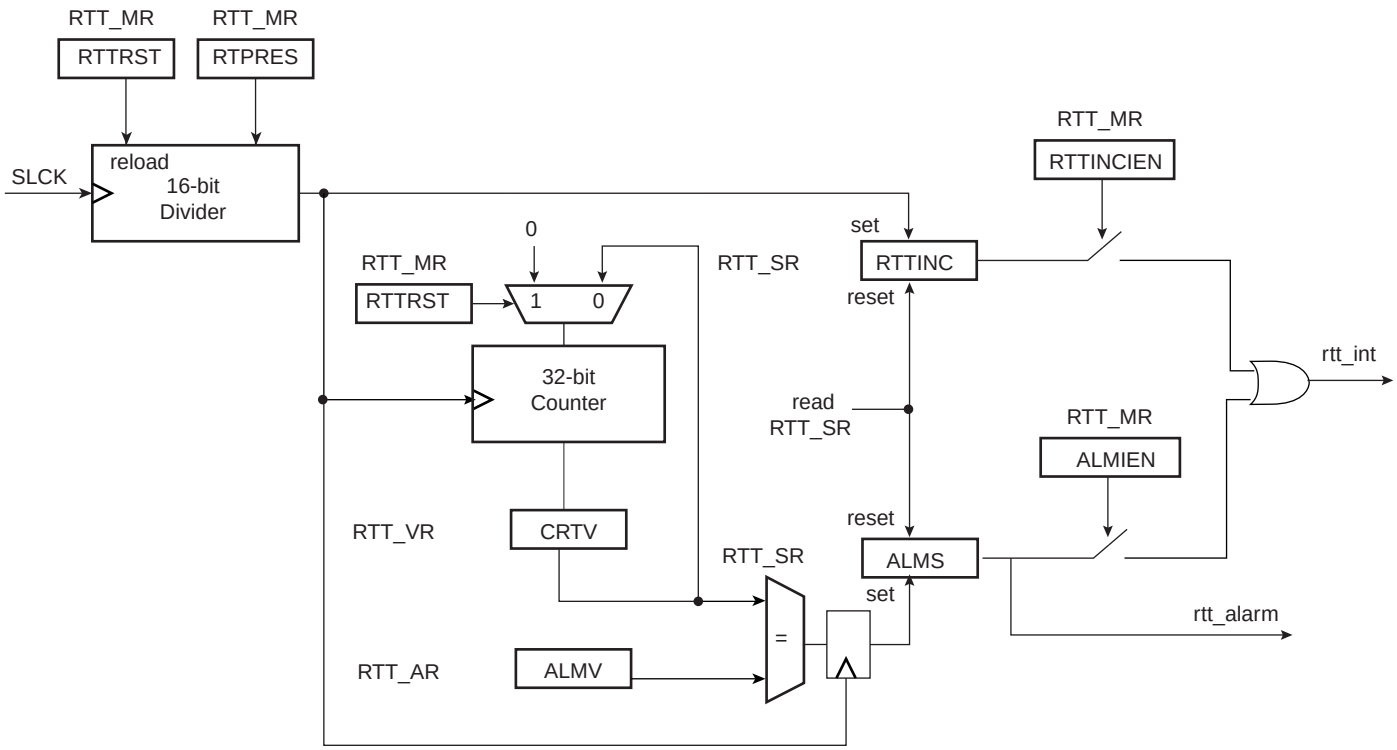
The Real-time Timer is built around a 32-bit counter used to count roll-over events of the programmable 16-bit prescaler which enables counting elapsed seconds from a 32 kHz slow clock source. It generates a periodic interrupt and/or triggers an alarm on a programmed value.

13.2 Embedded Characteristics

- 32-bit Free-running Counter on prescaled slow clock
- 16-bit Configurable Prescaler
- Interrupt on Alarm

13.3 Block Diagram

Figure 13-1. Real-time Timer



13.4 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 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.

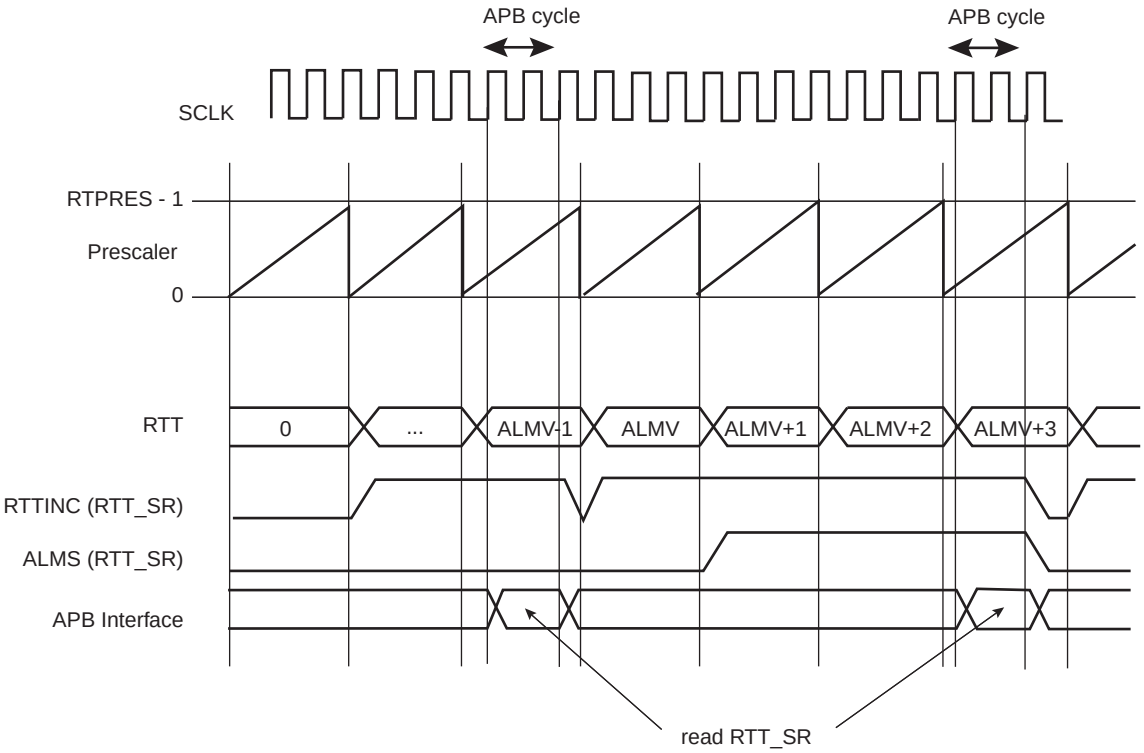
The bit RTTINC in RTT_SR is set each time the Real-time Timer counter is incremented. This bit can be used to start a periodic interrupt, the period being one second when the RTPRES is programmed with 0x8000 and Slow Clock equal to 32.768 Hz.

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.

Note: Because of the asynchronism between the Slow Clock (SCLK) and the System Clock (MCK):
1) The restart of the counter and the reset of the RTT_VR current value register is effective only 2 slow clock cycles after the write of the RTTRST bit in the RTT_MR register.
2) The status register flags reset is taken into account only 2 slow clock cycles after the read of the RTT_SR (Status Register).

Figure 13-2. RTT Counting



13.5 Real-time Timer (RTT) User Interface

Table 13-1. 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

13.5.1 Real-time Timer Mode Register

Name: RTT_MR

Address: 0x400E1A30

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	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 SLCK periods required to increment the Real-time timer. RTPRES is defined as follows:

RTPRES = 0: The prescaler period is equal to 2^{16} * SCLK period.

RTPRES ≠ 0: The prescaler period is equal to RTPRES * SCLK period.

- **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.

13.5.2 Real-time Timer Alarm Register

Name: RTT_AR

Address: 0x400E1A34

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.

13.5.3 Real-time Timer Value Register

Name: RTT_VR

Address: 0x400E1A38

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.

13.5.4 Real-time Timer Status Register

Name: RTT_SR

Address: 0x400E1A3C

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.

14. Real-time Clock (RTC)

14.1 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 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.

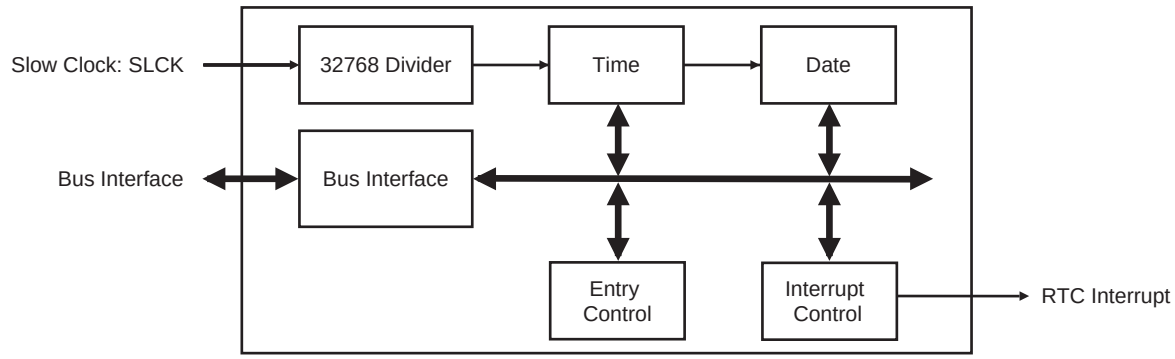
Updating time and calendar fields and configuring the alarm fields are performed by a parallel capture on the 32-bit data bus. An entry control is performed to avoid loading registers with incompatible BCD format data or with an incompatible date according to the current month/year/century.

14.2 Embedded Characteristics

- Low Power Consumption
- Full Asynchronous Design
- Two Hundred Year Gregorian Calendar
- Programmable Periodic Interrupt
- Time, Date and Alarm 32-bit Parallel Load
- Write Protected Registers

14.3 Block Diagram

Figure 14-1. RTC Block Diagram



14.4 Product Dependencies

14.4.1 Power Management

The Real-time Clock is continuously clocked at 32768 Hz. The Power Management Controller has no effect on RTC behavior.

14.4.2 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.

14.5 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.

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.

14.5.1 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.

14.5.2 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.

14.5.3 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.

14.5.4 Error Checking

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)
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_MODE register, a 12-hour value can be programmed and the returned value on RTC_TIME will be the corresponding 24-hour value. The entry control checks the value of the AM/PM indicator (bit 22 of RTC_TIME register) to determine the range to be checked.

14.5.5 Updating Time/Calendar

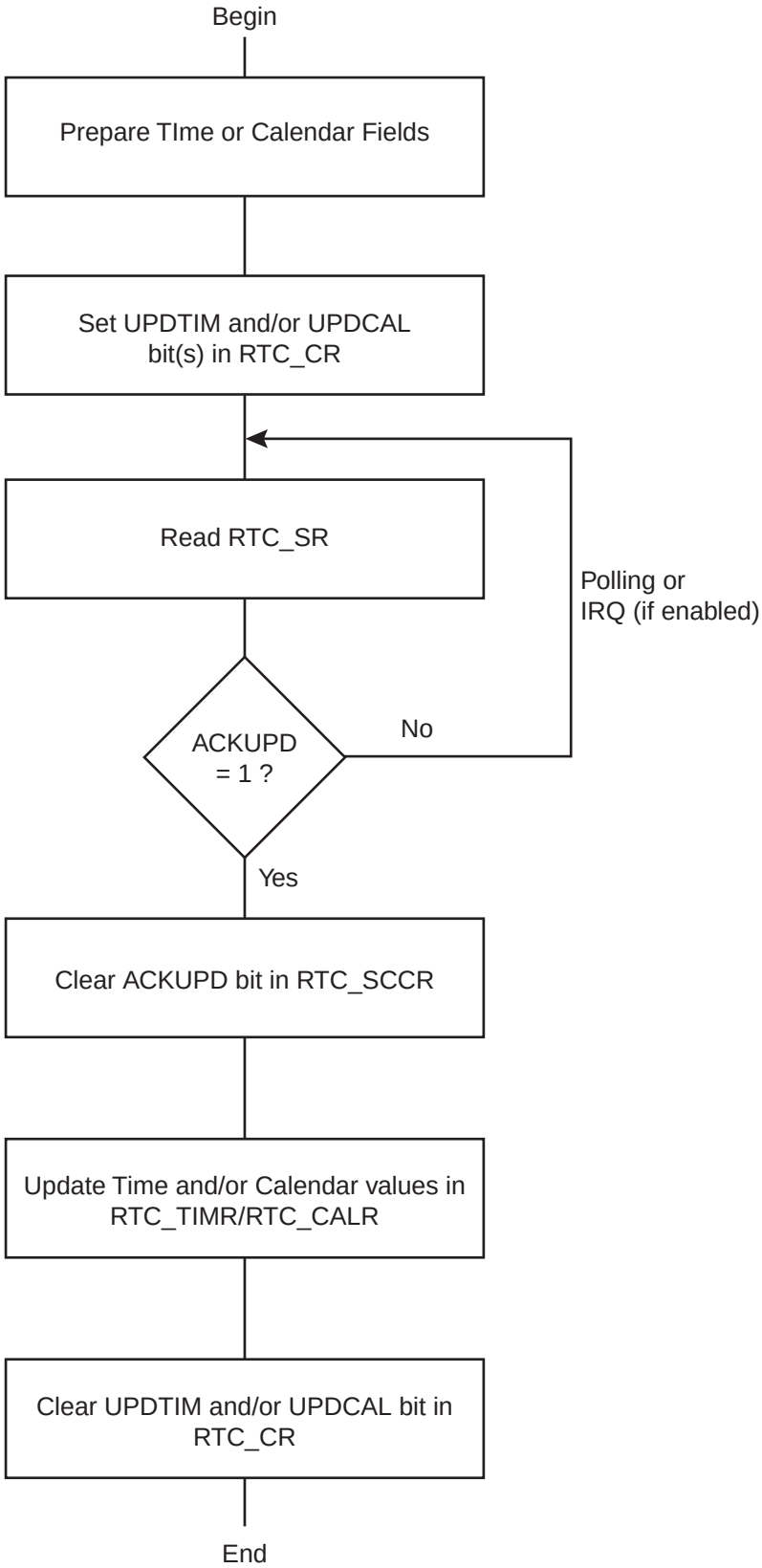
To update any of the time/calendar fields, the user must first stop the RTC by setting the corresponding field in the Control Register. Bit UPDTIM must be set to update time fields (hour, minute, second) and bit UPDCAL must be set to update calendar fields (century, year, month, date, day).

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 circuity (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 14-2. Update Sequence



14.6 Real-time Clock (RTC) User Interface

Table 14-1. 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	0x01210720
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
0x30–0xE0	Reserved Register	–	–	–
0xE4	Write Protect Mode Register	RTC_WPMR	Read-write	0x00000000
0xE8–0xF8	Reserved Register	–	–	–
0xFC	Reserved Register	–	–	–

Note: if an offset is not listed in the table it must be considered as reserved.

14.6.1 RTC Control Register

Name: RTC_CR

Address: 0x400E1A60

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

This register can only be written if the WPEN bit is cleared in “RTC Write Protect Mode Register” on page 259.

- **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)
3	–	

14.6.2 RTC Mode Register

Name: RTC_MR

Address: 0x400E1A64

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
–	–	–	–	–	–	–	HRMOD

- **HRMOD: 12-/24-hour Mode**

0: 24-hour mode is selected.

1: 12-hour mode is selected.

All non-significant bits read zero.

14.6.3 RTC Time Register

Name: RTC_TIMR

Address: 0x400E1A68

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.

14.6.4 RTC Calendar Register

Name: RTC_CALR

Address: 0x400E1A6C

Access: Read-write

31	30	29	28	27	26	25	24
–	–	DATE					

23	22	21	20	19	18	17	16
DAY			MONTH				

15	14	13	12	11	10	9	8
YEAR							

7	6	5	4	3	2	1	0
–	CENT						

- **CENT: Current Century**

The range that can be set is 19 - 20 (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.

14.6.5 RTC Time Alarm Register

Name: RTC_TIMALR

Address: 0x400E1A70

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						

This register can only be written if the WPEN bit is cleared in “RTC Write Protect Mode Register” on page 259.

- **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.

14.6.6 RTC Calendar Alarm Register

Name: RTC_CALALR

Address: 0x400E1A74

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
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in “RTC Write Protect Mode Register” on page 259.

- **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.

14.6.7 RTC Status Register

Name: RTC_SR

Address: 0x400E1A78

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
–	–	–	CALEV	TIMEV	SEC	ALARM	ACKUPD

- **ACKUPD: Acknowledge for Update**

0: Time and calendar registers cannot be updated.

1: Time and calendar registers can be updated.

- **ALARM: Alarm Flag**

0: No alarm matching condition occurred.

1: An alarm matching condition has occurred.

- **SEC: Second Event**

0: No second event has occurred since the last clear.

1: At least one second event has occurred since the last clear.

- **TIMEV: Time Event**

0: No time event has occurred since the last clear.

1: 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 calendar event has occurred since the last clear.

1: 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.

14.6.8 RTC Status Clear Command Register

Name: RTC_SCCR

Address: 0x400E1A7C

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
–	–	–	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).

14.6.9 RTC Interrupt Enable Register

Name: RTC_IER

Address: 0x400E1A80

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
–	–	–	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.

14.6.10 RTC Interrupt Disable Register

Name: RTC_IDR

Address: 0x400E1A84

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
–	–	–	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.

14.6.11 RTC Interrupt Mask Register

Name: RTC_IMR

Address: 0x400E1A88

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.

14.6.12 RTC Valid Entry Register

Name: RTC_VER

Address: 0x400E1A8C

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.

14.6.13 RTC Write Protect Mode Register

Name: RTC_WPMR

Address: 0x400E1B44

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 0x525443 (“RTC” in ASCII).

1: Enables the Write Protect if WPKEY corresponds to 0x525443 (“RTC” in ASCII).

Protects the registers:

“RTC Mode Register” on page 248

“RTC Time Alarm Register” on page 251

“RTC Calendar Alarm Register” on page 252

15. Watchdog Timer (WDT)

15.1 Description

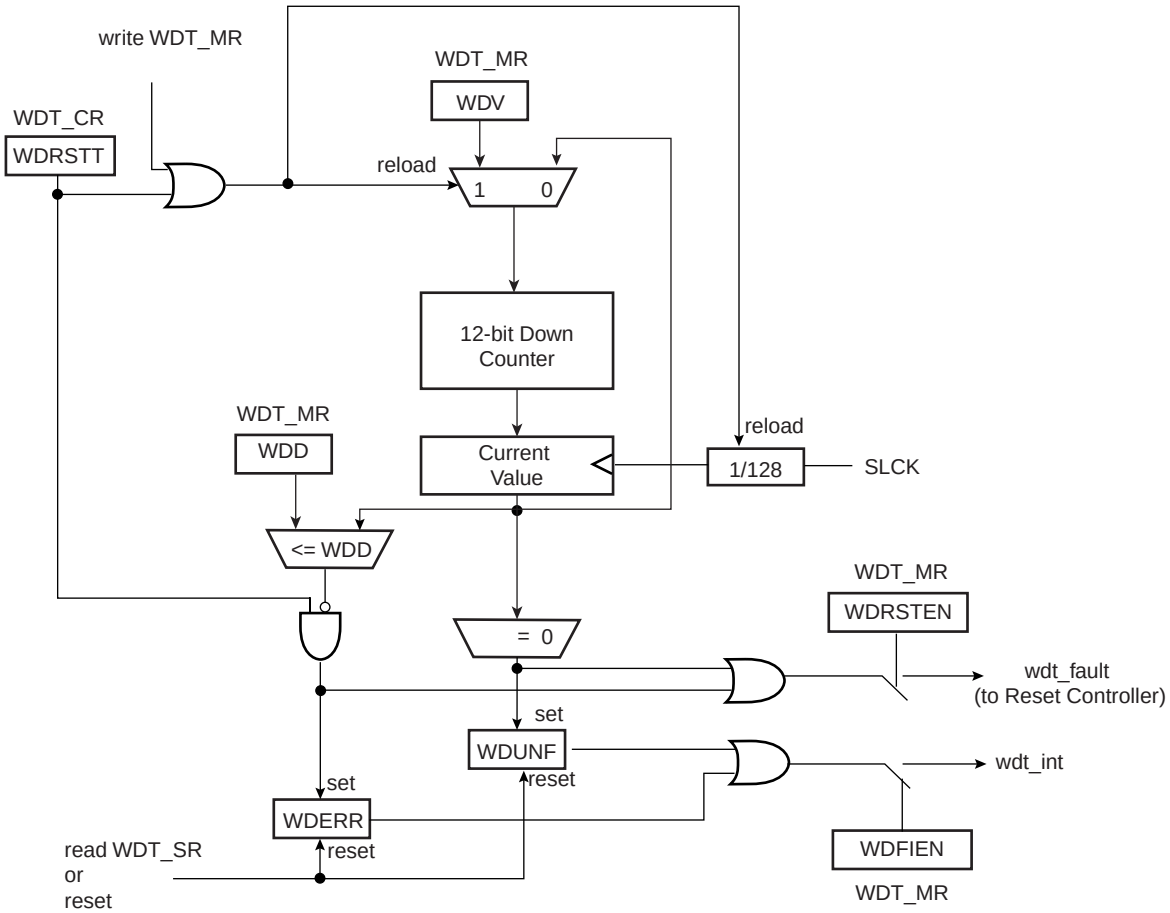
The Watchdog Timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow clock at 32.768 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in debug mode or idle mode.

15.2 Embedded Characteristics

- 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
- AMBA™-compliant Interface
  - Interfaces to the ARM® Advanced Peripheral Bus

15.3 Block Diagram

Figure 15-1. Watchdog Timer Block Diagram



15.4 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 0xFFF, 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.

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).

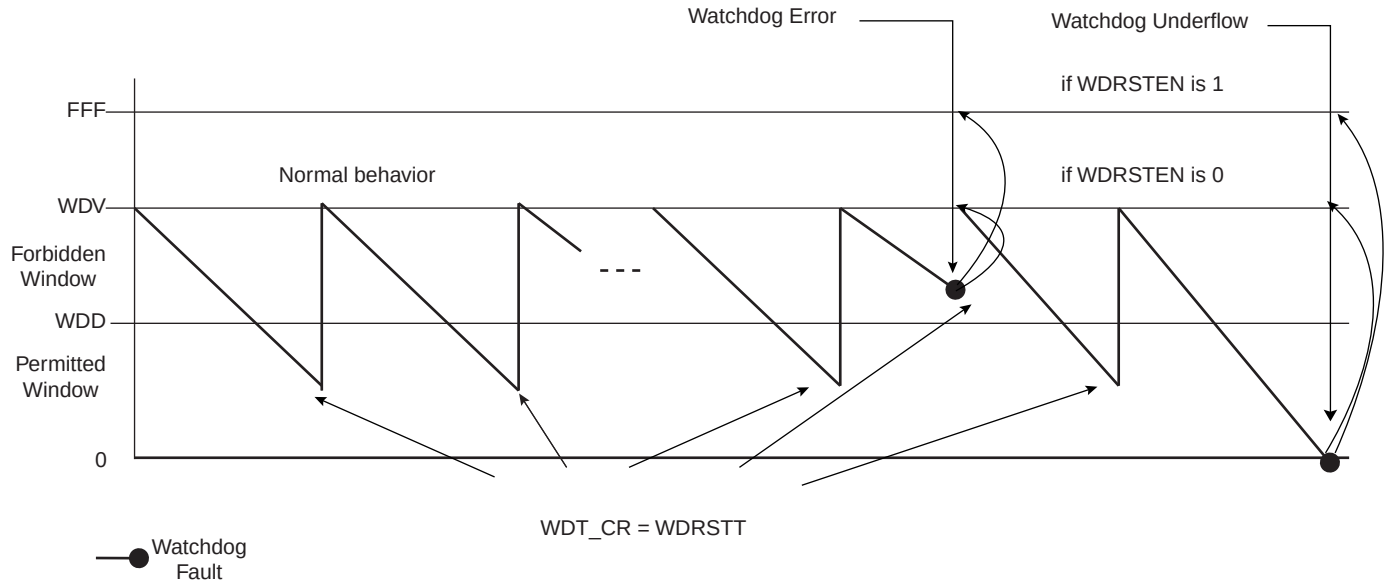
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDFIEN is set in the mode register. The signal “wdt_fault” to the reset controller causes a Watchdog reset if the WDRSTEN bit is set as already explained in the reset controller programmer datasheet. 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 15-2. Watchdog Behavior



15.5 Watchdog Timer (WDT) User Interface

Table 15-1. 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

15.5.1 Watchdog Timer Control Register

Name: WDT_CR

Address: 0x400E1A50

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 0xA5. Writing any other value in this field aborts the write operation.

15.5.2 Watchdog Timer Mode Register

Name: WDT_MR

Address: 0x400E1A54

Access: Read-write Once

31	30	29	28	27	26	25	24
		WDIDLEHLT	WDDBGHLT	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 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.

- **WDDBGHLT: 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.

15.5.3 Watchdog Timer Status Register

Name: WDT_SR

Address: 0x400E1A58

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.

16. Supply Controller (SUPC)

16.1 Description

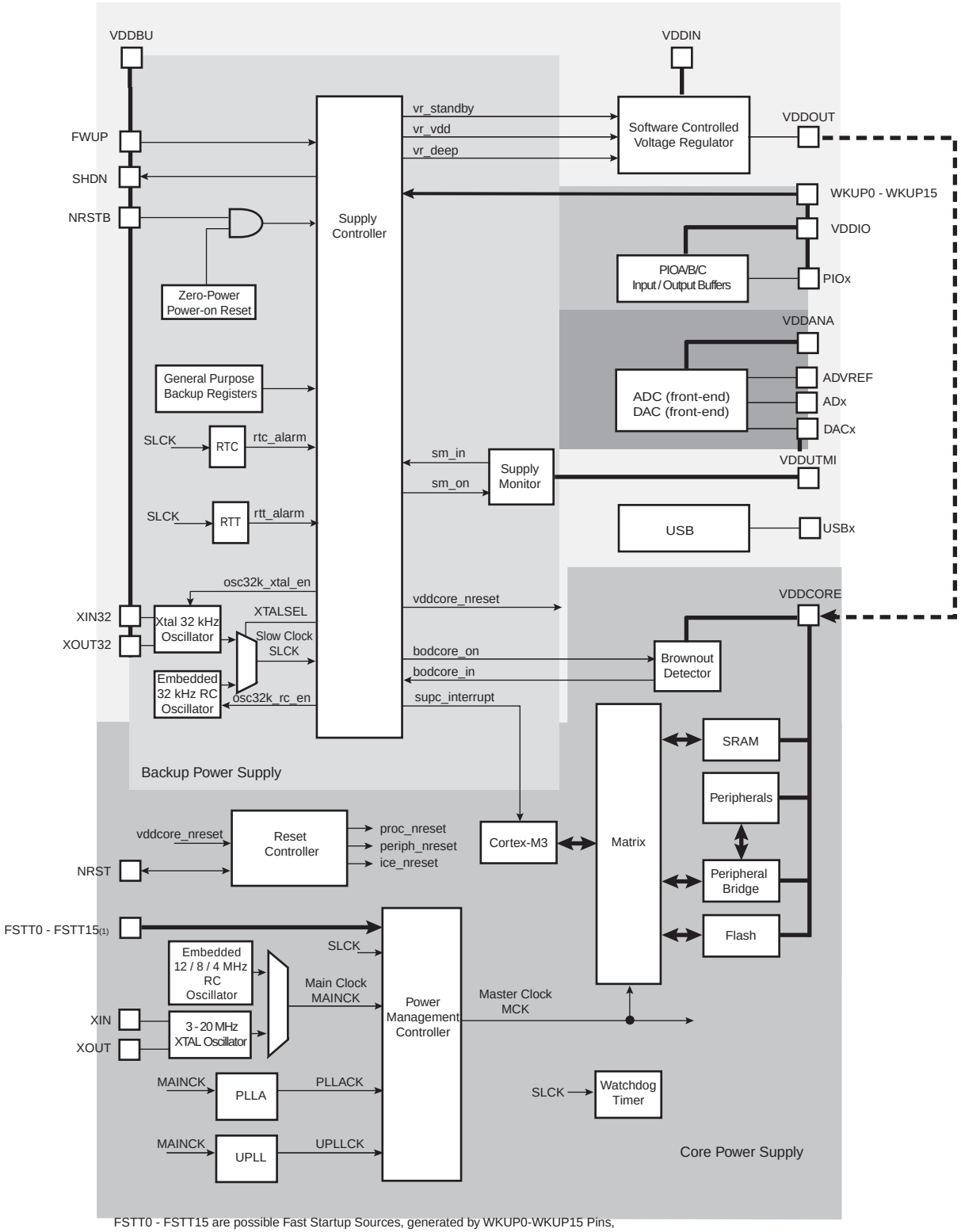
The Supply Controller (SUPC) controls the supply voltage of the Core of the system and manages the Backup Low Power Mode. In this mode, the current consumption is reduced to a few microamps for Backup power retention. Exit from this mode is possible on multiple wake-up sources including events on FWUP or WKUP pins, or a Clock alarm. The SUPC also generates the Slow Clock by selecting either the Low Power RC oscillator or the Low Power Crystal oscillator.

16.2 Embedded Characteristics

- Manages the Core Power Supply VDDCORE and the Backup Low Power Mode by Controlling the Embedded Voltage Regulator
- 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, with Programmable Debouncing
  - Real Time Clock Alarm
  - Real Time Timer Alarm
  - Supply Monitor Detection on VDDUTMI, with Programmable Scan Period and Voltage Threshold
- A Supply Monitor Detection on VDDUTMI or a Brownout Detection on VDDCORE can Trigger a Core Reset
- Embeds:
  - One 22 to 42 kHz Low Power RC Oscillator
  - One 32 kHz Low Power Crystal Oscillator
  - One Zero-Power Power-On Reset Cell
  - One Software Programmable Supply Monitor, on VDDUTMI Located in Backup Section
  - One Brownout Detector on VDDCORE Located in the Core

16.3 Block Diagram

Figure 16-1. Supply Controller Block Diagram



FSTT0 - FSTT15 are possible Fast Startup Sources, generated by WKUP0-WKUP15 Pins, but are not physical pins.

16.4 Supply Controller Functional Description

16.4.1 Supply Controller Overview

The device can be divided into two power supply areas:

- The Backup VDDBU 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 VDDUTMI 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 VDDUTMI power supply have a reset circuitry based on the NRSTB pin and a zero-power power-on reset cell. The zero-power power-on reset allows the SUPC to start properly as soon as the VDDUTMI voltage becomes valid. The NRSTB pin allows to reset the system from outside.

At startup of the system, once the backup voltage VDDUTMI is valid and the reset pin NRSTB is not driven low 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 VDDUTMI that is too low, the SUPC can assert the reset signal of the core “vddcore_nreset” signal until VDDUTMI 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 VDDUTMI 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 startup.

16.4.2 Slow Clock Generator

The Supply Controller embeds a slow clock generator that is supplied with the VDDBU power supply. As soon as the VDDBU 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 enables the crystal oscillator, then waits for 32,768 slow clock cycles, then switches the slow clock on the output of 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) allows knowing when the switch sequence is done.

Coming back on the RC oscillator is only possible by shutting down the VDDBU 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.

16.4.3 Voltage Regulator Control/Backup Low Power Mode

The Supply Controller can be used to control the embedded 1.8V 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 can be done also by using WFE (Wait for Event) Cortex-M3 instruction with the deep mode bit set to 1.

The Backup mode can also be entered by executing the WFI (Wait for Interrupt) or WFE (Wait for Event) Cortex-M3 instructions. To select the Backup mode entry mechanism, two options are available, depending on the SLEEPONEXIT bit in the Cortex-M3 System Control register:

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

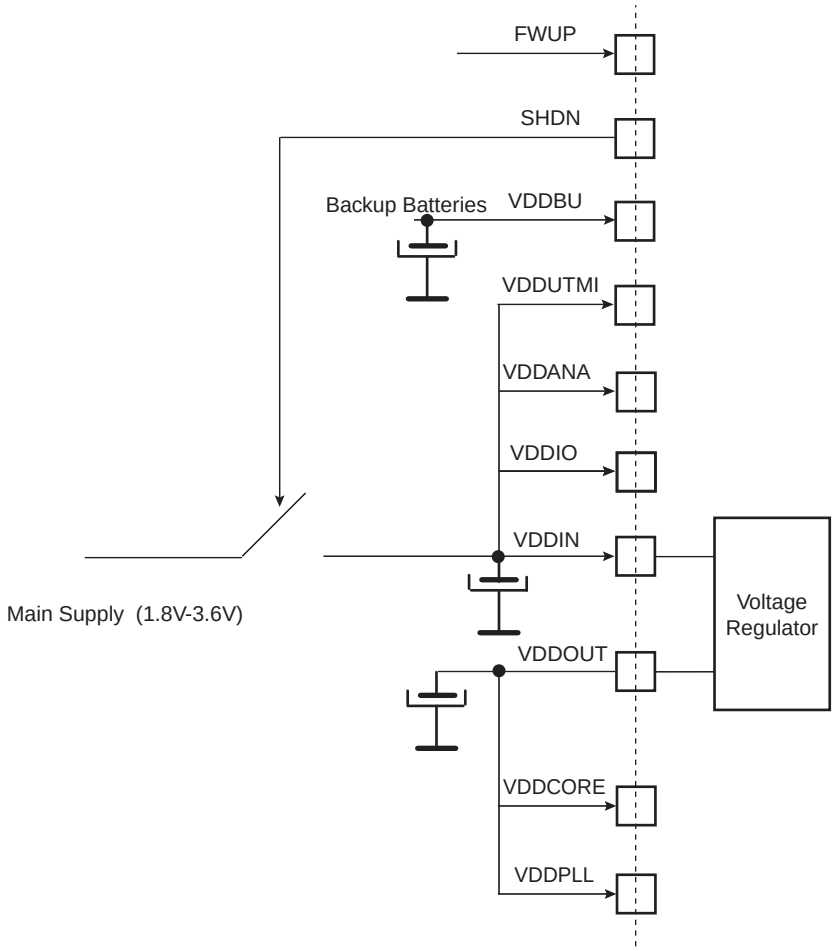
This asserts the vddcore_nreset signal after the write resynchronization time which lasts, in the worse case, two slow clock cycles. Once the vddcore_nreset signal is asserted, the processor and the peripherals are stopped one slow clock cycle before the core power supply shuts off.

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. Note that it is different from the Backup mode. Depending on the application, disabling the voltage regulator can reduce power consumption as the voltage regulator input (VDDIN) is shared with the ADC and DAC. This is done through ONREG bit in SUPC_MR.

16.4.4 Using Backup Batteries/Backup Supply

The product can be used with or without backup batteries, or more generally a backup supply. When a backup supply is used (See Figure 16-2), only VDDBU voltage is present in Backup mode and no other external supply is applied on the chip. In this case the user needs to clear VDDIORDY bit in the Supply Controller Mode Register (SUPC_MR) at least two slow clock periods before VDDIO voltage is removed. When waking up from Backup mode, the programmer needs to set VDDIORDY.

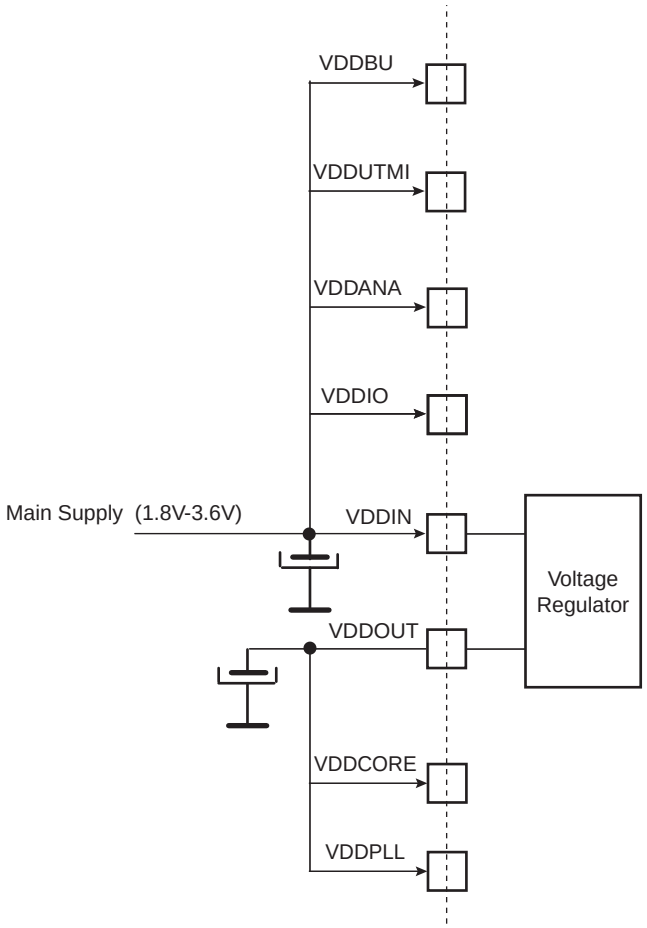
Figure 16-2. Separated Backup Supply Powering Scheme



Note: Restrictions: With Main Supply < 3V, some peripherals such as USB and ADC might not be operational. Refer to the DC Characteristics of the product for actual possible ranges for such peripherals.

When a separated backup supply for VDDBU is not used (See Figure 16-3), since the external voltage applied on VDDIO is kept, all of the I/O configurations (i.e. WKUP pin configuration) are kept during backup mode. When not using backup batteries, VDDIORDY is set so the user does not need to program it.

Figure 16-3. No Separated Backup Supply Powering Scheme



Note: Restrictions: With Main Supply < 3V, some peripherals such as USB and ADC might not be operational. Refer to the DC Characteristics of the product for actual possible ranges for such peripherals.

16.4.5 Supply Monitor

The Supply Controller embeds a supply monitor which is located in the VDDBU Backup Power Supply and which monitors VDDUTMI 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 VDDUTMI 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.

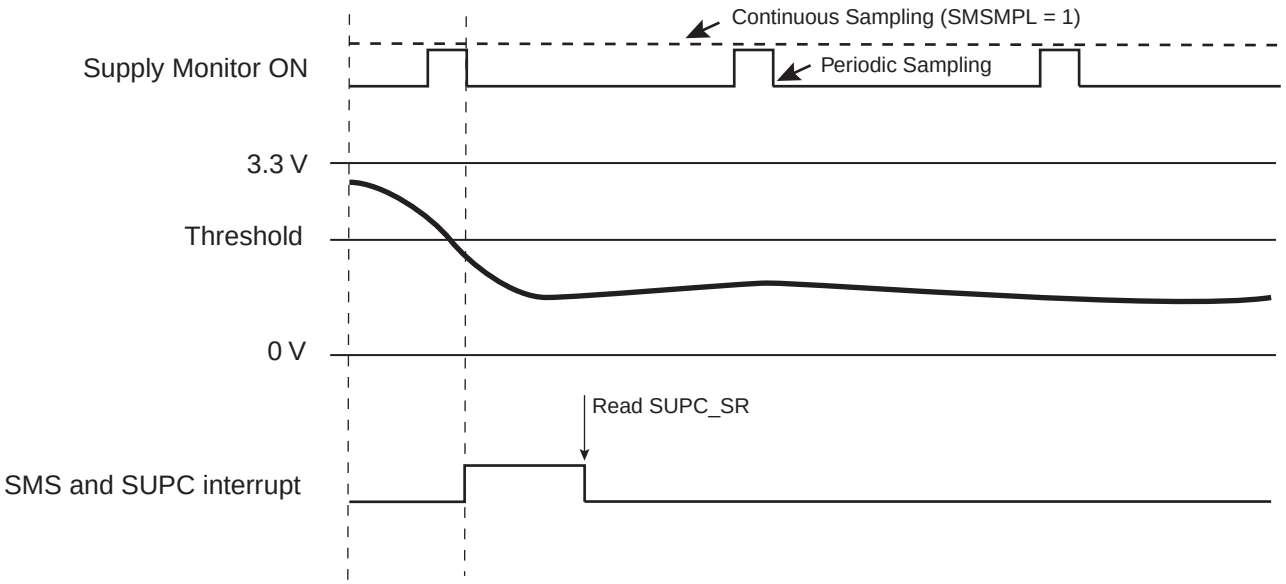
Waking up the core power supply when a supply monitor detection occurs can be enabled by programming the SMEN bit to 1 in the Supply Controller Wake Up Mode Register (SUPC_WUMR).

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 16-4. Supply Monitor Status Bit and Associated Interrupt



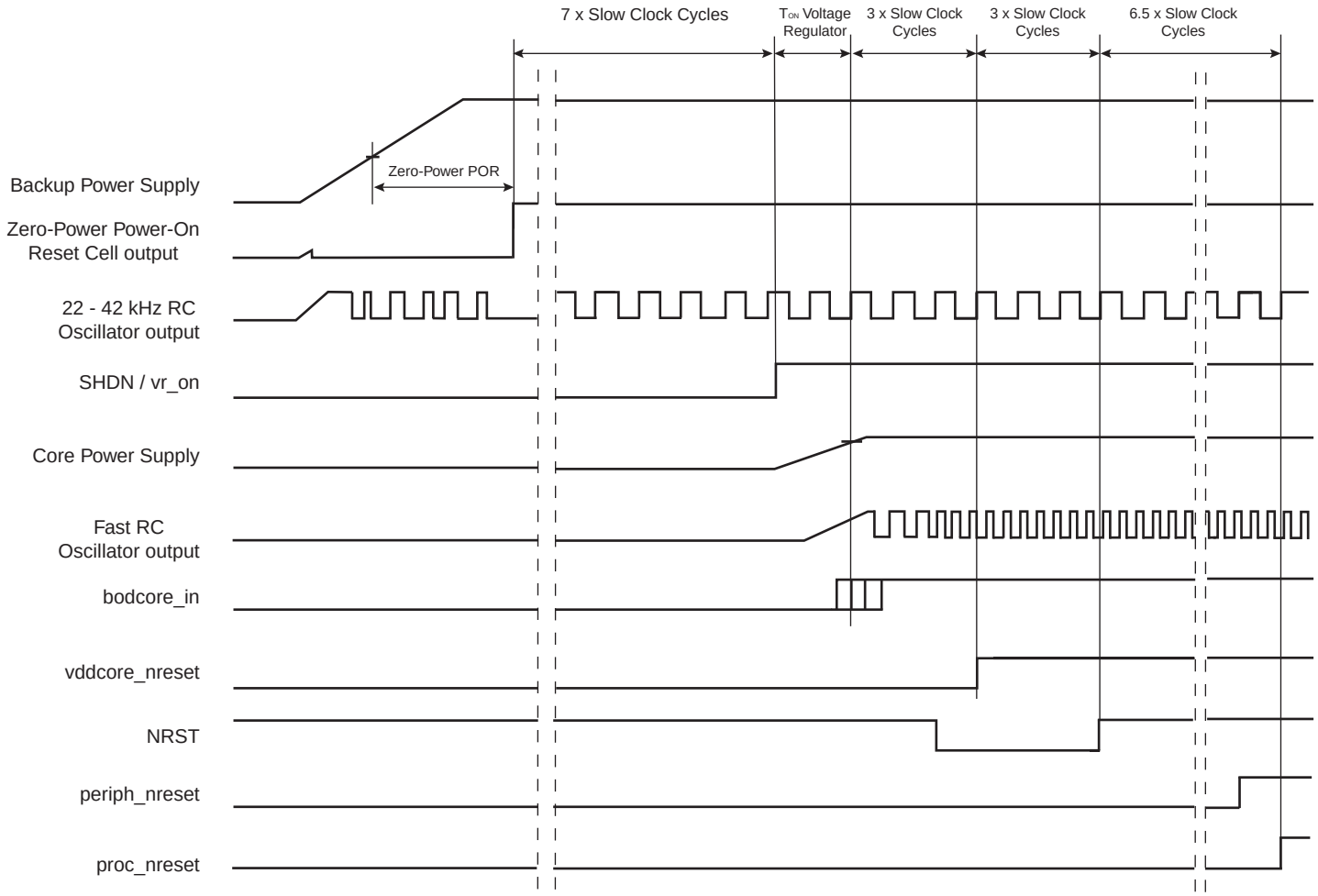
16.4.6 Backup Power Supply Reset

16.4.6.1 Raising the Backup Power Supply

As soon as the backup voltage VDDUTMI rises, the RC oscillator is powered up and the zero-power power-on reset cell maintains its output low as long as VDDUTMI has not reached its target voltage. During this time, the Supply Controller is entirely reset. When the VDDUTMI 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 SHDN pin is asserted and 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 16-5. Raising the VDDUTMI Power Supply



Note: After “proc_nreset” rising, the core starts fecthing instructions from Flash at 4 MHz.

16.4.6.2 NRSTB Asynchronous Reset Pin

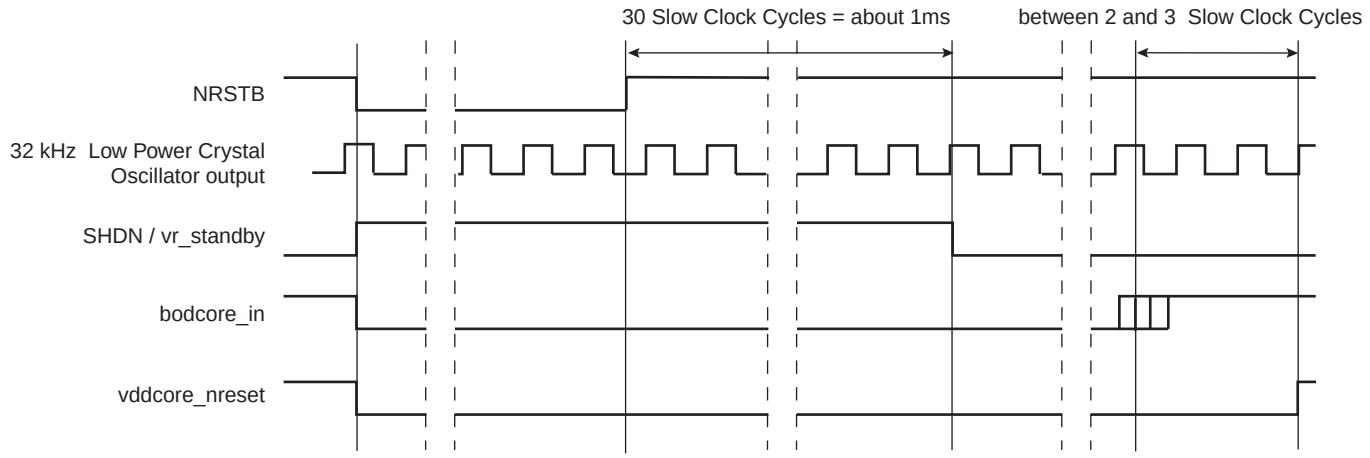
The NRSTB pin is an asynchronous reset input, which acts exactly like the zero-power power-on reset cell.

As soon as NRSTB is tied to GND, the supply controller is reset generating in turn, a reset of the whole system.

When NRSTB is released, the system can start as described in Section 16.4.6.1 ”Raising the Backup Power Supply”.

The NRSTB pin does not need to be driven during power-up phase to allow a reset of the system, it is done by the zero-power power-on cell.

Figure 16-6. NRSTB Reset



Note: periph_nreset, ice_reset and proc_nreset are not shown, but are asserted low thanks to the vddcore_nreset signal controlling the Reset controller.

16.4.6.3 SHDN output pin

As shown in Figure 16-6, the SHDN pin acts like the vr_standby signal making it possible to use the SHDN pin to control external voltage regulator with shutdown capabilities.

16.4.7 Core Reset

The Supply Controller manages the vddcore_nreset signal to the Reset Controller, as described previously in Section 16.4.6 ”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

16.4.7.1 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.

16.4.7.2 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).

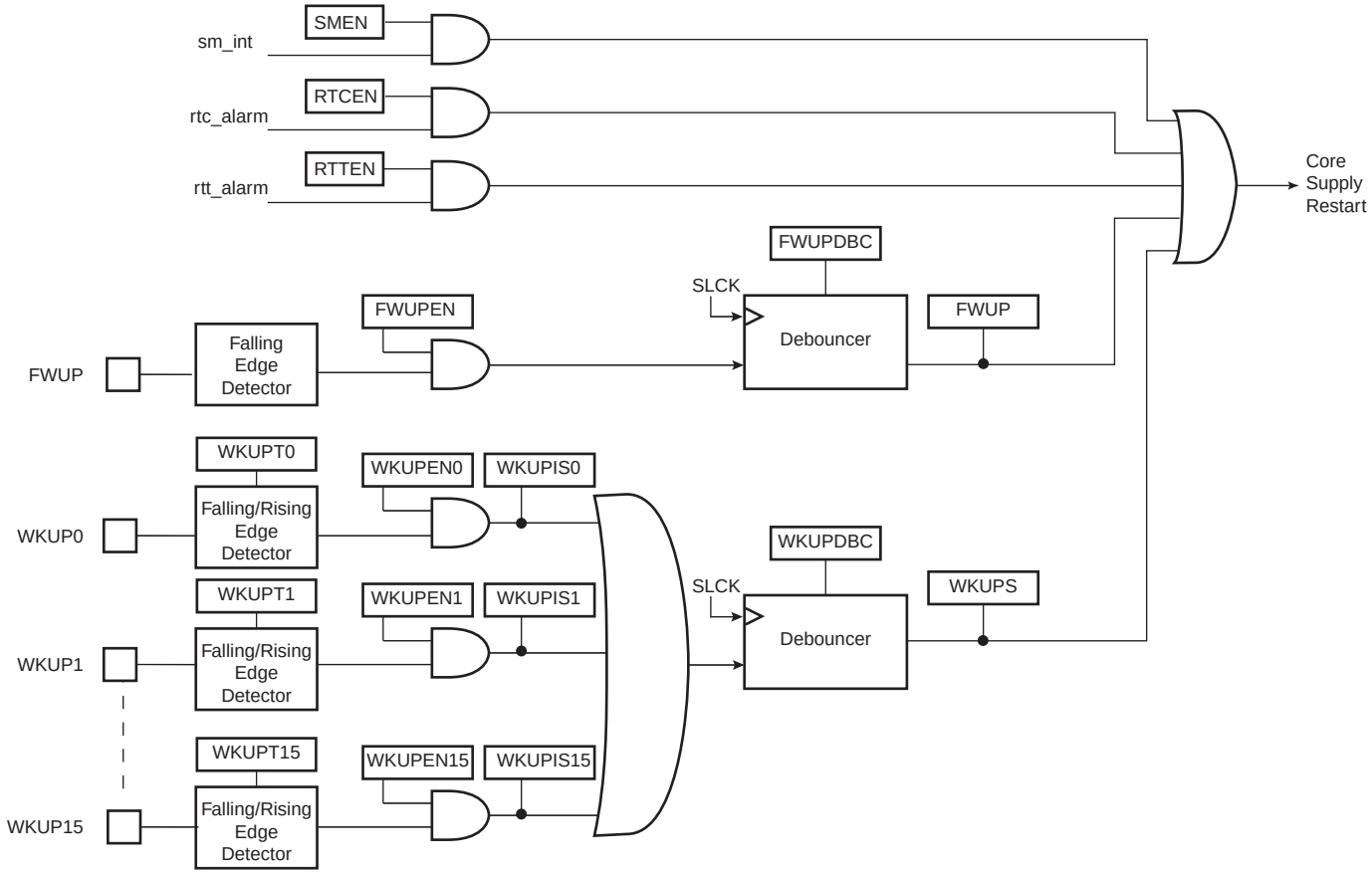
If BODRSTEN is set and the voltage regulation is lost (output voltage of the regulator too low), the vddcore_nreset signal is asserted for a minimum of 1 slow clock cycle and then released if bodcore_in has been reactivated. The BODRSTS bit is set in the Supply Controller Status Register (SUPC_SR) so that the user can know the source of the last reset.

Until bodcore_in is deactivated, the vddcore_nreset signal remains active.

16.4.8 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 16-7. Wake Up Source



16.4.8.1 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.

16.4.8.2 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 16-7, 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.

16.4.8.3 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.

16.4.8.4 Supply Monitor Detection

The supply monitor can generate a wakeup of the core power supply. See Section 16.4.5 ”Supply Monitor”.

16.5 Supply Controller (SUPC) User Interface

The User Interface of the Supply Controller is part of the System Controller User Interface.

16.5.1 System Controller (SYSC) User Interface

Table 16-1. 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	WDT
0x60-0x7C	Real Time Clock	RTC
0x90-0xDC	General Purpose Backup Register	GPBR

16.5.2 System Controller (SYSC) User Interface

Table 16-2. 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_0800
0x18	Reserved			

16.5.3 Supply Controller Control Register

Name: SUPC_CR

Address: 0x400E1A10

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 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**

Should be written to value 0xA5. Writing any other value in this field aborts the write operation.

16.5.4 Supply Controller Supply Monitor Mode Register

Name: SUPC_SMMR

Address: 0x400E1A14

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**

Value	Name	Description
0x0	1_9V	1.9 V
0x1	2_0V	2.0 V
0x2	2_1V	2.1 V
0x3	2_2V	2.2 V
0x4	2_3V	2.3 V
0x5	2_4V	2.4 V
0x6	2_5V	2.5 V
0x7	2_6V	2.6 V
0x8	2_7V	2.7 V
0x9	2_8V	2.8 V
0xA	2_9V	2.9 V
0xB	3_0V	3.0 V
0xC	3_1V	3.1 V
0xD	3_2V	3.2 V
0xE	3_3V	3.3 V
0xF	3_4V	3.4 V

- **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
0x5-0x7	Reserved	Reserved

- **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.

16.5.5 Supply Controller Mode Register

Name: SUPC_MR

Address: 0x400E1A18

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
–	VDDIORDY 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.

- **VDDIORDY: VDDIO Ready**

0 (VDDIO_REMOVED) = VDDIO is removed (used before going to backup mode when backup batteries are used)

1 (VDDIO_PRESENT) = VDDIO is present (used before going to backup mode when backup batteries are used)

If the backup batteries are not used, VDDIORDY must be kept set to 1.

- **ONREG: Voltage Regulator enable**

0 (ONREG_UNUSED) = Voltage Regulator is not used

1 (ONREG_USED) = 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**

Should be written to value 0xA5. Writing any other value in this field aborts the write operation.

16.5.6 Supply Controller Wake Up Mode Register

Name: SUPC_WUMR

Address: 0x400E1A1C

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
–	WKUPDBC			–	FWUPDBC		

7	6	5	4	3	2	1	0
–	–	–	–	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.

- **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

16.5.7 System Controller Wake Up Inputs Register

Name: SUPC_WUIR

Address: 0x400E1A20

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 (NOT_ENABLE) = 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 Transition 0 to 15**

0 (HIGH_TO_LOW) = a high to low level transition on the corresponding wake-up input forces the wake up of the core power supply.

1 (LOW_TO_HIGH) = a low to high level transition on the corresponding wake-up input forces the wake up of the core power supply.

16.5.8 Supply Controller Status Register

Name: SUPC_SR

Address: 0x400E1A24

Access: Read-write

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
–	–	–	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 VDDUTMI higher than its threshold at its last measurement.

1 (LOW) = the supply monitor detected VDDUTMI 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.

- **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.

17. General Purpose Backup Registers (GPBR)

17.1 Description

The System Controller embeds Eight general-purpose backup registers.

17.2 Embedded Characteristics

- Eight 32-bit General Purpose Backup Registers

17.3 General Purpose Backup Registers (GPBR) User Interface

Table 17-1. Register Mapping

Offset	Register	Name	Access	Reset
0x0	General Purpose Backup Register 0	SYS_GPBR0	Read-write	–
...	...	...	...	...
0x1c	General Purpose Backup Register 7	SYS_GPBR7	Read-write	–

17.3.1 General Purpose Backup Register x

Name: SYS_GPBRx

Address: 0x400E1A90 [0] .. 0x400E1AAC [7]

Access: Read-write

31	30	29	28	27	26	25	24
GPBR_VALUEx							

23	22	21	20	19	18	17	16
GPBR_VALUEx							

15	14	13	12	11	10	9	8
GPBR_VALUEx							

7	6	5	4	3	2	1	0
GPBR_VALUEx							

- **GPBR_VALUEx: Value of GPBR x**

18. Enhanced Embedded Flash Controller (EEFC)

18.1 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.

18.2 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 23 MHz
- 16 Lock Bits, Each Protecting a Lock Region
- 3 General-purpose GPNVM Bits
- One-by-one Lock Bit Programming
- Commands Protected by a Keyword
- Erases the Entire Flash
- Possibility of Erasing before Programming
- Locking and Unlocking Operations
- Consecutive Programming and Locking Operations
- Possibility to read the Calibration Bits

18.3 Product Dependencies

18.3.1 Power Management

The Enhanced Embedded Flash Controller (EEFC) is continuously clocked. The Power Management Controller has no effect on its behavior.

18.3.2 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 18-1. Peripheral IDs

Instance	ID
EFC0	6
EFC1	7

18.4 Functional Description

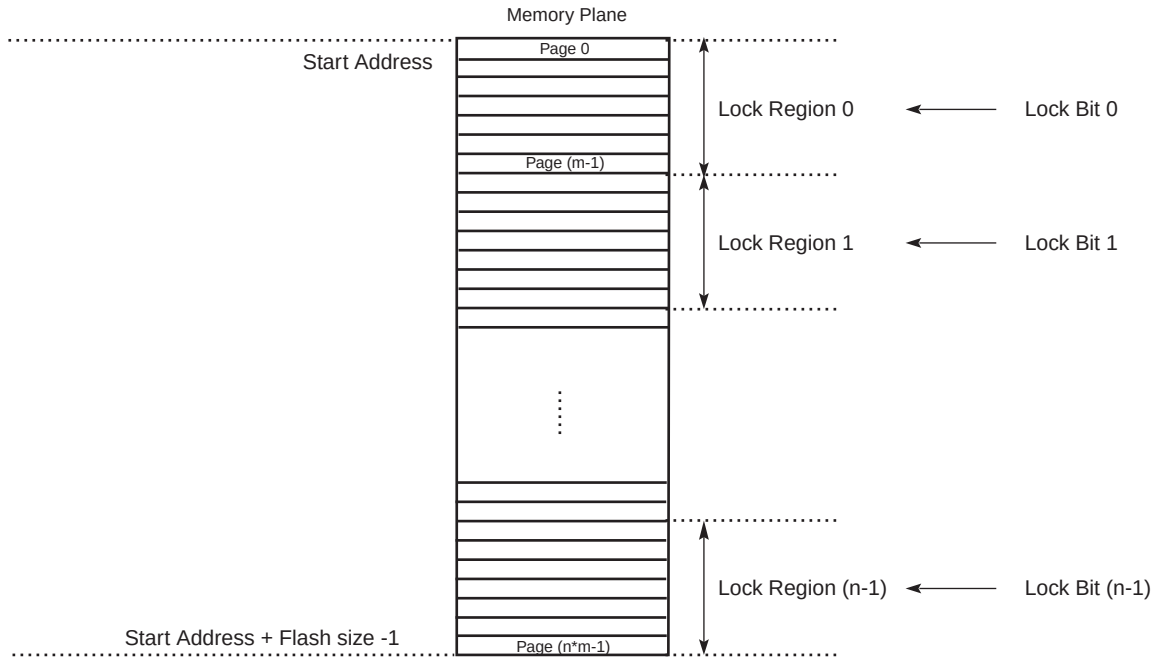
18.4.1 Embedded Flash Organization

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 described in the product definition section. 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 297).

Figure 18-1. Embedded Flash Organization



18.4.2 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.

18.4.2.1 128-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)

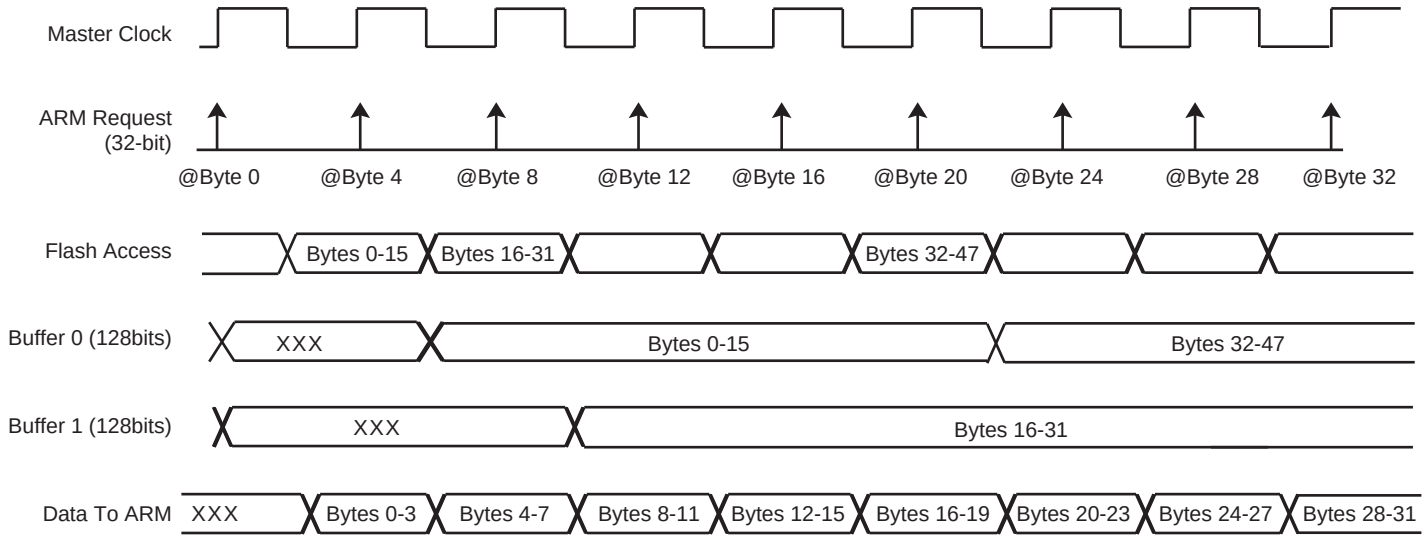
Please refer to the electrical characteristics section of the product datasheet for more details.

18.4.2.2 Code Read Optimization

A system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize sequential Code Fetch.

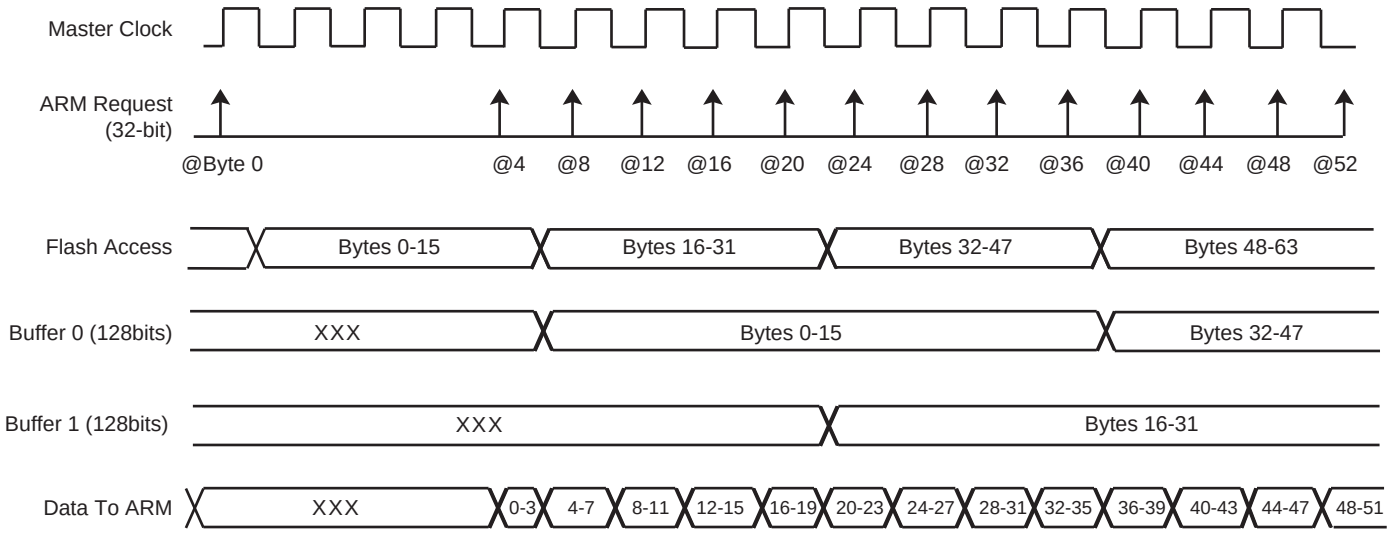
Note: Immediate consecutive code read accesses are not mandatory to benefit from this optimization.

Figure 18-2. Code Read Optimization for FWS = 0



Note: When FWS is equal to 0, all the accesses are performed in a single-cycle access.

Figure 18-3. Code Read Optimization for FWS = 3



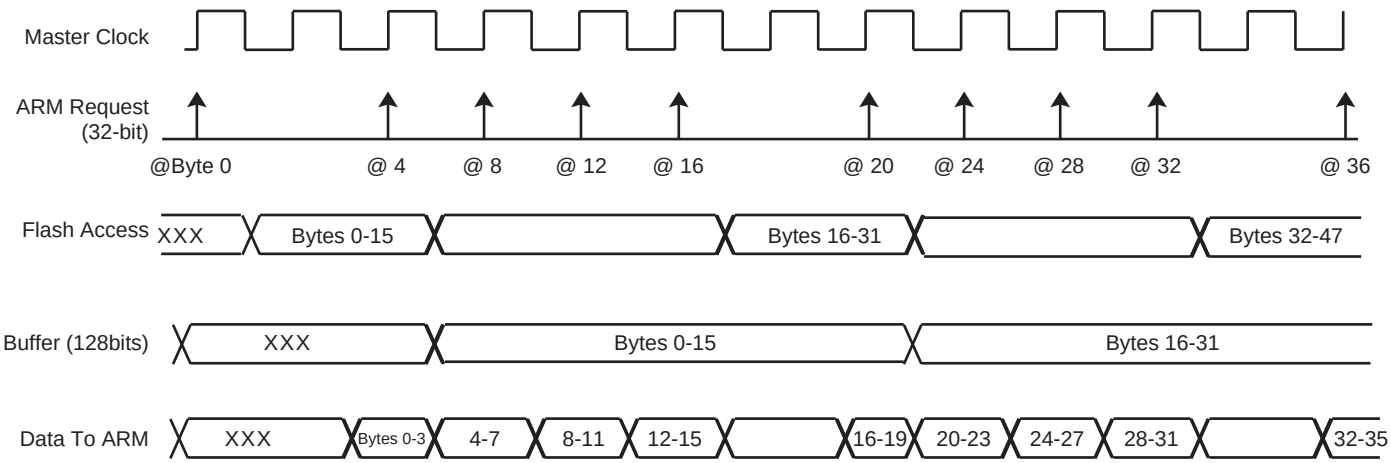
Note: When FWS is included between 1 and 3, in case of sequential reads, the first access takes (FWS+1) cycles, the other ones only 1 cycle.

18.4.2.3 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 18-4).

Note: No consecutive data read accesses are mandatory to benefit from this optimization.

Figure 18-4. Data Read Optimization for FWS = 1



18.4.3 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.

Commands and read operations can be performed in parallel only on different memory planes. Code can be fetched from one memory plane while a write or an erase operation is performed on another.

Table 18-2. 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
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
Stop Read Unique Identifier	0x0F	SPUI
Get CALIB Bit	0x10	GCALB

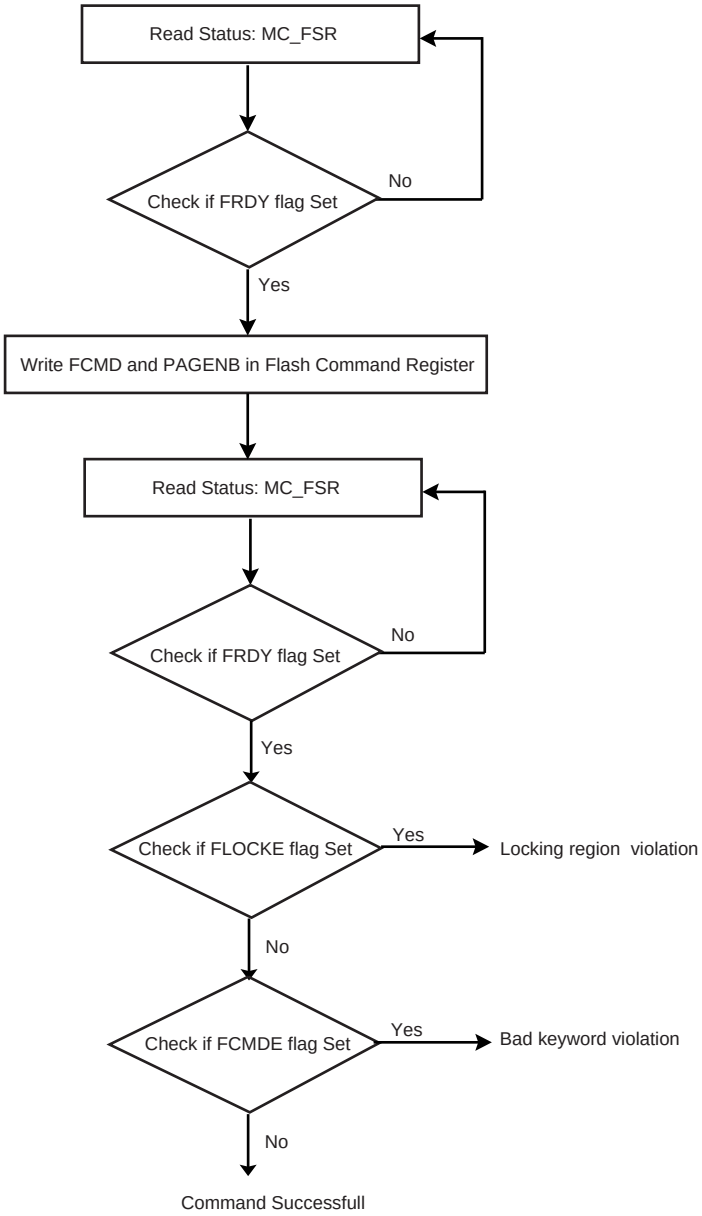
In order to perform one of these commands, the Flash Command Register (EEFC_FCR) has to be written with the correct command using the FCMD field. As soon as the EEFC_FCR register is written, **the FRDY flag and the FVALUE field in the EEFC_FRR register are automatically cleared**. Once the current command 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 18-5. Command State Chart



18.4.3.1 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 18-3. 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.
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.
...		

18.4.3.2 Write Commands

Several commands can be used to program the Flash.

Flash technology requires that an erase is 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 Section "The Partial Programming mode works only with 128-bit (or higher) boundaries. It cannot be used with boundaries lower than 128 bits (8, 16 or 32-bit for example)."). 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:

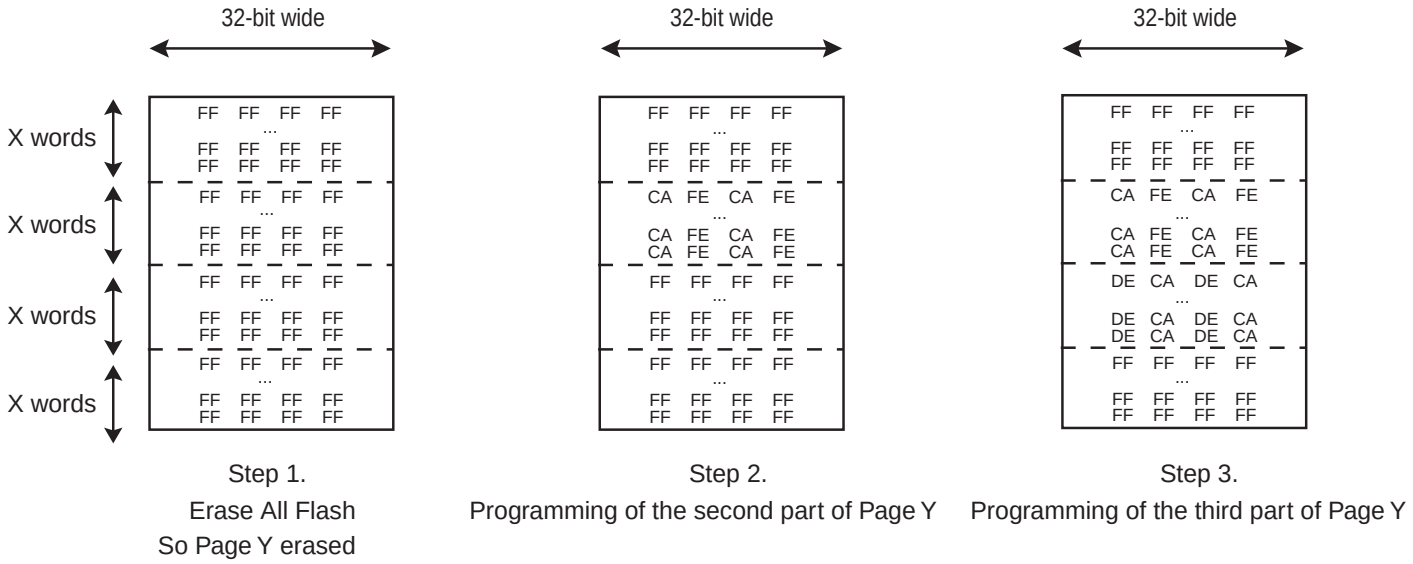
- Write the full page, at any page address, within the internal memory area address space.
- Programming starts as soon as the page number and the programming command are written to the Flash Command Register. The FRDY bit in the Flash Programming Status Register (EEFC_FSR) is automatically cleared.
- 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:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.
- a Lock Error: the page to be programmed belongs to a locked region. A command must be previously run to unlock the corresponding region.

By using the WP command, a page can be programmed in several steps if it has been erased before (see Figure 18-6).

Figure 18-6. Example of Partial Page Programming



The Partial Programming mode works only with 128-bit (or higher) boundaries. It cannot be used with boundaries lower than 128 bits (8, 16 or 32-bit for example).

18.4.3.3 Erase Commands

Erase commands are allowed only on unlocked regions.

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.
- 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:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.
- a 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.

18.4.3.4 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.
- If the lock bit number is greater than the total number of lock bits, then the command has no effect. The result of the SLB command can be checked running a GLB (Get Lock Bit) command.

One error can be detected in the EEFC_FSR register after a programming sequence:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.

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.
- If the lock bit number is greater than the total number of lock bits, then the command has no effect.

One error can be detected in the EEFC_FSR register after a programming sequence:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.

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:

- a 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 lock bit command is performed.

18.4.3.5 GPNVM Bit

GPNVM bits do not interfere with the embedded Flash memory plane. Refer to the product definition section for information on the 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 GPVNM 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.
- If the GPNVM bit number is greater than the total number of GPNVM bits, then the command has no effect. The result of the SGPB command can be checked by running a GGPB (Get GPNVM Bit) command.

One error can be detected in the EEFC_FSR register after a programming sequence:

- A Command Error: a bad keyword has been written in the EEFC_FCR register.

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.
- If the GPNVM bit number is greater than the total number of GPNVM bits, then the command has no effect.

One error can be detected in the EEFC_FSR register after a programming sequence:

- A Command Error: a bad keyword has been written in the EEFC_FCR register.

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:

- a 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.

18.4.3.6 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 18-2). 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:

RC Calibration Frequency	EEFC_FRR Bits
8 MHz output	[28 - 22]
12 MHz output	[38 - 32]

The RC calibration for 4 MHz is set to 1,000,000.

18.4.3.7 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.

18.4.3.8 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. So, at the address 0x80000-0xFFFFF.
- 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).

18.5 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 and 0x400E0C00.

Table 18-4. Register Mapping

Offset	Register	Name	Access	Reset State
0x00	EEFC Flash Mode Register	EEFC_FMR	Read-write	0x0
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	–	–	–

18.5.1 EEFC Flash Mode Register

Name: EEFC_FMR

Address: 0x400E0A00 (0), 0x400E0C00 (1)

Access: Read-write

Offset: 0x00

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	FAM
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
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:

Number of cycles for Read/Write operations = FWS+1

- **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.

18.5.2 EEFC Flash Command Register

Name: EEFC_FCR

Address: 0x400E0A04 (0), 0x400E0C04 (1)

Access: Write-only

Offset: 0x04

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

- **FCMD: Flash Command**

This field defines the flash commands. Refer to “Flash Commands” on page 295.

- **FARG: Flash Command Argument**

Erase command	For erase all command, this field is meaningless.
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.
Get commands	Field is meaningless.
Unique Identifier commands	Field is meaningless.

- **FKEY: Flash Writing Protection Key**

This field should be written with the value 0x5A to enable 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.

18.5.3 EEFC Flash Status Register

Name: EEFC_FSR

Address: 0x400E0A08 (0), 0x400E0C08 (1)

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
–	–	–	–	–	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 flags 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.

18.5.4 EEFC Flash Result Register

Name: EEFC_FRR

Address: 0x400E0A0C (0), 0x400E0C0C (1)

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.

19. Fast Flash Programming Interface (FFPI)

19.1 Description

The Fast Flash Programming Interface provides solutions for 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.

19.2 Parallel Fast Flash Programming

19.2.1 Device Configuration

In Fast Flash Programming Mode, the device is in a specific test mode. Only a certain set of pins is significant. Other pins must be left unconnected. The Fast Flash Programming Interface is enabled and the Fast Programming Mode is entered when TST, PA0, PA1 are set to high, PA2 and PA3 are set to low and NRST is toggled from 0 to 1.

Figure 19-1. Parallel Programming Interface

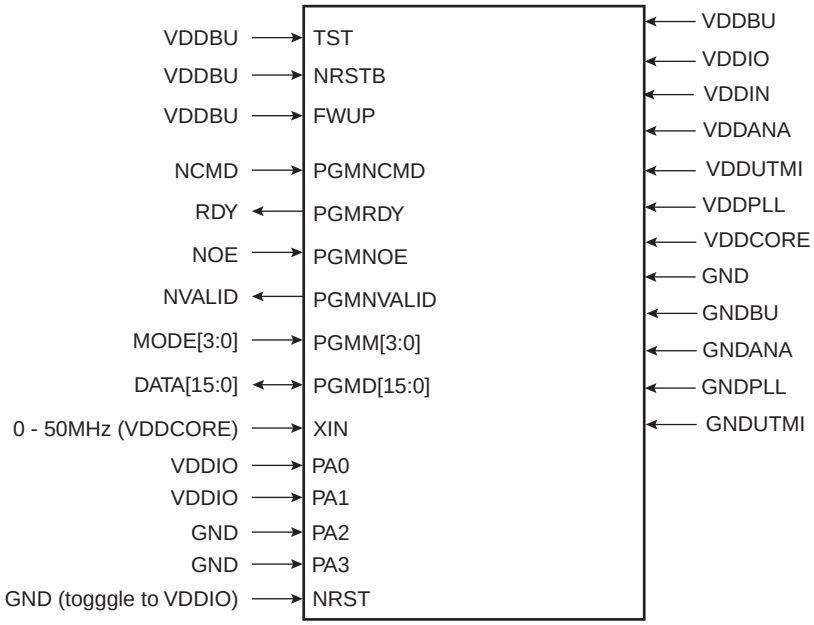


Table 19-1. Signal Description List

Signal Name	Function	Type	Active Level	Comments
Power				
VDDIO	I/O Lines Power Supply	Power		Apply external 3.0V-3.6V
VDDBU	Backup I/O Lines Power Supply	Power		Apply external 3.0V-3.6V
VDDUTMI	UTMI+ Interface Power Supply	Power		Apply external 3.0V-3.6V
VDDANA	ADC Analog Power Supply	Power		Apply external 3.0V-3.6V
VDDIN	Voltage Regulator Input	Power		Apply external 3.0V-3.6V
VDDCORE	Core Power Supply	Power		Apply external 1.65V-1.95V
VDDPLL	PLLs and Oscillator Power Supply	Power		Apply external 1.65V-1.95V
GND	Ground	Ground		
GNDPLL	Ground	Ground		
GNDBU	Ground	Ground		
GNDANA	Ground	Ground		
GNDUTMI	Ground	Ground		
Clocks				
XIN	Clock Input	Input		0 to 50MHz (0-VDDCORE square wave)
Test				
TST	Test Mode Select	Input	High	Must be connected to VDDIO
NRSTB	Asynchronous Microcontroller Reset	Input	High	Must be connected to VDDIO
FWUP	Wake-up pin	Input	High	Must be connected to VDDIO
PIO				
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 Table 19-3)	Input		Pulled-up input at reset
PGMD[15:0]	Bi-directional data bus	Input/Output		Pulled-up input at reset
PA0		Input		Must be connected to VDDIO at power-up
PA1		Input		Must be connected to VDDIO at power-up
PA2		Input		Must be connected to GND at power-up
PA3		Input		Must be connected to GND at power-up
NRST		Input		Must be connected at start-up. And toggle at VDDIO

The table below shows the signal assignment of the PIO lines in FFPI mode

Table 19-2. FFPI PIO Assignment

FFPI Signal	PIO Used
PGMNCMD	PA0
PGMRDY	PA1
PGMNOE	PA2
PGMNVALID	PA3
PGMM[0]	PA4
PGMM[1]	PA5
PGMM[2]	PA6
PGMM[3]	PA7
PGMD[0]	PA8
PGMD[1]	PA9
PGMD[2]	PA10
PGMD[3]	PA11
PGMD[4]	PA12
PGMD[5]	PA13
PGMD[6]	PA14
PGMD[7]	PA15
PGMD[8]	PA16
PGMD[9]	PA17
PGMD[10]	PA18
PGMD[11]	PA19
PGMD[12]	PA20
PGMD[13]	PA21
PGMD[14]	PA22
PGMD[15]	PA23

19.2.2 Signal Names

Depending on the MODE settings, DATA is latched in different internal registers.

Table 19-3. Mode Coding

MODE[3:0]	Symbol	Data
0000	CMDE	Command Register
0001	ADDR0	Address Register LSBs
0010	ADDR1	Address Register MSBs
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 19-4. 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
0x0016	SEFC	Select EEFC Controller(1)
0x001E	GVE	Get Version

Note: 1. Applies to 256 kbytes Flash version (dual EEFC)

19.2.3 Entering Programming Mode

The following algorithm puts the device in Parallel Programming Mode:

- Apply GND, TST, NRTSB, FWUP, PA0, PA1, PA2, PA3, NRST and the supplies as described in Table 19-1, “Signal Description List,” on page 309.
- Wait for 20 ms
- Toggle NRST from 0 to 1 (GND to VDDIO).
- Apply XIN clock
- Wait for 20 ms
- Start a read or write handshaking.

19.2.4 Programmer Handshaking

A 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.

19.2.4.1 Write Handshaking

For details on the write handshaking sequence, refer to Figure 19-2 and Table 19-5.

Figure 19-2. Parallel Programming Timing, Write Sequence

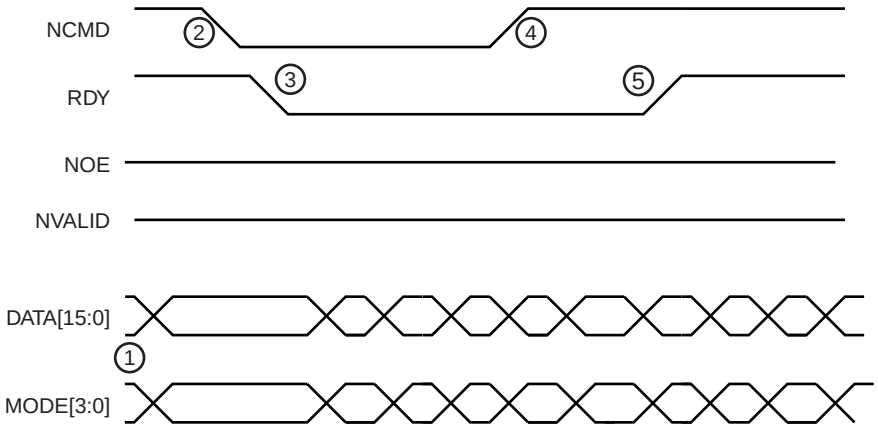


Table 19-5. Write Handshake

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

19.2.4.2 Read Handshaking

For details on the read handshaking sequence, refer to Figure 19-3 and Table 19-6.

Figure 19-3. Parallel Programming Timing, Read Sequence

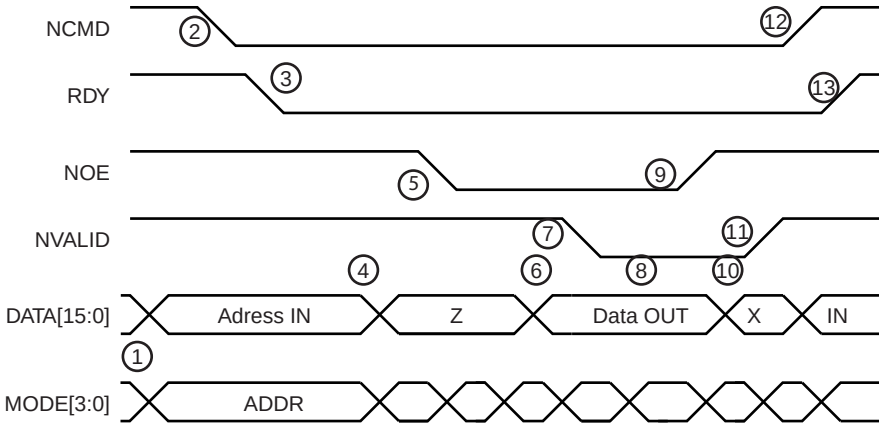


Table 19-6. Read Handshake

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

19.2.5 Device Operations

Several commands on the Flash memory are available. These commands are summarized in Table 19-4 on page 311. 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.

19.2.5.1 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 19-7. 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++
...	...	...	...

19.2.5.2 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 19-8. 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
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.

19.2.5.3 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 19-9. Full Erase Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	EA
2	Write handshaking	DATA	0

19.2.5.4 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.

Likewise, the **Clear Lock** command **(CLB)** is used to clear lock bits.

Table 19-10. 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 nth lock bit is active when the bit n of the bit mask is set..

Table 19-11. 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

19.2.5.5 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.

Likewise, the **Clear GPNVM** command **(CGPB)** is used to clear general-purpose NVM bits. All the general-purpose NVM bits are also cleared by the EA command. The general-purpose NVM bit is deactivated when the corresponding bit in the pattern value is set to 1.

Table 19-12. 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 nth GP NVM bit is active when bit n of the bit mask is set..

Table 19-13. 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

19.2.5.6 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.

The SAM3X/A security bit is controlled by the EEFC0. To use the **Set Security Bit** command, the EEFC0 must be selected using the **Select EFC** command.

Table 19-14. 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.

19.2.5.7 SAM3X/A Flash Select EEFC Command

The commands WPx, EA, xLB, xFB are executed using the current EFC controller. The default EEFC controller is EEFC0. The **Select EEFC** command (SEFC) allows selection of the current EEFC controller.

Table 19-15. Select EFC Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	SEFC
2	Write handshaking	DATA	0 = Select EEFC0 1 = Select EEFC1

19.2.5.8 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 19-16. 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++
...	...	...	...

19.2.5.9 Get Version Command

The **Get Version** (GVE) command retrieves the version of the FFPI interface.

Table 19-17. Get Version Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	GVE
2	Write handshaking	DATA	Version

20. SAM3X/A Boot Program

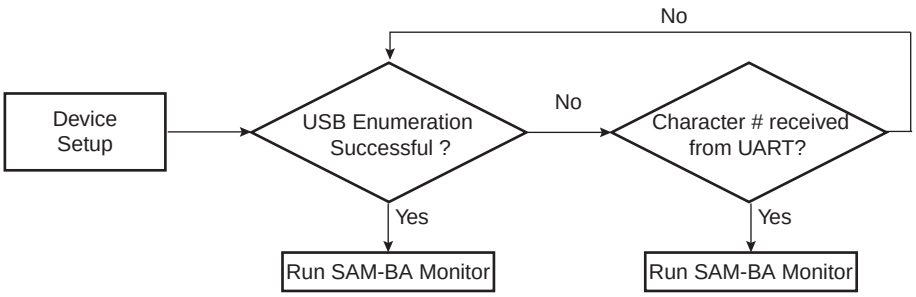
20.1 Description

The SAM-BA Boot Program integrates an array of programs permitting download and/or upload into the different memories of the product.

20.2 Flow Diagram

The Boot Program implements the algorithm illustrated in Figure 20-1.

Figure 20-1. 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 12 MHz 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 12 MHz (taking into account the frequency range of the 32 kHz RC oscillator). If the frequency is 12 MHz, USB activation is allowed, else (no clock or frequency other than 12 MHz), 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.

20.3 Device Initialization

The initialization sequence is the following:

1. Stack setup
2. Set up the Embedded Flash Controller
3. External Clock detection (quartz or external clock on XIN)
4. If quartz or external clock is 12.000 MHz, allow USB activation
5. Else, does not allow USB activation and use internal RC 12 MHz
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. UPLL setup in case of USB activation allowed
11. Disable of the Watchdog
12. Initialization of the UART (115200 bauds, 8, N, 1)
13. Initialization of the USB Device Port (in case of USB activation allowed)
14. Wait for one of the following events
    a. Check if USB device enumeration has occurred
    b. Check if characters have been received in the UART
15. Jump to SAM-BA Monitor (see Section 20.4 ”SAM-BA Monitor”)

20.4 SAM-BA Monitor

Once the communication interface is identified, the monitor runs in an infinite loop waiting for different commands as shown in Table 20-1.

Table 20-1. Commands Available through the SAM-BA Boot

Command	Action	Argument(s)	Example
N	set Normal mode	No argument	**N**#
T	set Terminal mode	No argument	**T**#
O	write a byte	Address, Value#	**O**200001,CA#
o	read a byte	Address,#	**o**200001,#
H	write a half word	Address, Value#	**H**200002,CAFE#
h	**read a half word**	**Address,#**	**h**200002,#
W	**write a word**	**Address, Value#**	**W**200000,CAFEDECA#
w	**read a word**	**Address,#**	**w**200000,#
S	**send a file**	**Address,#**	**S**200000,#
R	**receive a file**	**Address, NbOfBytes#**	**R**200000,1234#
G	**go**	**Address#**	**G**200200#
V	**display version**	**No argument**	**V**#

- 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.
  - *Output*: '>'.
- 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 following by '>'
- Send a file (**S**): Send a file to a specified address
  - *Address*: Address in hexadecimal
  - *Output*: '>'.

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
  - *Output*: '>'
- Go (**G**): Jump to a specified address and execute the code
  - *Address*: Address to jump in hexadecimal
  - *Output*: '>'
- Get Version (**V**): Return the SAM-BA boot version
  - *Output*: '>'

20.4.1 UART Serial Port

Communication is performed through the UART 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 20.5 ”Hardware and Software Constraints”.

20.4.2 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.

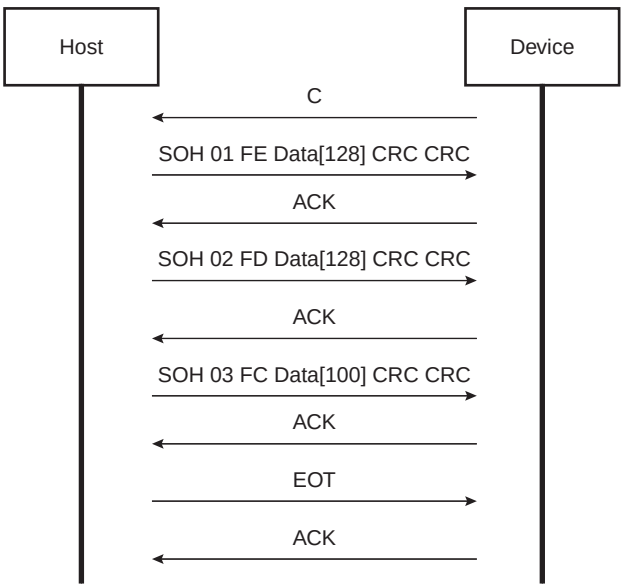
Xmodem protocol with CRC is accurate provided both sender and receiver report successful transmission. Each block of the transfer looks like:

<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 20-2 shows a transmission using this protocol.

Figure 20-2. Xmodem Transfer Example



20.4.3 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® beginning with Windows 98SE. The CDC document, available at 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 on www.usb.org.

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 application note *Basic USB Application* (Atmel literature number 6123) for more details.

20.4.3.1 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 20-2. 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 20-3. 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.

20.4.3.2 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.

20.4.4 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 EEFC_FSR).

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 (0x00100008).

This function takes one argument in parameter: the command to be sent to the EEFC.

This function returns the value of the EEFC_FSR.

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);

}
```

20.5 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.
- UART requirements:
  - 12.000 MHz quartz or 12.000 MHz external clock on XIN, or
  - No quartz or external clock on XIN, or
  - Below 5.0 MHz quartz or below 5.0 MHz external clock on XIN
- USB requirements:
  - 12.000 MHz quartz or 12.000 MHz external clock on XIN. 12 MHz must be ±500 ppm and 1.8V Square Wave Signal.

Table 20-4. Pins Driven during Boot Program Execution

Peripheral	Pin	PIO Line
UART	URXD	PA8
UART	UTXD	PA9

21. Bus Matrix (MATRIX)

21.1 Description

Bus Matrix implements a multi-layer AHB, based on AHB-Lite protocol, that enables parallel access paths between multiple AHB masters and slaves in a system, which increases the overall bandwidth. Bus Matrix interconnects 6 AHB Masters to 9 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 the ARM Advance Peripheral Bus (APB) and provides a Chip Configuration User Interface with Registers that allow the Bus Matrix to support application specific features.

21.2 Embedded Characteristics

21.2.1 Matrix Masters

The Bus Matrix of the SAM3A/X series product manages 5 (SAM3A) or 6 (SAM3X) 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 decoding.

Table 21-1. List of Bus Matrix Masters

Master 0	Cortex-M3 Instruction/Data
Master 1	Cortex-M3 System
Master 2	Peripheral DMA Controller (PDC)
Master 3	USB OTG High Speed DMA
Master 4	DMA Controller
Master 5	Ethernet MAC (AT91SAM3X)

21.2.2 Matrix Slaves

The Bus Matrix of the SAM3A/X series product manages 9 slaves. Each slave has its own arbiter, allowing a different arbitration per slave.

Table 21-2. List of Bus Matrix Slaves

Slave 0	Internal SRAM0
Slave 1	Internal SRAM1
Slave 2	Internal ROM
Slave 3	Internal Flash
Slave 4	USB High Speed Dual Port RAM (DPR)
Slave 5	Nand Flash Controller RAM
Slave 6	External Bus Interface
Slave 7	High Speed Peripheral Bridge
Slave 8	Low Speed Peripheral Bridge

21.2.3 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 USB High speed DMA to the Internal Peripherals. Thus, these paths are forbidden or simply not wired, and shown as “-” in the following table.

Table 21-3. SAM3A/X series Master to Slave Access

Masters		0	1	2	3	4	5
Slaves		**Cortex-M3 I/D Bus**	**Cortex-M3 S Bus**	**PDC**	**USB High Speed DMA**	**DMA Controller**	**EMAC DMA**
0	Internal SRAM0	-	X	X	X	X	X
1	Internal SRAM1	-	X	X	X	X	X
2	Internal ROM	X	-	X	X	X	X
3	Internal Flash	X	-	-	-	-	-
4	USB High Speed Dual Port RAM	-	X	-	-	X	-
5	Nand Flash Controller RAM	-	X	X	X	X	X
6	External Bus Interface	-	X	X	X	X	X
7	High Speed Peripheral Bridge	-	X	X	-	X	-
8	Low Speed Peripheral Bridge	-	X	-	-	X	-

21.3 Memory Mapping

Bus Matrix provides one decoder for every AHB Master Interface. The decoder offers each AHB Master several memory mappings. In fact, depending on the product, each memory area may be assigned to several slaves. Booting at the same address while using different AHB slaves (i.e. internal ROM or internal Flash) becomes possible.

The Bus Matrix user interface provides Master Remap Control Register (MATRIX_MRCR) that allows to perform remap action for every master independently.

21.4 Special Bus Granting Techniques

The Bus Matrix provides some speculative bus granting techniques in order to anticipate access requests from some masters. This mechanism allows to reduce latency at first accesses of a burst or single transfer. The bus granting mechanism allows to set a 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 and fixed default master.

21.4.1 No Default Master

At the end of the current access, if no other request is pending, the slave is disconnected from all masters. No Default Master suits low power mode.

21.4.2 Last Access Master

At 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.

21.4.3 Fixed Default Master

At the end of the current access, if no other request is pending, the slave connects to its fixed default master. Unlike last access master, the fixed master doesn't change unless the user modifies it by a software action (field FIXED_DEFMSTR of the related MATRIX_SCFG).

To change from one kind of default master to another, the Bus Matrix user interface provides the Slave Configuration Registers, one for each slave, that allow to 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 allows to choose the default master type (no default, last access master, fixed default master) whereas the 4-bit FIXED_DEFMSTR field allows to choose a fixed default master provided that DEFMSTR_TYPE is set to fixed default master. Please refer to the Bus Matrix user interface description.

21.5 Arbitration

The Bus Matrix provides an arbitration mechanism that allows to reduce latency when conflict cases occur, basically when two or more masters try to access the same slave at the same time. One arbiter per AHB slave is provided, allowing to arbitrate each slave differently.

The Bus Matrix provides the user the possibility to choose between 2 arbitration types for each slave:

1. Round-Robin Arbitration (the default)
2. Fixed Priority Arbitration

This choice is given through the ARBT field of the Slave Configuration Registers (MATRIX_SCFG).

Each algorithm may be complemented by selecting a default master configuration for each slave.

When re-arbitration has to be done, it is realized only under specific conditions as detailed in the following paragraph.

21.5.1 Arbitration Rules

Each arbiter has the ability to arbitrate between two or more different master's 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 21.5.1.1 "Undefined Length Burst Arbitration" on page 325").
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 21.5.1.2 "Slot Cycle Limit Arbitration" on page 326).

21.5.1.1 Undefined Length Burst Arbitration

In order to avoid too long slave handling during undefined length bursts (INCR), the Bus Matrix provides specific logic in order to re-arbitrate before the end of the INCR transfer.

A predicted end of burst is used as for defined length burst transfer, which is selected between the following:

1. Infinite: no predicted end of burst is generated and therefore INCR burst transfer will never be broken.
2. Four beat bursts: predicted end of burst is generated at the end of each four beat boundary inside INCR transfer.
3. Eight beat bursts: predicted end of burst is generated at the end of each eight beat boundary inside INCR transfer.

4. Sixteen beat bursts: predicted end of burst is generated at the end of each sixteen beat boundary inside INCR transfer.

This selection can be done through the ULBT field of the Master Configuration Registers (MATRIX_MCFG).

21.5.1.2 Slot Cycle Limit Arbitration

The Bus Matrix contains specific logic to break too long accesses such as very long bursts on a very slow slave (e.g. an external low speed memory). At the beginning of the burst access, 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 reaches zero, the arbiter has the ability to re-arbitrate at the end of the current byte, half word or word transfer.

21.5.2 Round-Robin Arbitration

This algorithm allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave in a round-robin manner. If two or more master's requests arise at the same time, the master with the lowest number is first serviced then the others are serviced in a round-robin manner.

There are three round-robin algorithm implemented:

- Round-Robin arbitration without default master
- Round-Robin arbitration with last access master
- Round-Robin arbitration with fixed default master

21.5.2.1 Round-Robin arbitration without default master

This is the main algorithm used by Bus Matrix arbiters. It allows the Bus Matrix to dispatch requests from different masters to the same slave in a pure round-robin manner. At the end of the current access, if no other request is pending, the slave is disconnected from all masters. This configuration incurs one latency cycle for the first access of a burst. Arbitration without default master can be used for masters that perform significant bursts.

21.5.2.2 Round-Robin arbitration with last access master

This is a biased round-robin algorithm used by Bus Matrix arbiters. It allows the Bus Matrix to remove the one latency cycle for the last master that accessed the slave. In fact, at the end of the current transfer, if no other master request is pending, the slave remains connected to the last master that performs the access. Other non privileged masters will still get one latency cycle if they want to access the same slave. This technique can be used for masters that mainly perform single accesses.

21.5.2.3 Round-Robin arbitration with fixed default master

This is another biased round-robin algorithm, it allows the Bus Matrix arbiters to remove the one latency cycle for the fixed default master per slave. At the end of the current access, the slave remains connected to its fixed default master. Every request attempted by this fixed default master will not cause any latency whereas other non privileged masters will still get one latency cycle. This technique can be used for masters that mainly perform single accesses.

21.5.3 Fixed Priority Arbitration

This algorithm 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. If two or more master's requests are active at the same time, the master with the highest priority number is serviced first. If two or more master's requests with the same priority are active at the same time, the master with the highest number is serviced first.

For each slave, the priority of each master may be defined through the Priority Registers for Slaves (MATRIX_PRAS and MATRIX_PRBS).

21.6 System I/O Configuration

The System I/O Configuration register (CCFG_SYSIO) allows to configure I/O lines in System I/O mode (such as ERASE) 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, has no effect. However, the direction (input or output), pull-up and other mode control, is still managed by the PIO controller.

21.7 Write Protect Registers

To prevent any single software error that may corrupt MATRIX behavior, the entire MATRIX address space from address offset 0x000 to 0x1FC can be write-protected by setting the WPEN bit in the MATRIX Write Protect Mode Register (MATRIX_WPMR).

If a write access to anywhere in the MATRIX address space from address offset 0x000 to 0x1FC is detected, then the WPVS flag in the MATRIX Write Protect Status Register (MATRIX_WPSR) is set and the WPVSRC field indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the 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 For Slaves”

“Bus Matrix Master Remap Control Register”

“System I/O Configuration Register”

21.8 Bus Matrix (MATRIX) User Interface

Table 21-4. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Master Configuration Register 0	MATRIX_MCFG0	Read-write	0x00000000
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 - 0x003C	Reserved	–	–	–
0x0040	Slave Configuration Register 0	MATRIX_SCFG0	Read-write	0x00010010
0x0044	Slave Configuration Register 1	MATRIX_SCFG1	Read-write	0x00050010
0x0048	Slave Configuration Register 2	MATRIX_SCFG2	Read-write	0x00000010
0x004C	Slave Configuration Register 3	MATRIX_SCFG3	Read-write	0x00000010
0x0050	Slave Configuration Register 4	MATRIX_SCFG4	Read-write	0x00000010
0x0054	Slave Configuration Register 5	MATRIX_SCFG5	Read-write	0x00000010
0x0058	Slave Configuration Register 6	MATRIX_SCFG6	Read-write	0x00000010
0x005C	Slave Configuration Register 7	MATRIX_SCFG7	Read-write	0x00000010
0x0060	Slave Configuration Register 8	MATRIX_SCFG8	Read-write	0x00000010
0x0064 - 0x007C	Reserved	–	–	–
0x0080	Priority Register A for Slave 0	MATRIX_PRAS0	Read-write	0x00000000
0x0084	Reserved	–	–	–
0x0088	Priority Register A for Slave 1	MATRIX_PRAS1	Read-write	0x00000000
0x008C	Reserved	–	–	–
0x0090	Priority Register A for Slave 2	MATRIX_PRAS2	Read-write	0x00000000
0x0094	Reserved	–	–	–
0x0098	Priority Register A for Slave 3	MATRIX_PRAS3	Read-write	0x00000000
0x009C	Reserved	–	–	–
0x00A0	Priority Register A for Slave 4	MATRIX_PRAS4	Read-write	0x00000000
0x00A4	Reserved	–	–	–
0x00A8	Priority Register A for Slave 5	MATRIX_PRAS5	Read-write	0x00000000
0x00AC	Reserved	–	–	–
0x00B0	Priority Register A for Slave 6	MATRIX_PRAS6	Read-write	0x00000000
0x00B4	Reserved	–	–	–
0x00B8	Priority Register A for Slave 7	MATRIX_PRAS7	Read-write	0x00000000
0x00BC	Reserved	–	–	–
0x00C0	Priority Register A for Slave 8	MATRIX_PRAS8	Read-write	0x00000000

Table 21-4. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x00C4- 0x00FC	Reserved	–	–	–
0x0100	Master Remap Control Register	MATRIX_MRCR	Read-write	0x00000000
0x0104 - 0x0110	Reserved	–	–	–
0x0114	System I/O Configuration register	CCFG_SYSIO	Read/Write	0x00000000
0x0118 - 0x01E0	Reserved	–	–	–
0x1E4	Write Protect Mode Register	MATRIX_WPMR	Read-write	0x0
0x1E8	Write Protect Status Register	MATRIX_WPSR	Read-only	0x0
0x01EC-0x01FC	Reserved	–	–	–

21.8.1 Bus Matrix Master Configuration Registers

Name: MATRIX_MCFG0..MATRIX_MCFG5

Address: 0x400E0400

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: Infinite Length Burst

No predicted end of burst is generated and therefore INCR bursts coming from this master cannot be broken.

1: Single Access

The undefined length burst is treated as a succession of single access allowing rearbitration at each beat of the INCR burst.

2: Four Beat Burst

The undefined length burst is split into 4 beats burst allowing rearbitration at each 4 beats burst end.

3: Eight Beat Burst

The undefined length burst is split into 8 beats burst allowing rearbitration at each 8 beats burst end.

4: Sixteen Beat Burst

The undefined length burst is split into 16 beats burst allowing rearbitration at each 16 beats burst end.

21.8.2 Bus Matrix Slave Configuration Registers

Name: MATRIX_SCFG0..MATRIX_SCFG8

Address: 0x400E0440

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	ARBT	

23	22	21	20	19	18	17	16
–	–	–	FIXED_DEFMSTR			DEFMSTR_TYPE	

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

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 Number of Allowed Cycles for a Burst**

When the SLOT_CYCLE limit is reach for a burst it may be broken by another master trying to access this slave.

This limit has been placed to avoid locking very slow slave by when very long burst are used.

This limit should not be very small though. Unreasonable small value will break every burst and Bus Matrix will spend its time to arbitrate without performing any data transfer. 16 cycles is a reasonable value for SLOT_CYCLE.

- **DEFMSTR_TYPE: Default Master Type**

0: No Default Master

At the end of current slave access, if no other master request is pending, the slave is disconnected from all masters.

This results in having a one cycle latency for the first access of a burst transfer or for a single access.

1: Last Default Master

At the end of current slave access, if no other master request is pending, the slave stay connected with the last master having accessed it.

This results in not having the one cycle latency when the last master re-tries access on 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 which number has been written in the FIXED_DEFMSTR field.

This results in not having the one cycle latency when the fixed master re-tries access on 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.

- **ARBT: Arbitration Type**

0: Round-Robin Arbitration

1: Fixed Priority Arbitration

2: Reserved

3: Reserved

21.8.3 Bus Matrix Priority Registers For Slaves

Name: MATRIX_PRAS0..MATRIX_PRAS8

Address: 0x400E0480 [0], 0x400E0488 [1], 0x400E0490 [2], 0x400E0498 [3], 0x400E04A0 [4], 0x400E04A8 [5], 0x400E04B0 [6], 0x400E04B8 [7], 0x400E04C0 [8]

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

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 WPEN 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.

21.8.4 Bus Matrix Master Remap Control Register

Name: MATRIX_MRCR

Address: 0x400E0500

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
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	RCB5	RCB4	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 AHB Master x**

0: Disable remapped address decoding for the selected Master

1: Enable remapped address decoding for the selected Master

21.8.5 System I/O Configuration Register

Name: CCFG_SYSIO

Address: 0x400E0514

Access: Read-write

Reset: 0x0000_1000

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	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **SYSIO12: PC0 or ERASE Assignment**

1: ERASE function selected (Default at reset).

0: PC0 function selected.

When ERASE function is selected, pull down of PC0 is enabled, and pull up of PC0 is disabled.

When PC0 function is selected, pull down of PC0 is disabled and pull up of PC0 is configurable through PIO block.

21.8.6 Write Protect Mode Register

Name: MATRIX_WPMR

Address: 0x400E05E4

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, refer to Section 21.7 “Write Protect Registers” on page 327.

- **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 MATRIX address space from address offset 0x000 to 0x1FC.

- **WPKEY: Write Protect KEY** (Write-only)

Should be written at value 0x4D4154 (“MAT” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

21.8.7 Write Protect Status Register

Name: MATRIX_WPSR

Address: 0x400E05E8

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, refer to Section 21.7 “Write Protect Registers” on page 327.

- **WPVS: Write Protect Violation Status**

0: No Write Protect Violation has occurred since the last write of MATRIX_WPMR.

1: At least one Write Protect Violation has occurred since the last write of MATRIX_WPMR.

- **WPVSRC: Write Protect Violation Source**

Should be written at value 0x4D4154 (“MAT” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

The protected registers are:

- “Bus Matrix Master Configuration Registers”
- “Bus Matrix Slave Configuration Registers”
- “Bus Matrix Priority Registers For Slaves”
- “Bus Matrix Master Remap Control Register”
- “System I/O Configuration Register”

22. AHB DMA Controller (DMAC)

22.1 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.

The DMAC embeds 6 channels:

DMAC Channel Number	FIFO Size
0	8
1	8
2	8
3	32
4	8
5	32

22.2 Embedded Characteristics

- Programmable Arbitration Policy, Modified Round Robin and Fixed Priority are Available
- Acting as one Matrix Master
- Embeds 4 (SAM3A and 100-pin SAM3X) or 6 (144-pin SAM3X and 217-pin SAM3X8H[1]) channels

Table 22-1. DMA Channels

DMA Channel Size	SAM3A 100-pin SAM3X	144-pin SAM3X 217-pin SAM3X8H[1]
8 bytes FIFO for Channel Buffering	3 (Channels 0, 1 and 2)	4 (Channels 0, 1, 2 and 4)
32 bytes FIFO for Channel Buffering	1 (Channel 3)	2 (Channels 3 and 5)

Note: 1.This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

- Linked List support with Status Write Back operation at End of Transfer
- Word, HalfWord, Byte transfer support.
- Handles high speed transfer of SPI0-1, SSC and HSMCI (peripheral to memory, memory to peripheral)
- Memory to memory transfer

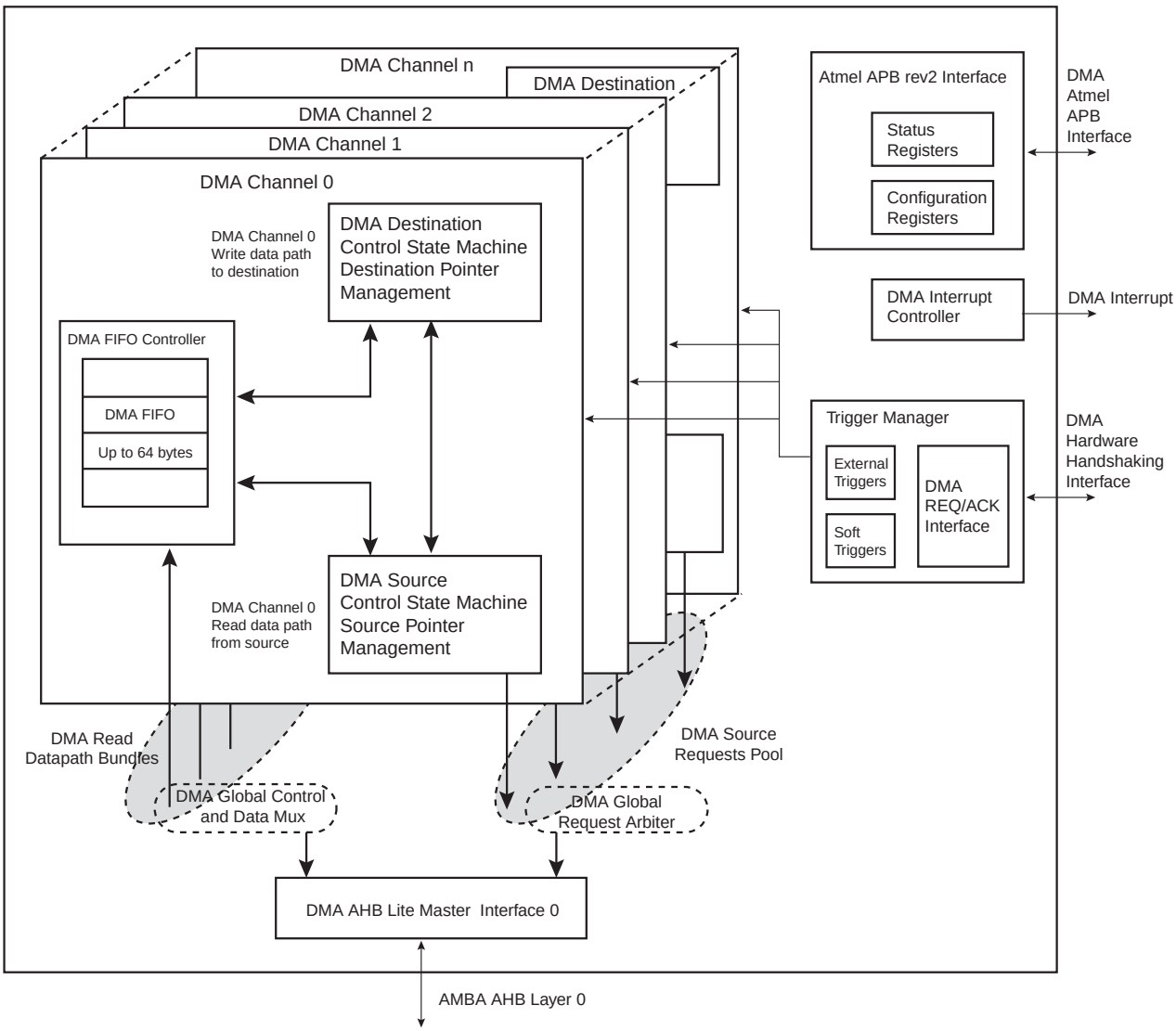
The DMA controller can handle the transfer between peripherals and memory and so receives the triggers from the peripherals below. The hardware interface numbers are also given in Table 22-2.

Table 22-2. DMA Controller

Instance Name	Channel T/R	DMA Channel HW Interface Number
HSMCI	Transmit/Receive	0
SPI0	Transmit	1
SPI0	Receive	2
SSC	Transmit	3
SSC	Receive	4
SPI1	Transmit	5
SPI1	Receive	6
TWI0	Transmit	7
TWI0	Receive	8
-	-	-
-	-	-
USART0	Transmit	11
USART0	Receive	12
USART1	Transmit	13
USART1	Receive	14
PWM	Transmit	15

22.3 Block Diagram

Figure 22-1. DMA Controller (DMAC) Block Diagram



22.4 Functional Description

22.4.1 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.

Flow controller: The device (either the DMAC or source/destination peripheral) that determines the length of and terminates a DMAC buffer transfer. If the length of a buffer is known before enabling the channel, then the DMAC should be programmed as the flow controller. If the length of a buffer is not known prior to enabling the channel, the source or destination peripheral needs to terminate a buffer transfer. In this mode, the peripheral is the flow controller.

Transfer hierarchy: Figure 22-2 on page 342 illustrates the hierarchy between DMAC transfers, buffer transfers, chunk or single, and AMBA transfers (single or burst) for non-memory peripherals. Figure 22-3 on page 342 shows the transfer hierarchy for memory.

Figure 22-2. DMAC Transfer Hierarchy for Non-Memory Peripheral

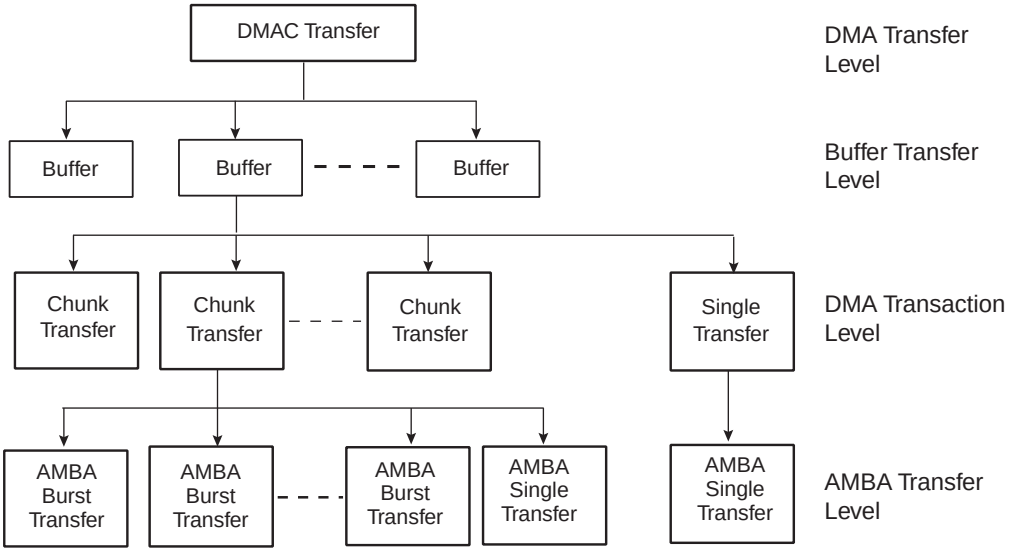
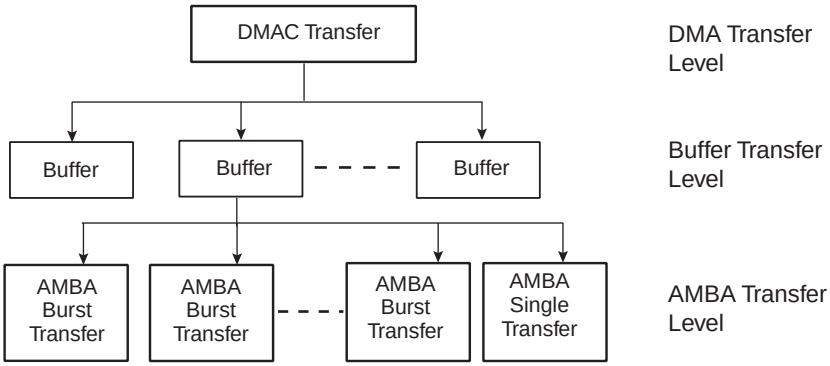


Figure 22-3. DMAC Transfer Hierarchy for Memory



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.

22.4.2 Memory Peripherals

Figure 22-3 on page 342 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.

22.4.3 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.

22.4.3.1 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[2*x*] 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.

22.4.4 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_CTRLBx 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_CTRLBx.These registers, along with the DMAC_CFGx register, are used by the DMAC to set up and describe the buffer transfer.

22.4.4.1 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_CTRLBx).

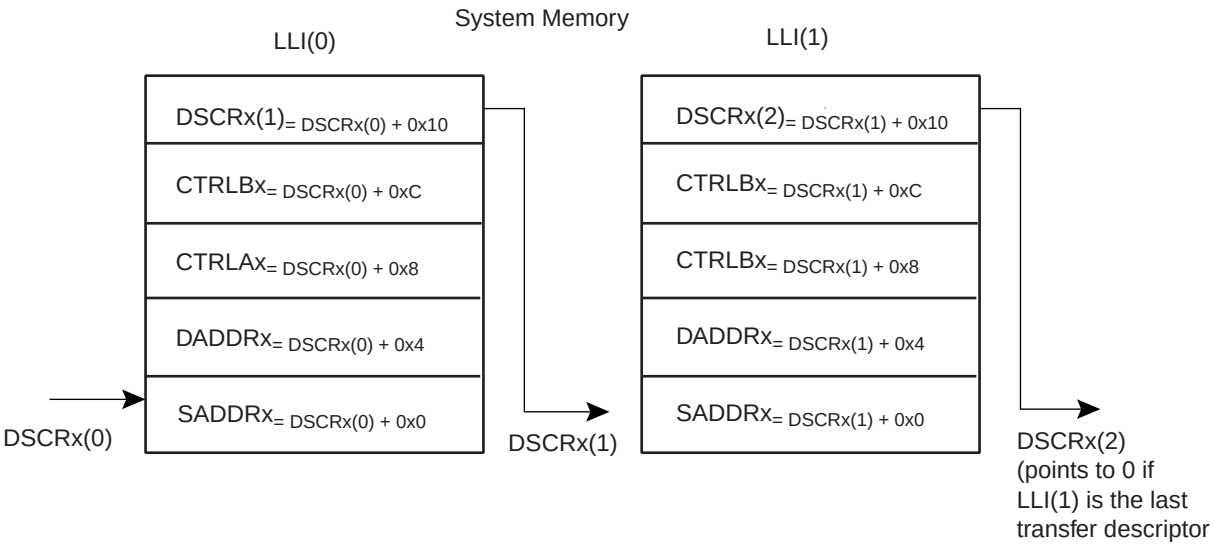
To set up buffer chaining, a sequence of linked lists must be programmed in memory.

The DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx and DMAC_CTRLBx 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 22-4 on page 345 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 22-4. Multi Buffer Transfer Using Linked List



22.4.4.2 Programming DMAC for Multiple Buffer Transfers

Table 22-4. 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.

22.4.4.3 Ending Multi-buffer Transfers

All multi-buffer transfers must end as shown in Row 1 of Table 22-4 on page 345. 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_CRTLBx.AUTO cleared), the user must set up the last buffer descriptor in memory so that LLI.DMAC_CTRLBx.SRC_DSCR is set to 0.

22.4.5 Programming a Channel

Four registers, the DMAC_DSCRx, the DMAC_CTRLAx, the DMAC_CTRLBx and DMAC_CFGx, 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 22-4 on page 345.

The “BTSIZE, SADDR and DADDR” columns indicate where the values of DMAC_SARx, DMAC_DARx, DMAC_CTLx, and DMAC_LLPx are obtained for the next buffer transfer when multi-buffer DMAC transfers are enabled.

22.4.5.1 Programming Examples

Single-buffer Transfer (Row 1)

1. Read the Channel Handler Status Register DMAC_CHSR.ENAx 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:
   a. Write the starting source address in the DMAC_SADDRx register for channel x.
   b. Write the starting destination address in the DMAC_DADDRx register for channel x.
   c. Write the next descriptor address in the DMA_DSCRx register for channel x with 0x0..
   d. Program DMAC_CTRLAx, DMAC_CTRLBx and DMAC_CFGx according to Row 1 as shown in Table 22-4 on page 345.
   e. Write the control information for the DMAC transfer in the DMAC_CTRLAx and DMAC_CTRLBx 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_CTRLBx 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.
   f. Write the channel configuration information into the DMAC_CFGx 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.ENAx 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.
6. 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)

1. Read the Channel Handler Status register to choose a free (disabled) channel.
2. 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 22-5 on page 348) for channel x. For example, in the register, you can program the following:
   a. 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.
   b. 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 channel configuration information into the DMAC_CFGx register for channel x.
   a. 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.
   b. 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.
4. 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 22-4 on page 345. The LLI.DMAC_CTRLBx register of the last Linked List Item must be set as described in Row 1 of Table 22-4. Figure 22-4 on page 345 shows a Linked List example with two list items.
5. 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.
6. 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.
7. Make sure that the LLI.DMAC_CTRLAx.DONE field of the LLI.DMAC_CTRLAx register locations of all LLI entries in memory are cleared.
8. Clear any pending interrupts on the channel from the previous DMAC transfer by reading the status register: DMAC_EBCISR.
9. Program the DMAC_CTRLBx, DMAC_CFGx registers according to Row 4 as shown in Table 22-4 on page 345.
10. Program the DMAC_DSCRx register with DMAC_DSCRx(0), the pointer to the first Linked List item.

11. Finally, enable the channel by writing a ‘1’ to the DMAC_CHER.ENAx bit, where x is the channel number. The transfer is performed.
12. The DMAC fetches the first LLI from the location pointed to by DMAC_DSCRx(0).

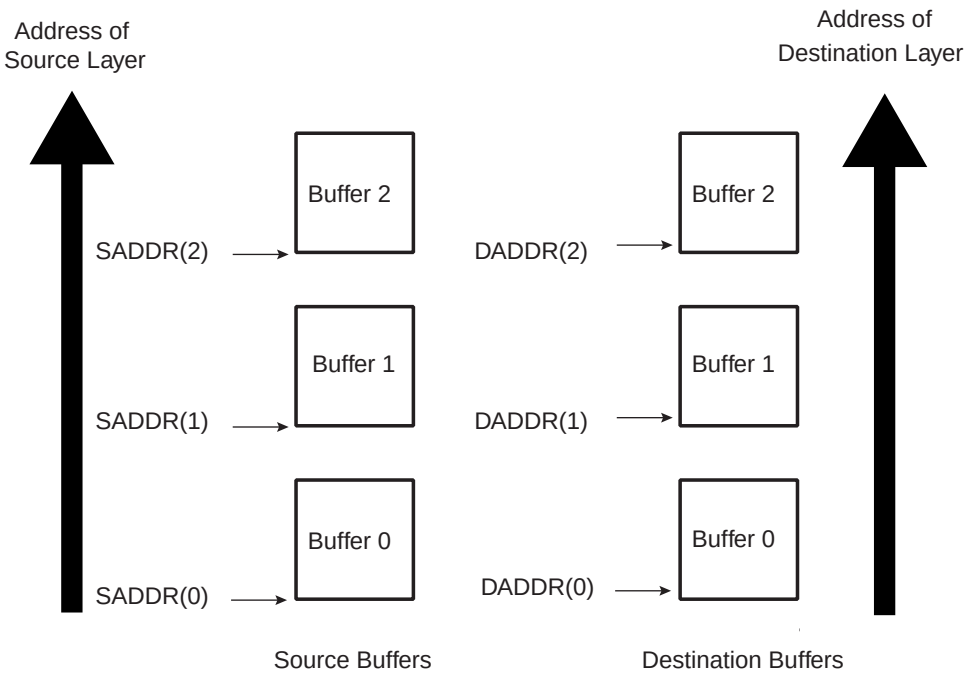
Note: The LLI.DMAC_SADDRx, LLI. DMAC_DADDRx, LLI.DMAC_DSCRx, LLI.DMAC_CTRLAx and LLI.DMAC_CTRLBx registers are fetched. The DMAC automatically reprograms the DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLBx and DMAC_CTRLAx channel registers from the DMAC_DSCRx(0).

13. 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.

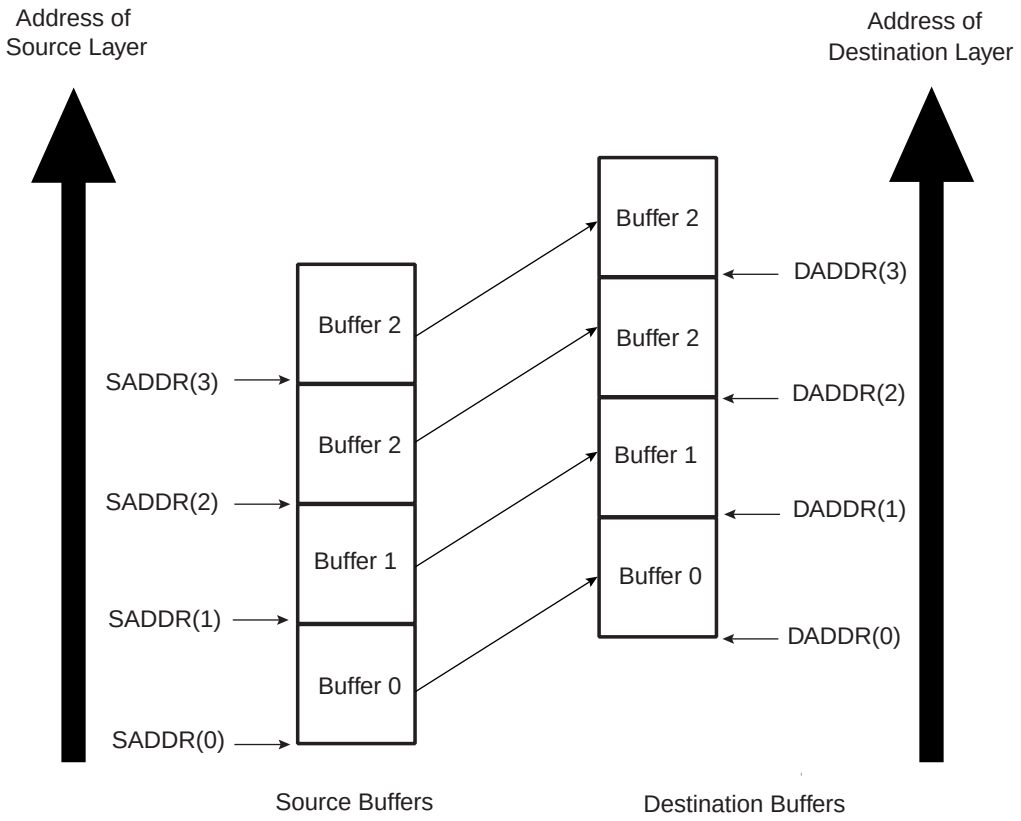
15. The DMAC does not wait for the buffer interrupt to be cleared, but continues fetching the next LLI from the memory location pointed to by current DMAC_DSCRx register and automatically reprograms the DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx and DMAC_CTRLBx channel registers. The DMAC transfer 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 22-4 on page 345. 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 22-5 on page 348.

Figure 22-5. Multi-buffer with Linked List Address for Source and Destination



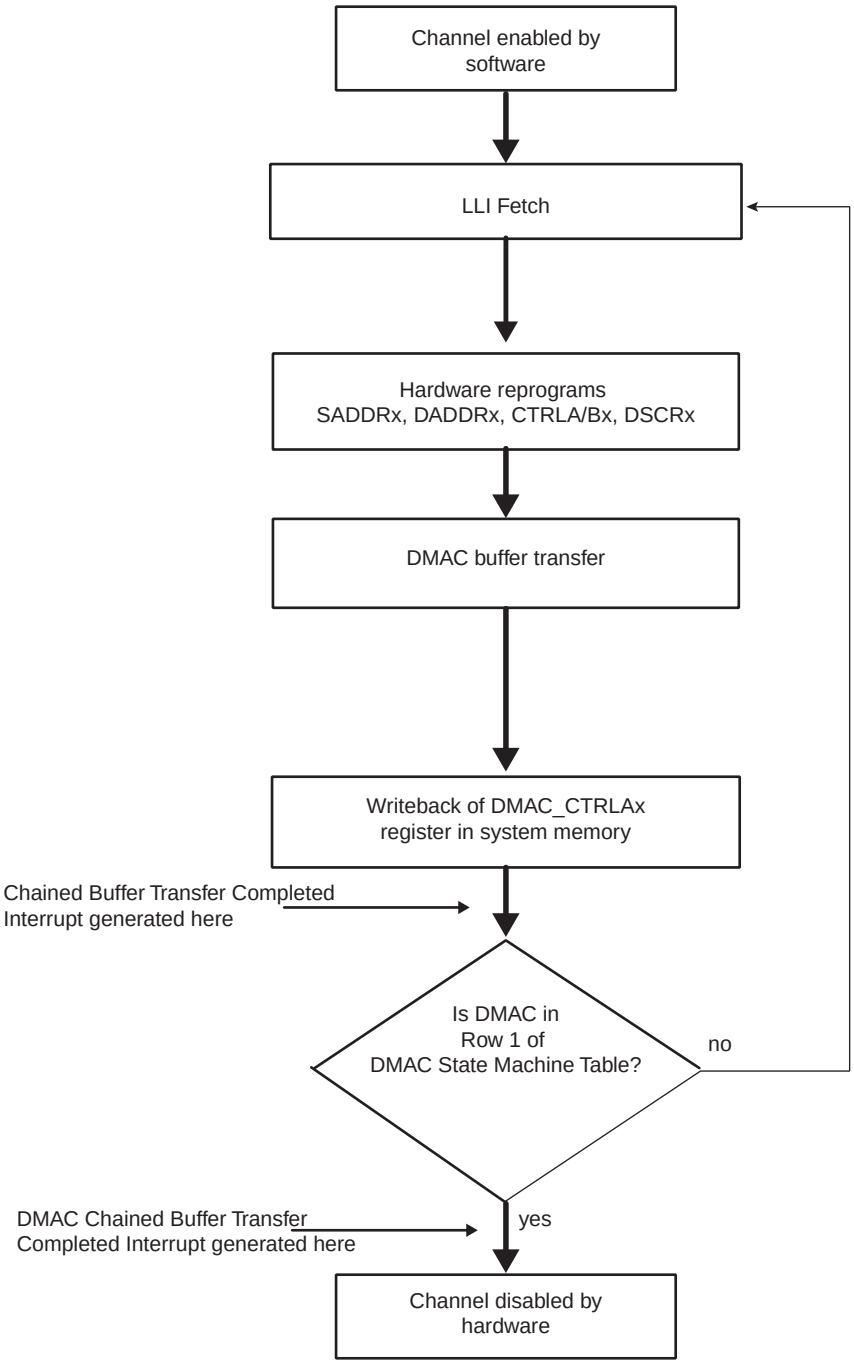
If the user needs to execute a DMAC transfer where the source and destination address are contiguous but the amount of data to be transferred is greater than the maximum buffer size DMAC_CTRLAx.BTSIZE, then this can be achieved using the type of multi-buffer transfer as shown in Figure 22-6 on page 349.

Figure 22-6. Multi-buffer with Linked Address for Source and Destination Buffers are Contiguous



The DMAC transfer flow is shown in Figure 22-7 on page 350.

Figure 22-7. DMAC Transfer Flow for Source and Destination Linked List Address



- 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_CTRLBx register.
- 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 for the specific channel. 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 handshaking interface to the source and destination peripheral. This requires programming the SRC_PER and DST_PER bits, respectively.

16. Source and destination request single and chunk DMAC transactions to transfer the buffer of data (assuming non-memory peripherals). The DMAC acknowledges on completion of each chunk/single transaction and carries out the buffer transfer.
    a. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control peripheral by programming the FC of the DMAC_CTRLBx register.
    a. 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 source/destination requests.
    b. If the hardware handshaking interface is activated for the source or destination peripheral, assign handshaking interface to the source and destination peripheral. This requires programming the SRC_PER and DST_PER bits, respectively.
17. Source and destination request single and chunk DMAC transactions to transfer the buffer of data (assuming non-memory peripherals). DMAC acknowledges at the completion of every transaction (chunk and single) in the buffer and carries out the buffer transfer.
    - 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_CTRLBx register.
    - 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 for the specific channel. 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 the handshaking interface to the source and destination peripheral. This requires programming the SRC_PER and DST_PER bits, respectively.
18. 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.

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:
    a. 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.
    b. 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.

a. 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.

b. 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 22-4 on page 345, while the LLI.DMAC_CTRLBx register of the last Linked List item must be set as described in Row 1 of Table 22-4. Figure 22-4 on page 345 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 22-4 on page 345
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.

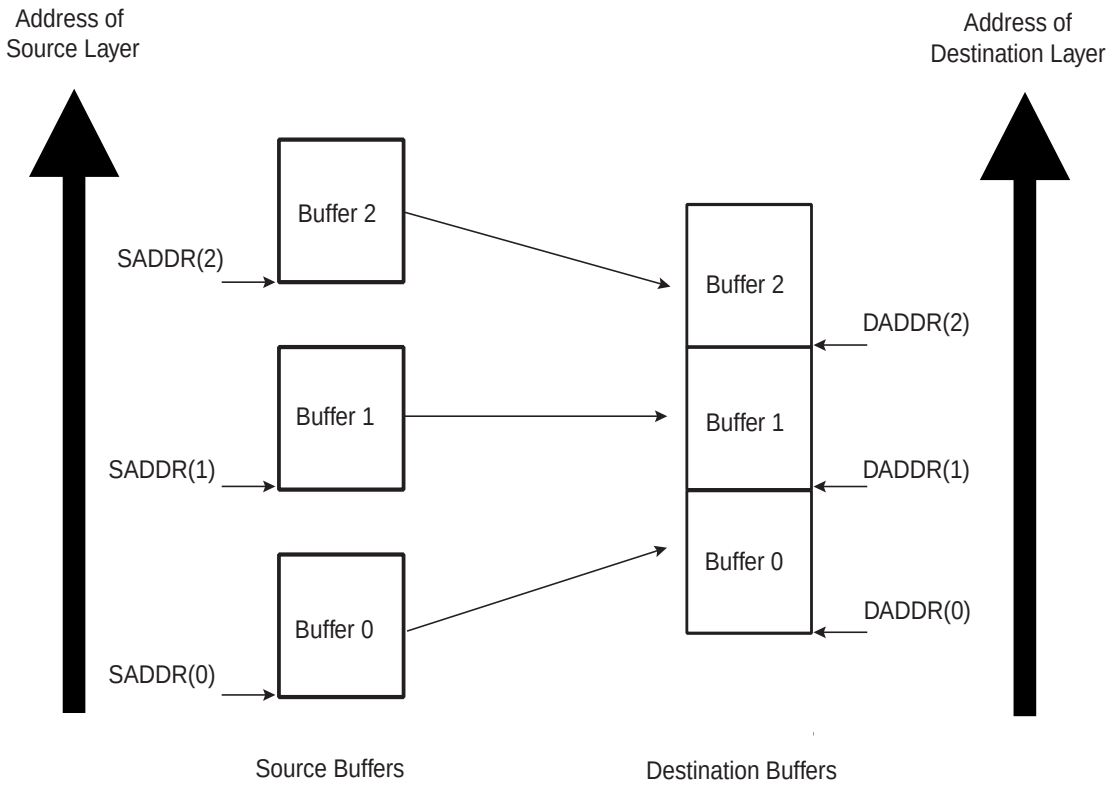
14. Source and destination requests single and chunk DMAC transactions to transfer the buffer of data (assuming non-memory peripherals). The DMAC acknowledges at the completion of every transaction (chunk and single) in the buffer and carries out the buffer transfer.
15. Once the buffer of data is transferred, 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 22-4 on page 345. The DMAC then knows that the previous buffer transferred was the last buffer in the DMAC transfer.

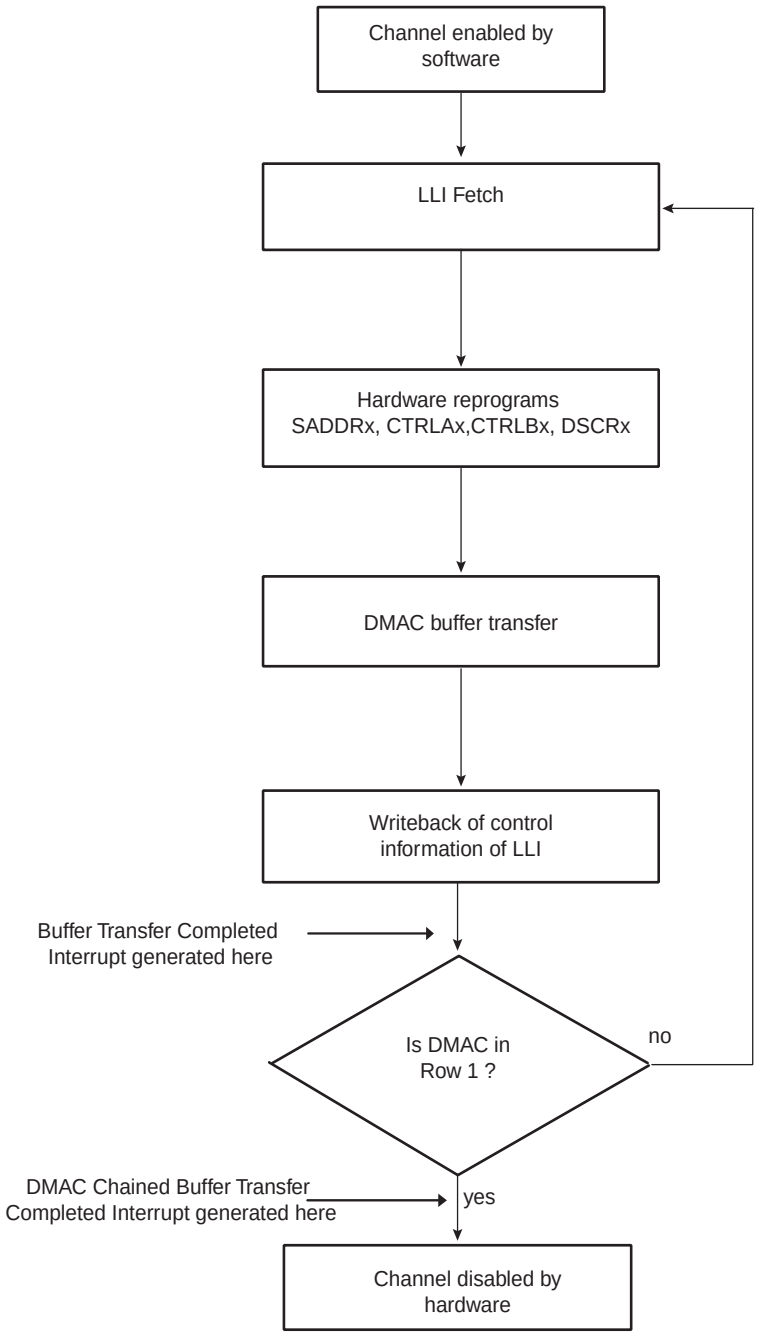
The DMAC transfer might look like that shown in Figure 22-8 on page 353. Note that the destination address is decrementing.

Figure 22-8. DMAC Transfer with Linked List Source Address and Contiguous Destination Address



The DMAC transfer flow is shown in Figure 22-9 on page 354.

Figure 22-9. DMAC Transfer Flow for Linked List Source Address and Contiguous Destination Address



22.4.6 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.

22.4.6.1 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.

22.5 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.
- When the destination peripheral has been defined as the flow controller, source single transfer requests are not serviced until the destination peripheral has asserted its Last Transfer Flag.
- When the source peripheral has been defined as the flow controller, destination single transfer requests are not serviced until the source peripheral has asserted its Last Transfer Flag.
- When the destination peripheral has been defined as the flow controller, if the destination width is smaller than the source width, then a data loss may occur, and the loss is equal to the Source Single Transfer size in bytes- destination Single Transfer size in bytes.
- When a Memory to Peripheral transfer occurs, if the destination peripheral has been defined as the flow controller, then a prefetch operation is performed. It means that data is extracted from the memory before any request from the peripheral is generated.
- 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.

22.6 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 WPVSRC 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 359
- "DMAC Enable Register" on page 360
- "DMAC Channel x [x = 0..5] Source Address Register" on page 371
- "DMAC Channel x [x = 0..5] Destination Address Register" on page 372
- "DMAC Channel x [x = 0..5] Descriptor Address Register" on page 373
- "DMAC Channel x [x = 0..5] Control A Register" on page 374
- "DMAC Channel x [x = 0..5] Control B Register" on page 376
- "DMAC Channel x [x = 0..5] Configuration Register" on page 378

22.7 AHB DMA Controller (DMAC) User Interface

Table 22-5. 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	–	–	–

22.7.1 DMAC Global Configuration Register

Name: DMAC_GCFG

Address: 0x400C4000

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.

22.7.2 DMAC Enable Register

Name: DMAC_EN

Address: 0x400C4004

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**

0: DMA Controller is disabled.

1: DMA Controller is enabled.

22.7.3 DMAC Software Single Request Register

Name: DMAC_SREQ

Address: 0x400C4008

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
–	–	–	–	DSREQ5	SSREQ5	DSREQ4	SSREQ4
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.

22.7.4 DMAC Software Chunk Transfer Request Register

Name: DMAC_CREQ

Address: 0x400C400C

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
–	–	–	–	DCREQ5	SCREQ5	DCREQ4	SCREQ4
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.

22.7.5 DMAC Software Last Transfer Flag Register

Name: DMAC_LAST

Address: 0x400C4010

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
–	–	–	–	DLAST5	SLAST5	DLAST4	SLAST4
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.

22.7.6 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Enable Register

Name: DMAC_EBCIER

Address: 0x400C4018

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	ERR5	ERR4	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
–	–	CBTC5	CBTC4	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
–	–	BTC5	BTC4	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [5:0]**

Buffer Transfer Completed Interrupt Enable Register. Set the relevant bit in the BTC field to enable the interrupt for channel i.

- **CBTCx: Chained Buffer Transfer Completed [5:0]**

Chained Buffer Transfer Completed Interrupt Enable Register. Set the relevant bit in the CBTC field to enable the interrupt for channel i.

- **ERRx: Access Error [5:0]**

Access Error Interrupt Enable Register. Set the relevant bit in the ERR field to enable the interrupt for channel i.

22.7.7 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Disable Register

Name: DMAC_EBCIDR

Address: 0x400C401C

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	ERR5	ERR4	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
–	–	CBTC5	CBTC4	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
–	–	BTC5	BTC4	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [5:0]**

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 [5:0]**

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 [5:0]**

Access Error Interrupt Disable Register. When set, a bit of the ERR field disables the interrupt from the relevant DMAC channel.

22.7.8 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Mask Register

Name: DMAC_EBCIMR

Address: 0x400C4020

Access: Read-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	ERR5	ERR4	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
–	–	CBTC5	CBTC4	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
–	–	BTC5	BTC4	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [5:0]**

0: Buffer Transfer Completed Interrupt is disabled for channel i.

1: Buffer Transfer Completed Interrupt is enabled for channel i.

- **CBTCx: Chained Buffer Transfer Completed [5:0]**

0: Chained Buffer Transfer interrupt is disabled for channel i.

1: Chained Buffer Transfer interrupt is enabled for channel i.

- **ERRx: Access Error [5:0]**

0: Transfer Error Interrupt is disabled for channel i.

1: Transfer Error Interrupt is enabled for channel i.

22.7.9 DMAC Error, Buffer Transfer and Chained Buffer Transfer Status Register

Name: DMAC_EBCISR

Address: 0x400C4024

Access: Read-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	ERR5	ERR4	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
–	–	CBTC5	CBTC4	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
–	–	BTC5	BTC4	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [5:0]**

When BTC[*i*] is set, Channel *i* buffer transfer has terminated.

- **CBTCx: Chained Buffer Transfer Completed [5:0]**

When CBTC[*i*] is set, Channel *i* Chained buffer has terminated. LLI Fetch operation is disabled.

- **ERRx: Access Error [5:0]**

When ERR[*i*] is set, Channel *i* has detected an AHB Read or Write Error Access.

22.7.10 DMAC Channel Handler Enable Register

Name: DMAC_CHER

Address: 0x400C4028

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	KEEP5	KEEP4	KEEP3	KEEP2	KEEP1	KEEP0
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	SUSP5	SUSP4	SUSP3	SUSP2	SUSP1	SUSP0
7	6	5	4	3	2	1	0
–	–	ENA5	ENA4	ENA3	ENA2	ENA1	ENA0

- **ENAx: Enable [5:0]**

When set, a bit of the ENA field enables the relevant channel.

- **SUSPx: Suspend [5:0]**

When set, a bit of the SUSP field freezes the relevant channel and its current context.

- **KEEPx: Keep on [5:0]**

When set, a bit of the KEEP field resumes the current channel from an automatic stall state.

22.7.11 DMAC Channel Handler Disable Register

Name: DMAC_CHDR

Address: 0x400C402C

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
–	–	RES5	RES4	RES3	RES2	RES1	RES0
7	6	5	4	3	2	1	0
–	–	DIS5	DIS4	DIS3	DIS2	DIS1	DIS0

- **DISx: Disable [5:0]**

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[5:0] field in the DMAC_CHSR register to be sure that the channel is disabled.

- **RESx: Resume [5:0]**

Write one to this field to resume the channel transfer restoring its context.

22.7.12 DMAC Channel Handler Status Register

Name: DMAC_CHSR

Address: 0x400C4030

Access: Read-only

Reset: 0x00FF0000

31	30	29	28	27	26	25	24
–	–	STAL5	STAL4	STAL3	STAL2	STAL1	STAL0
23	22	21	20	19	18	17	16
–	–	EMPT5	EMPT4	EMPT3	EMPT2	EMPT1	EMPT0
15	14	13	12	11	10	9	8
–	–	SUSP5	SUSP4	SUSP3	SUSP2	SUSP1	SUSP0
7	6	5	4	3	2	1	0
–	–	ENA5	ENA4	ENA3	ENA2	ENA1	ENA0

- **ENAx: Enable [5:0]**

A one in any position of this field indicates that the relevant channel is enabled.

- **SUSPx: Suspend [5:0]**

A one in any position of this field indicates that the channel transfer is suspended.

- **EMPTx: Empty [5:0]**

A one in any position of this field indicates that the relevant channel is empty.

- **STALx: Stalled [5:0]**

A one in any position of this field indicates that the relevant channel is stalling.

22.7.13 DMAC Channel x [x = 0..5] Source Address Register

Name: DMAC_SADDRx [x = 0..5]

Address: 0x400C403C [0], 0x400C4064 [1], 0x400C408C [2], 0x400C40B4 [3], 0x400C40DC [4], 0x400C4104 [5]

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
SADDR							

23	22	21	20	19	18	17	16
SADDR							

15	14	13	12	11	10	9	8
SADDR							

7	6	5	4	3	2	1	0
SADDR							

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.

22.7.14 DMAC Channel x [x = 0..5] Destination Address Register

Name: DMAC_DADDRx [x = 0..5]

Address: 0x400C4040 [0], 0x400C4068 [1], 0x400C4090 [2], 0x400C40B8 [3], 0x400C40E0 [4], 0x400C4108 [5]

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
DADDR							

23	22	21	20	19	18	17	16
DADDR							

15	14	13	12	11	10	9	8
DADDR							

7	6	5	4	3	2	1	0
DADDR							

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.

22.7.15 DMAC Channel x [x = 0..5] Descriptor Address Register

Name: DMAC_DSCRx [x = 0..5]

Address: 0x400C4044 [0], 0x400C406C [1], 0x400C4094 [2], 0x400C40BC [3], 0x400C40E4 [4], 0x400C410C [5]

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
DSCR							

23	22	21	20	19	18	17	16
DSCR							

15	14	13	12	11	10	9	8
DSCR							

7	6	5	4	3	2	1	0
DSCR						–	

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.

22.7.16 DMAC Channel x [x = 0..5] Control A Register

Name: DMAC_CTRLAx [x = 0..5]

Address: 0x400C4048 [0], 0x400C4070 [1], 0x400C4098 [2], 0x400C40C0 [3], 0x400C40E8 [4], 0x400C4110 [5]

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
–	DCSIZE			–	SCSIZE		

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 380

- **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.

- **SCSIZE: Source Chunk Transfer Size.**

Value	Name	Description
000	CHK_1	1 data transferred
001	CHK_4	4 data transferred
010	CHK_8	8 data transferred
011	CHK_16	16 data transferred
100	CHK_32	32 data transferred
101	CHK_64	64 data transferred
110	CHK_128	128 data transferred
111	CHK_256	256 data transferred

- **DCSIZE: Destination Chunk Transfer Size**

Value	Name	Description
000	CHK_1	1 data transferred
001	CHK_4	4 data transferred
010	CHK_8	8 data transferred
011	CHK_16	16 data transferred
100	CHK_32	32 data transferred
101	CHK_64	64 data transferred
110	CHK_128	128 data transferred
111	CHK_256	256 data transferred

- **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**

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 transfer.

22.7.17 DMAC Channel x [x = 0..5] Control B Register

Name: DMAC_CTRLBx [x = 0..5]

Address: 0x400C404C [0], 0x400C4074 [1], 0x400C409C [2], 0x400C40C4 [3], 0x400C40EC [4], 0x400C4114 [5]

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
000	MEM2MEM_DMA_FC	Memory-to-Memory Transfer DMAC is flow controller
001	MEM2PER_DMA_FC	Memory-to-Peripheral Transfer DMAC is flow controller
010	PER2MEM_DMA_FC	Peripheral-to-Memory Transfer DMAC is flow controller
011	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**

If this bit is cleared, when the buffer transfer is completed, the BTCx flag is set in the EBCISR status register. This bit is active low.

22.7.18 DMAC Channel x [x = 0..5] Configuration Register

Name: DMAC_CFGx [x = 0..5]

Address: 0x400C4050 [0], 0x400C4078 [1], 0x400C40A0 [2], 0x400C40C8 [3], 0x400C40F0 [4], 0x400C4118 [5]

Access: Read-write

Reset: 0x0100000000

31	30	29	28	27	26	25	24
–	–	FIFOCFG		–	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 380

- **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

- **FIFOCFG: 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.

22.7.19 DMAC Write Protect Mode Register

Name: DMAC_WPMR

Address: 0x400C41E4

Access: Read-write

Reset: See Table 22-5

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 0x444D4143 (“DMAC” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x444D4143 (“DMAC” in ASCII).

Protects the registers:

- “DMAC Global Configuration Register” on page 359
- “DMAC Enable Register” on page 360
- “DMAC Channel x [x = 0..5] Source Address Register” on page 371
- “DMAC Channel x [x = 0..5] Destination Address Register” on page 372
- “DMAC Channel x [x = 0..5] Descriptor Address Register” on page 373
- “DMAC Channel x [x = 0..5] Control A Register” on page 374
- “DMAC Channel x [x = 0..5] Control B Register” on page 376
- “DMAC Channel x [x = 0..5] Configuration Register” on page 378

- **WPKEY: Write Protect KEY**

Should be written at value 0x50494F (“DMAC” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

22.7.20 DMAC Write Protect Status Register

Name: DMAC_WPSR

Address: 0x400C41E8

Access: Read-only

Reset: See Table 22-5

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 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.

23. External Memory Bus

23.1 Description

The external memory bus allows external devices connection to the various embedded memory controllers of the microcontroller. The external memory bus handles control, address and data bus of each embedded memory controllers. The SAM3X embeds a Static Memory/NAND Flash/ECC Controller (SMC/NFC/ECC) and a SDR-SDRAM Controller (SDR-SDRAMC). Furthermore, the external memory bus handles data transfers with up to eight external devices, each assigned to eight address spaces. Data transfers are performed through a 8-bit or 16-bit data bus, an address bus of up to 24 bits, up to eight chip select lines (NCS[7:0]) and several control pins that are generally multiplexed between the different embedded memory controllers.

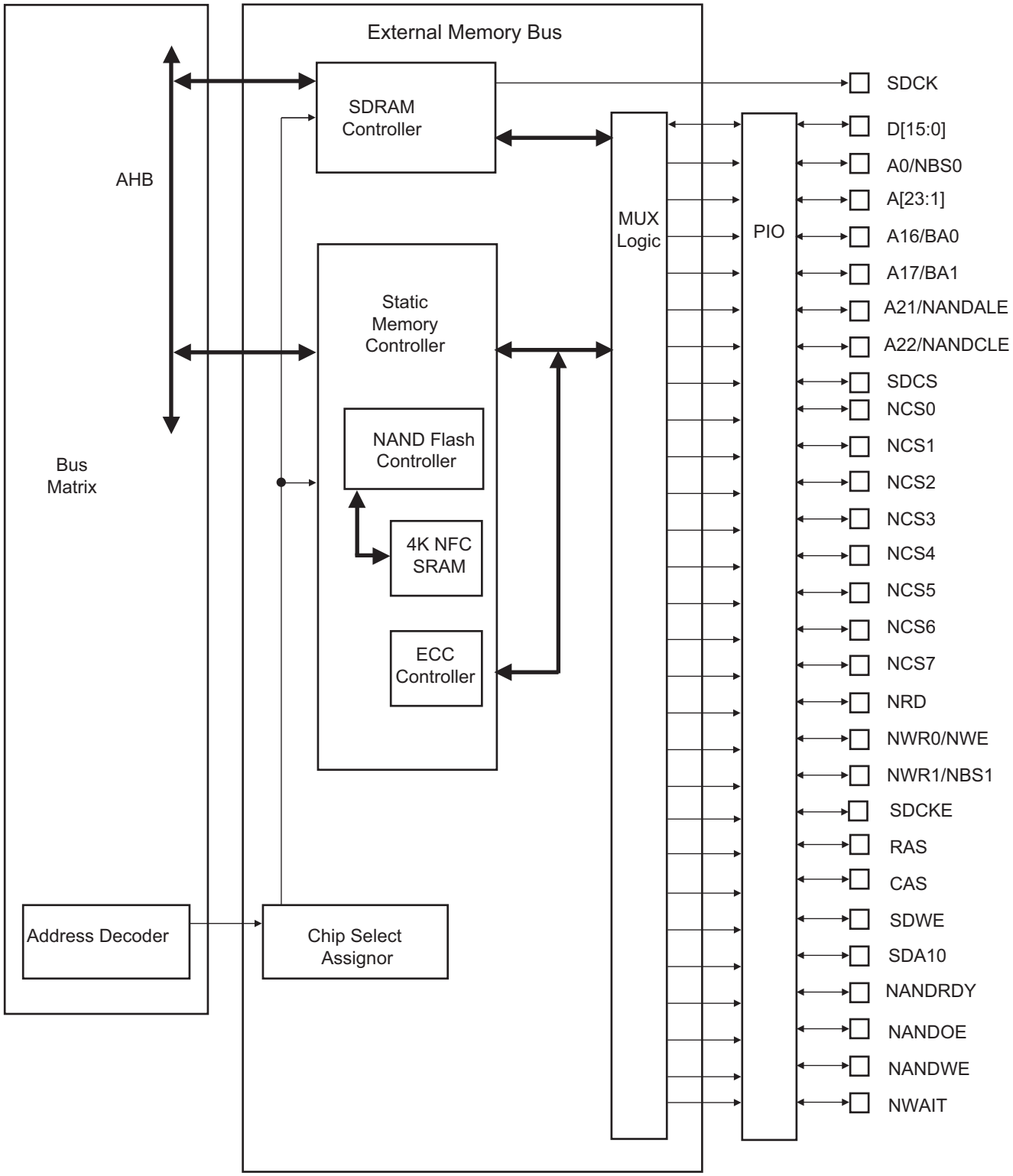
23.2 Embedded Characteristics

- Only present on 144-pin SAM3X and 217-pin SAM3X8H(1) versions
- Managing SMC, NAND Flash and SDR-SDRAM Controller accesses offering:
  - Up to 8 configurable chip selects
  - Programmable timing on a per chip select basis
  - 16 Mbyte Address Space per Chip Select
  - 8- or 16-bit Data Bus
  - Word, Halfword, Byte Transfers
  - Byte Write or Byte Select Lines
  - 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 Wait Request
  - Automatic Switch to Slow Clock Mode
  - Asynchronous Read in Page Mode Supported: Page Size Ranges from 4 to 32 bytes
  - NAND Flash Controller supporting NAND Flash with Multiplexed Data/Address buses
  - Supports SLC NAND Flash technology
  - Supports Hardware Error Correcting Code (ECC), 1-bit error correction, 2-bit error detection
  - Detection and Correction by Software
  - Supports SDRAM with 8- or 16-bit Data Path (217-pin SAM3X8H(1))

Note: 1.This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

23.3 Block Diagram

Figure 23-1. Organization of the External Bus Interface



23.4 I/O Lines Description

Table 23-1. I/O Lines Description

Name	Function	Type	Active Level
External Memory Bus			
D[15:0]	Data Bus	I/O	
A[23:0]	Address Bus	Output	
Static Memory Controller (SMC)			
NCS[7:0]	Chip Select Lines	Output	Low
NWR0/NWE	Write Signals	Output	Low
NRD	Read Signal	Output	Low
NWR1/NBS1	Write Enable/Upper Byte Select	Output	Low
A0/NBS0	Lower Byte Select	Output	Low
NWAIT	External Wait Signal	Input	Low
NAND Flash Controller (NFC)			
NCS[7:0]	Chip Select Lines	Output	Low
NANDOE	NAND Flash Output Enable	Output	Low
NANDWE	NAND Flash Write Enable	Output	Low
NANDCLE	NAND Flash Command Line Enable	Output	Low
NANDALE	NAND Flash Address Line Enable	Output	Low
NANDRDY	NAND Flash Ready/Busy	Input	Low
SDR-SDRAM Controller			
SDCK	SDRAM Clock	Output	
SDCKE	SDRAM Clock Enable	Output	High
SDCS	SDRAM Controller Chip Select Line	Output	Low
BA[1:0]	Bank Select	Output	
SDWE	SDRAM Write Enable	Output	Low
RAS - CAS	Row and Column Signal	Output	Low
NBS[1:0]	Byte Mask Signals	Output	Low
SDA10	SDRAM Address 10 Line	Output	

The connection of some signals through the Mux logic is not direct and depends on the Memory Controller in use at the moment.

Table 23-2 details the connections between the two memory controllers and the bus pins.

Table 23-2. External Memory Bus Pins and Memory Controllers I/O Lines Connections

EBI Pins	SDRAMC I/O Lines	SMC I/O Lines
NWR1/NBS1	NBS1	NWR1/NBS1
A0/NBS0	Not Supported	A0/NBS0
A1	Not Supported	A1
A[11:2]	A[9:0]	A[11:2]
SDA10	A10	Not Supported
A12	Not Supported	A12
A[14:13]	A[12:11]	A[14:13]
A[23:15]	Not Supported	A[23:15]
D[15:0]	D[15:0]	D[15:0]

23.5 Application Example

23.5.1 Hardware Interface

Table 23-3 details the connections to be applied between the External Memory Bus pins and the external devices for each memory controller.

Table 23-3. EBI Pins and External Static Device Connections

EBI Pins	SMC Pins				SDR-SDRAMC Pins
	8-bit Static Device	2 x 8-bit Static Devices	16-bit Static Device	NAND Flash	
D0–D7	D0–D7	D0–D7	D0–D7	I/O0–I/O7	D0–D7
D8–D15	–	D8–D15	D8–D15	I/O8–I/O15[1]	D8–D15
A0/NBS0	A0	–	NLB	–	DQM0
A1	A1	A0	A0	–	–
A2–A9	A2–A9	A1–A8	A1–A8	–	A0–A7
A10	A10	A9	A9	–	A8
A11	A11	A10	A10	–	A9
SDCS	–	–	–	–	CS
SDA10	–	–	–	–	A10
A12	A12	A11	A11	–	–
A13–A14	A13–A14	A12–A13	A12–A13	–	A11–A12
A15	A15	A14	A14	–	–
A16/BA0	A16	A15	A15	–	BA0
A17/BA1	A17	A16	A16	–	BA1
A18–A20	A18–A20	A17–A19	A17–A19	–	–
A21/NANDALE	A21	A20	A20	ALE	–
A22/NANDCLE	A22	A21	A21	CLE	–
A23	A23	A22	A22	–	–

Table 23-3. EBI Pins and External Static Device Connections (Continued)

EBI Pins	SMC Pins				SDR-SDRAMC Pins
	8-bit Static Device	2 x 8-bit Static Devices	16-bit Static Device	NAND Flash	
NCS0	CS	CS	CS	CE[3]	–
NCS1	CS	CS	CS	–	CS
NCS2	CS	CS	CS	CE[3]	–
NCS3	CS	CS	CS	CE[3]	–
NCS4	CS	CS	CS	CE[3]	–
NCS5	CS	CS	CS	CE[3]	–
NCS6	CS	CS	CS	CE[3]	–
NCS7	CS	CS	CS	CE[3]	–
NANDOE	–	–	–	RE	–
NANDWE	–	–	–	WE	–
NRD	OE	OE	OE		–
NWR0/NWE	WE	WE[2]	WE		–
NWR1/NBS1	WE	WE[2]	NUB	–	DQM1
SDCK	–	–	–	–	CLK
SDCKE	–	–	–	–	CKE
RAS	–	–	–	–	RAS
CAS	–	–	–	–	CAS
SDWE	–	–	–	–	WE
NWAIT	–	–	–	–	–
NANDRDY	–	–	–	RDY	–

Notes:
1. I/O8–I/O15 bits used only for 16-bit NAND Flash.
2. NWR1 enables upper byte writes. NWR0 enables lower byte writes.
3. CE connection depends on the NAND Flash.
   For standard NAND Flash devices, it must be connected to any free PIO line.
   For “CE don’t care” 8-bit NAND Flash devices, it can be either connected to any NCS
   For “CE don’t care” 16-bit NAND Flash devices, it must be connected to any free PIO line.

23.6 Product Dependencies

23.6.1 I/O Lines

The pins used for interfacing the External Bus Interface may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the External Bus Interface pins to their peripheral function. If I/O lines of the External Bus Interface are not used by the application, they can be used for other purposes by the PIO Controller.

23.7 Functional Description

The external memory bus transfers data between the internal AHB Bus (handled by the memory controllers) and the external memories or peripheral devices. It controls the waveforms and the parameters of the external address, data and control buses and is composed of the following elements:

- Static Memory Controller (SMC)
- NAND Flash Controller (NFC)
- SDRAM Controller (SDRAMC)
- ECC Controller (ECC)

23.7.1 Bus Multiplexing

The external memory bus offers a complete set of control signals that share the 16-bit data lines, the address lines of up to 24 bits and the control signals through a multiplex logic operating in function of the memory area requests.

Multiplexing is specifically organized in order to guarantee the maintenance of the address and output control lines at a stable state while no external access is being performed. Multiplexing is also designed to respect the data float times defined in the Memory Controllers. Furthermore, refresh cycles of the SDRAM are executed independently by the SDRAM Controller without delaying the other external Memory Controller accesses.

23.7.2 Static Memory Controller

For information on the Static Memory Controller, refer to Section 25. “Static Memory Controller (SMC)”.

23.7.3 NAND Flash Controller

For information on the NAND Flash Controller, refer to Section 25. “Static Memory Controller (SMC)”.

23.7.4 SDRAM Controller

For information on the SDRAM Controller, refer to Section 24. “AHB SDRAM Controller (SDRAMC)”.

23.7.5 ECC Controller

For information on the ECC Controller, refer to Section 25. “Static Memory Controller (SMC)”.

23.8 Implementation Examples

23.8.1 SDR-SDRAM

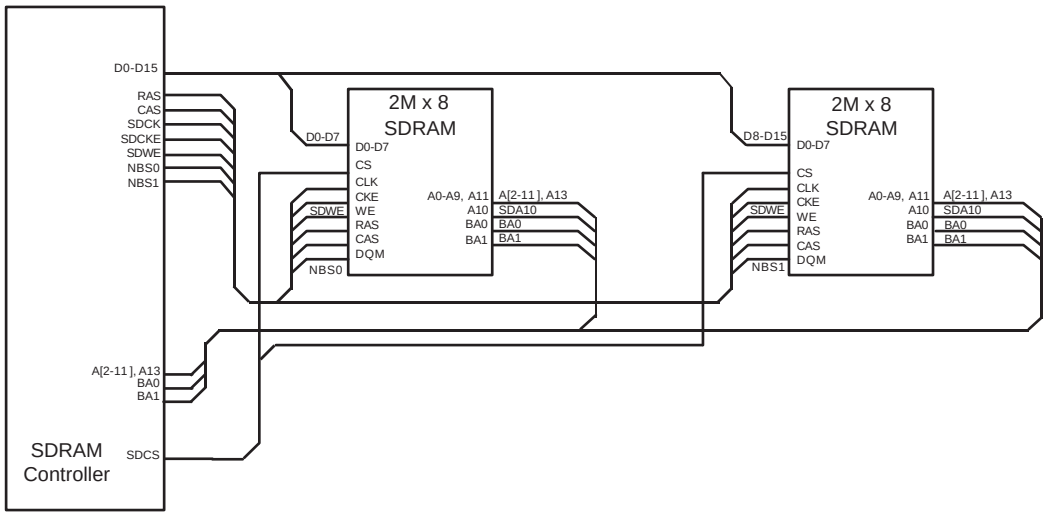
23.8.1.1 Software Configuration (2 x 8-bit SDR-SDRAM)

The following configuration must be respected:

- Address lines A[2..11], A[13], BA0, BA1, SDA10, SDCS, SDWE, SDCKE, NBS1, RAS, CAS, and data lines D[0..15] are multiplexed with PIO lines and thus dedicated PIOs must be programmed in peripheral mode in the PIO controller.
- Initialize the SDRAM Controller depending on the SDRAM device and system bus frequency.

The SDRAM initialization sequence is described in Section 24.5.1 “SDRAM Device Initialization”.

Figure 23-2. 2 x 8-bit SDR-SDRAM Hardware Configuration



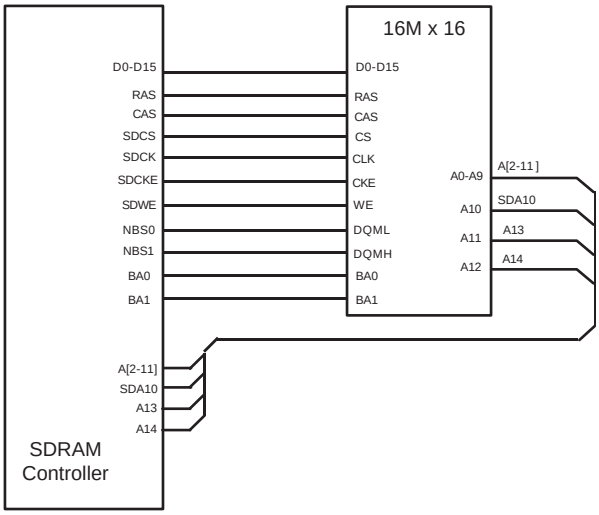
23.8.1.2 Software Configuration (1 x 16-bit SDR-SDRAM)

The following configuration must be respected:

- Address lines A[2..11], A[13-14], BA0, BA1, SDA10, SDCS, SDWE, SDCKE, NBS1, RAS, CAS, and data lines D[0..15] are multiplexed with PIO lines and thus dedicated PIOs must be programmed in peripheral mode in the PIO controller.
- Initialize the SDRAM Controller depending on the SDRAM device and system bus frequency.

The SDRAM initialization sequence is described in Section 24.5.1 “SDRAM Device Initialization”.

Figure 23-3. 1 x 16-bit SDR-SDRAM Hardware Configuration



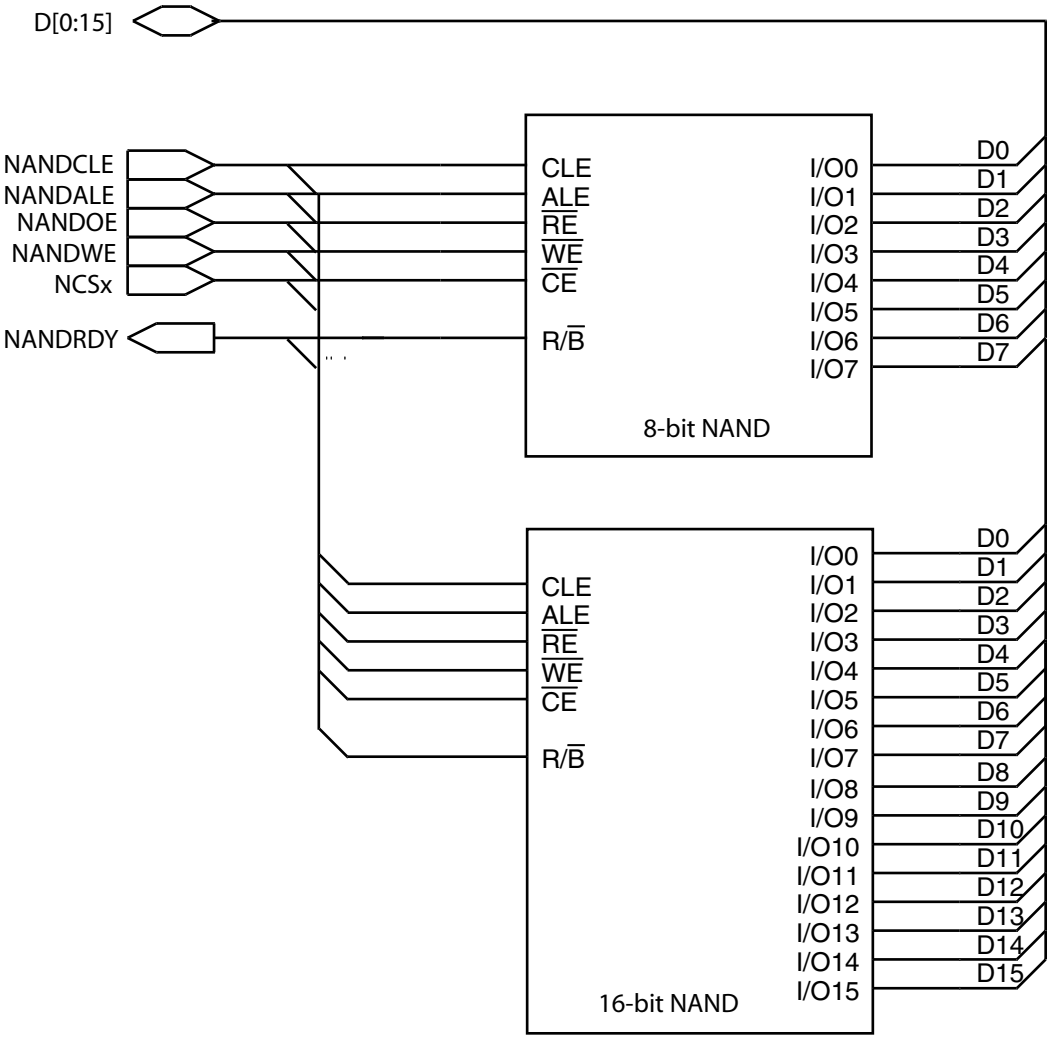
23.8.2 8-bit and 16-bit NAND Flash

23.8.2.1 Software Configuration (8-bit and 16-bit NAND Flash)

The following configuration must be respected:

- NADNCLE, NANDALE, NANDOE, NANDRDY and NANDWE signals are multiplexed with PIO lines and thus the dedicated PIOs must be programmed in peripheral mode in the PIO controller. D[0:7] or D[0:15] must be programmed in peripheral mode in the PIO controller.
- Assign the SMC chip select line CSx to the NAND Flash by setting the CSID field in the NFCADDR_CMD Register.
- Configure Static Memory Controller CSx Setup, Pulse, Cycle and Mode according to NAND Flash timings, the data bus width and the system bus frequency.

Figure 23-4. 8-bit and 16-bit NAND Flash Hardware Configuration



23.8.3 NOR Flash on NCS0

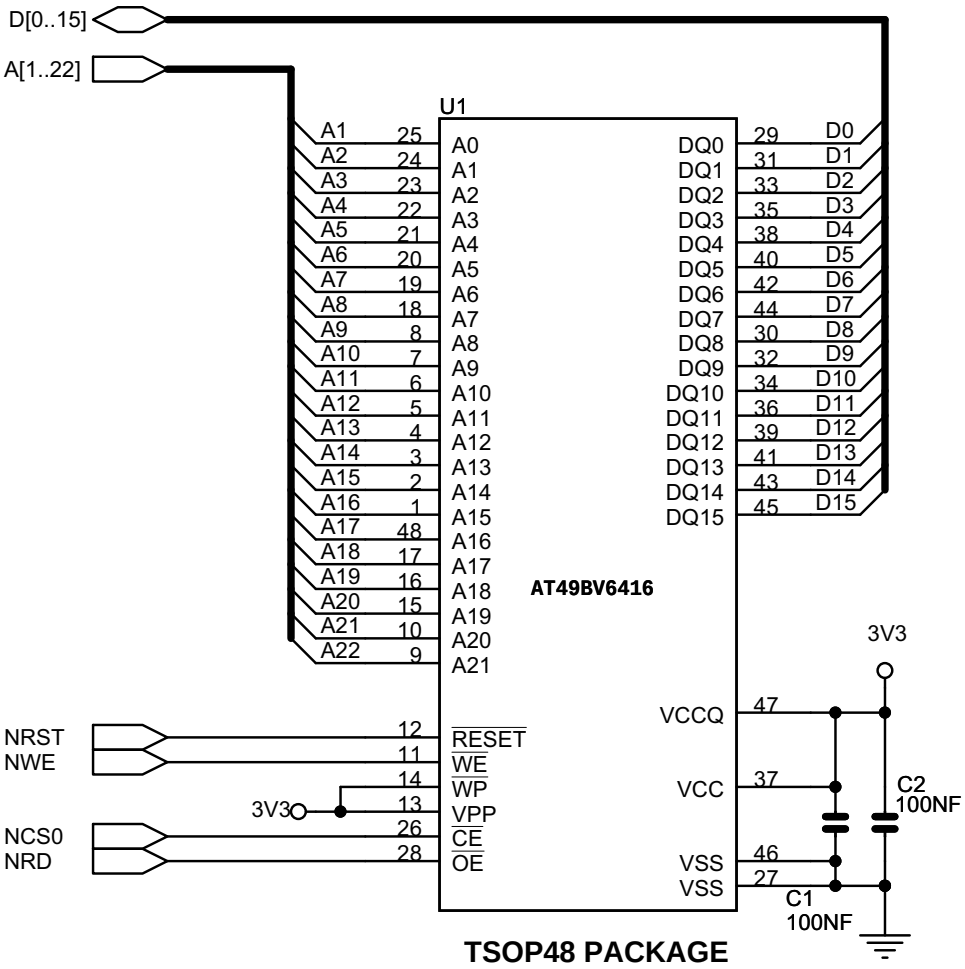
23.8.3.1 Software Configuration (NOR Flash on NCS0)

Address lines A[1..22], NCS0, NRD, NWE and data lines D[0..15] are multiplexed with PIO lines and thus dedicated PIOs must be programmed in peripheral mode in the PIO controller.

The default configuration for the Static Memory Controller, byte select mode, 16-bit data bus, Read/Write controlled by Chip Select, allows access on 16-bit non-volatile memory at slow clock.

For another configuration, configure the Static Memory Controller CS0 Setup, Pulse, Cycle and Mode depending on Flash timings and system bus frequency.

Figure 23-5. NOR Flash on NCS0 Hardware Configuration



24. AHB SDRAM Controller (SDRAMC)

24.1 Description

The SDRAM Controller (SDRAMC) extends the memory capabilities of a chip by providing the interface to an external 16-bit DRAM device. The page size supports ranges from 2048 to 8192 and the number of columns from 256 to 2048. It supports byte (8-bit), half-word (16-bit) and word (32-bit) accesses.

The SDRAM Controller supports a read or write burst length of one location. It keeps track of the active row in each bank, thus maximizing SDRAM performance, e.g., the application may be placed in one bank and data in the other banks. So as to optimize performance, it is advisable to avoid accessing different rows in the same bank.

The SDRAM controller supports a CAS latency of 1, 2 or 3 and optimizes the read access depending on the frequency.

The different modes available - self-refresh, power-down and deep power-down modes - minimize power consumption on the SDRAM device.

24.2 Embedded Characteristics

- Numerous Configurations Supported
  - 2K, 4K, 8K Row Address Memory Parts
  - SDRAM with Two or Four Internal Banks
  - SDRAM with 16-bit Data Path
- Programming Facilities
  - Word, Half-word, Byte Access
  - Automatic Page Break When Memory Boundary Has Been Reached
  - Multibank Ping-pong Access
  - Timing Parameters Specified by Software
  - Automatic Refresh Operation, Refresh Rate is Programmable
  - Automatic Update of DS, TCR and PASR Parameters (Mobile SDRAM Devices)
- Energy-saving Capabilities
  - Self-refresh, Power-down and Deep Power Modes Supported
  - Supports Mobile SDRAM Devices
- Error Detection
  - Refresh Error Interrupt
- SDRAM Power-up Initialization by Software
- CAS Latency of 1, 2, 3 Supported
- Auto Precharge Command Not Used

24.3 I/O Lines Description

Table 24-1. I/O Line Description

Name	Description	Type	Active Level
SDCK	SDRAM Clock	Output	
SDCKE	SDRAM Clock Enable	Output	High
SDCS	SDRAM Controller Chip Select	Output	Low
BA[1:0]	Bank Select Signals	Output	
RAS	Row Signal	Output	Low
CAS	Column Signal	Output	Low
SDWE	SDRAM Write Enable	Output	Low
NBS[3:0]	Data Mask Enable Signals	Output	Low
SDRAMC_A[12:0]	Address Bus	Output	
D[15:0]	Data Bus	I/O	

24.4 Application Example

24.4.1 Software Interface

The SDRAM address space is organized into banks, rows, and columns. The SDRAM controller allows mapping different memory types according to the values set in the SDRAMC configuration register.

The SDRAM Controller's function is to make the SDRAM device access protocol transparent to the user. Table 24-2 to Table 24-4 illustrate the SDRAM device memory mapping seen by the user in correlation with the device structure. Various configurations are illustrated.

24.4.1.1 16-bit Memory Data Bus Width

Table 24-2. SDRAM Configuration Mapping: 2K Rows, 256/512/1024/2048 Columns

CPU Address Line																										
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
						Bk[1:0]		Row[10:0]											Column[7:0]								M0
					Bk[1:0]			Row[10:0]											Column[8:0]								M0
				Bk[1:0]				Row[10:0]											Column[9:0]								M0
			Bk[1:0]					Row[10:0]											Column[10:0]								M0

Table 24-3. SDRAM Configuration Mapping: 4K Rows, 256/512/1024/2048 Columns

CPU Address Line																										
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
					Bk[1:0]			Row[11:0]											Column[7:0]								M0
				Bk[1:0]				Row[11:0]											Column[8:0]								M0
			Bk[1:0]					Row[11:0]											Column[9:0]								M0
		Bk[1:0]						Row[11:0]											Column[10:0]								M0

Table 24-4. SDRAM Configuration Mapping: 8K Rows, 256/512/1024/2048 Columns

CPU Address Line																										
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
				Bk[1:0]				Row[12:0]											Column[7:0]								M0
			Bk[1:0]					Row[12:0]											Column[8:0]								M0
		Bk[1:0]						Row[12:0]											Column[9:0]								M0
	Bk[1:0]							Row[12:0]											Column[10:0]								M0

Notes: 1. M0 is the byte address inside a 16-bit half-word.
2. Bk[1] = BA1, Bk[0] = BA0.

24.5 Product Dependencies

24.5.1 SDRAM Device Initialization

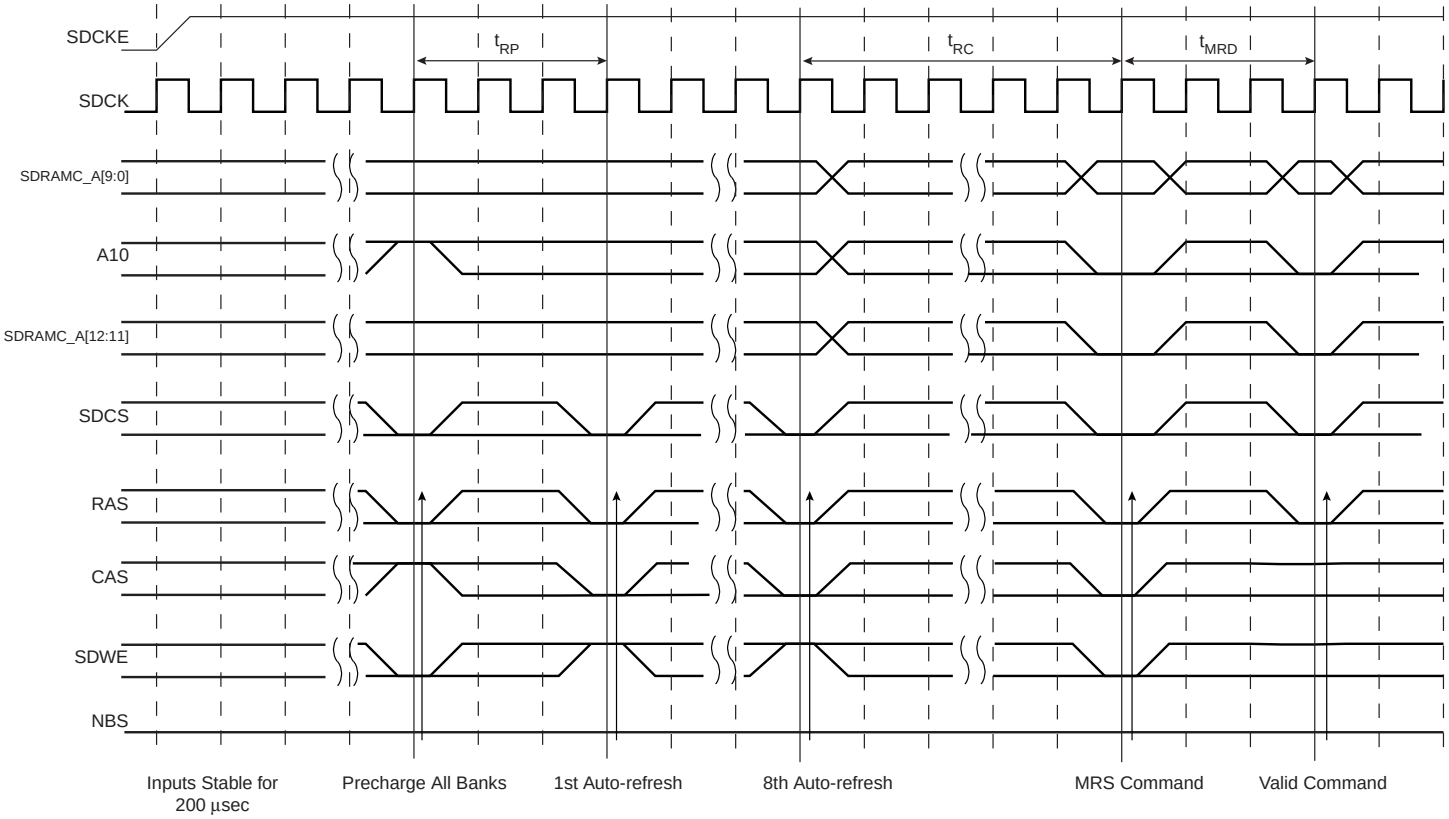
The initialization sequence is generated by software. The SDRAM devices are initialized by the following sequence:

1. SDRAM features must be set in the configuration register: asynchronous timings (TRC, TRAS, etc.), number of columns, rows, CAS latency, and the data bus width.
2. For mobile SDRAM, temperature-compensated self refresh (TCSR), drive strength (DS) and partial array self refresh (PASR) must be set in the Low Power Register.
3. The SDRAM memory type must be set in the Memory Device Register.
4. A minimum pause of 200 µs is provided to precede any signal toggle.
5. (1)A NOP command is issued to the SDRAM devices. The application must set Mode to 1 in the Mode Register and perform a write access to any SDRAM address.
6. An All Banks Precharge command is issued to the SDRAM devices. The application must set Mode to 2 in the Mode Register and perform a write access to any SDRAM address.
7. Eight auto-refresh (CBR) cycles are provided. The application must set the Mode to 4 in the Mode Register and perform a write access to any SDRAM location eight times.
8. A Mode Register set (MRS) cycle is issued to program the parameters of the SDRAM devices, in particular CAS latency and burst length. The application must set Mode to 3 in the Mode Register and perform a write access to the SDRAM. The write address must be chosen so that BA[1:0] are set to 0. For example, with a 16-bit 128 MB SDRAM (12 rows, 9 columns, 4 banks) bank address, the SDRAM write access should be done at the address 0x70000000.
9. For mobile SDRAM initialization, an Extended Mode Register set (EMRS) cycle is issued to program the SDRAM parameters (TCSR, PASR, DS). The application must set Mode to 5 in the Mode Register and perform a write access to the SDRAM. The write address must be chosen so that BA[1] or BA[0] are set to 1. For example, with a 16-bit 128 MB SDRAM, (12 rows, 9 columns, 4 banks) bank address the SDRAM write access should be done at the address 0x70800000 or 0x70400000.
10. The application must go into Normal Mode, setting Mode to 0 in the Mode Register and performing a write access at any location in the SDRAM.
11. Write the refresh rate into the count field in the SDRAMC Refresh Timer register. (Refresh rate = delay between refresh cycles). The SDRAM device requires a refresh every 15.625 µs or 7.81 µs. With a 100 MHz frequency, the Refresh Timer Counter Register must be set with the value 1562(15.625 µs x 100 MHz) or 781(7.81 µs x 100 MHz).

After initialization, the SDRAM devices are fully functional.

Note: 1. It is strongly recommended to respect the instructions stated in Step 5 of the initialization process in order to be certain that the subsequent commands issued by the SDRAMC will be taken into account.

Figure 24-1. SDRAM Device Initialization Sequence



24.5.2 I/O Lines

The pins used for interfacing the SDRAM Controller may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the SDRAM Controller pins to their peripheral function. If I/O lines of the SDRAM Controller are not used by the application, they can be used for other purposes by the PIO Controller.

24.5.3 Interrupt

The SDRAM Controller interrupt (Refresh Error notification) is connected to the Memory Controller.

Using the SDRAM Controller interrupt requires the Interrupt Controller to be programmed first.

Table 24-5. Peripheral IDs

Instance	ID
SDRAMC	10

24.5.4 Power Management

The SDRAM Controller may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SDRAM Controller clock. The SDRAM Controller Clock (not the SDCK pin) is managed by the Static Memory Controller Clock.

The SDRAM Clock on SDCK pin will be output as soon as the first access to the SDRAM is done during the initialization phase.

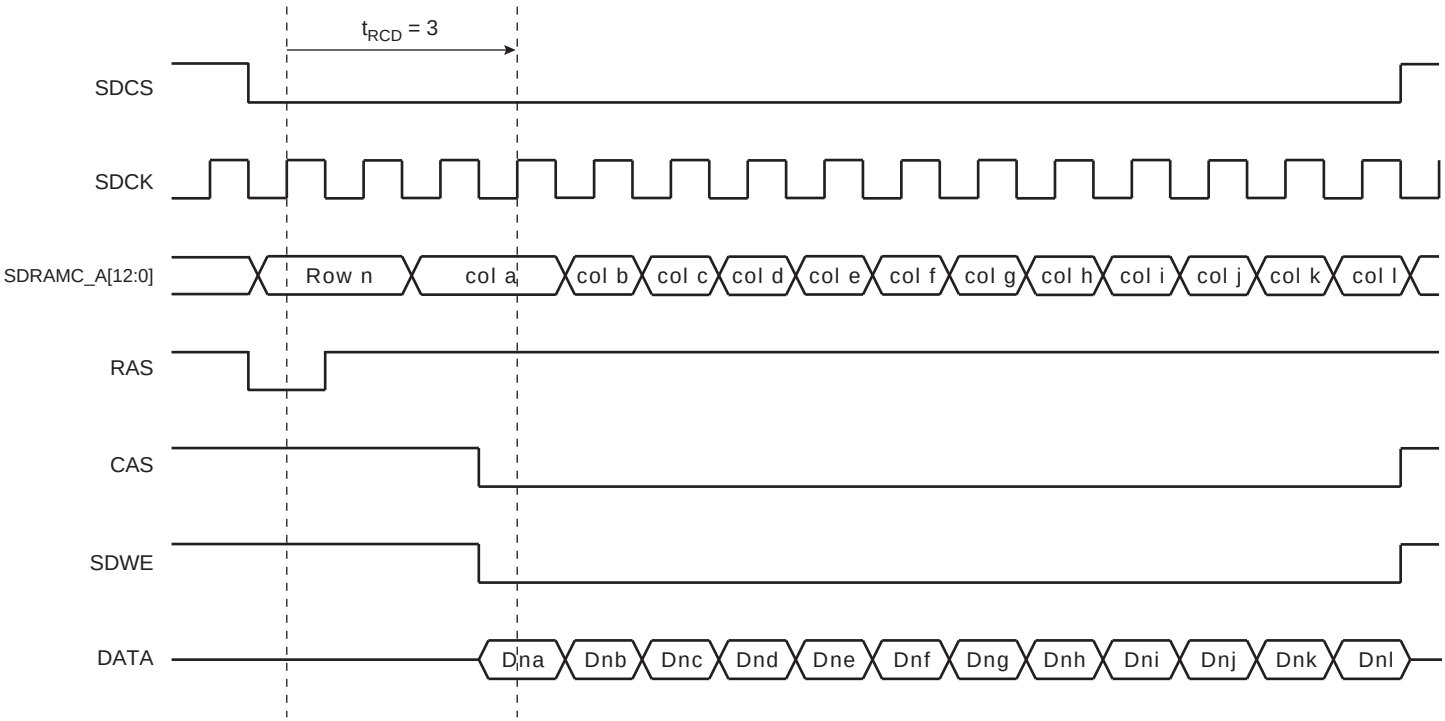
If one needs to stop the SDRAM clock signal, the Low Power Mode Register (SDRAMC_LPR) must be programmed with the sefl-refresh command.

24.6 Functional Description

24.6.1 SDRAM Controller Write Cycle

The SDRAM Controller allows burst access or single access. In both cases, the SDRAM controller keeps track of the active row in each bank, thus maximizing performance. To initiate a burst access, the SDRAM Controller uses the transfer type signal provided by the master requesting the access. If the next access is a sequential write access, writing to the SDRAM device is carried out. If the next access is a write-sequential access, but the current access is to a boundary page, or if the next access is in another row, then the SDRAM Controller generates a precharge command, activates the new row and initiates a write command. To comply with SDRAM timing parameters, additional clock cycles are inserted between precharge/active (t_{RP}) commands and active/write (t_{RCD}) commands. For definition of these timing parameters, refer to the “SDRAMC Configuration Register” on page 408. This is described in Figure 24-2 below.

Figure 24-2. Write Burst SDRAM Access



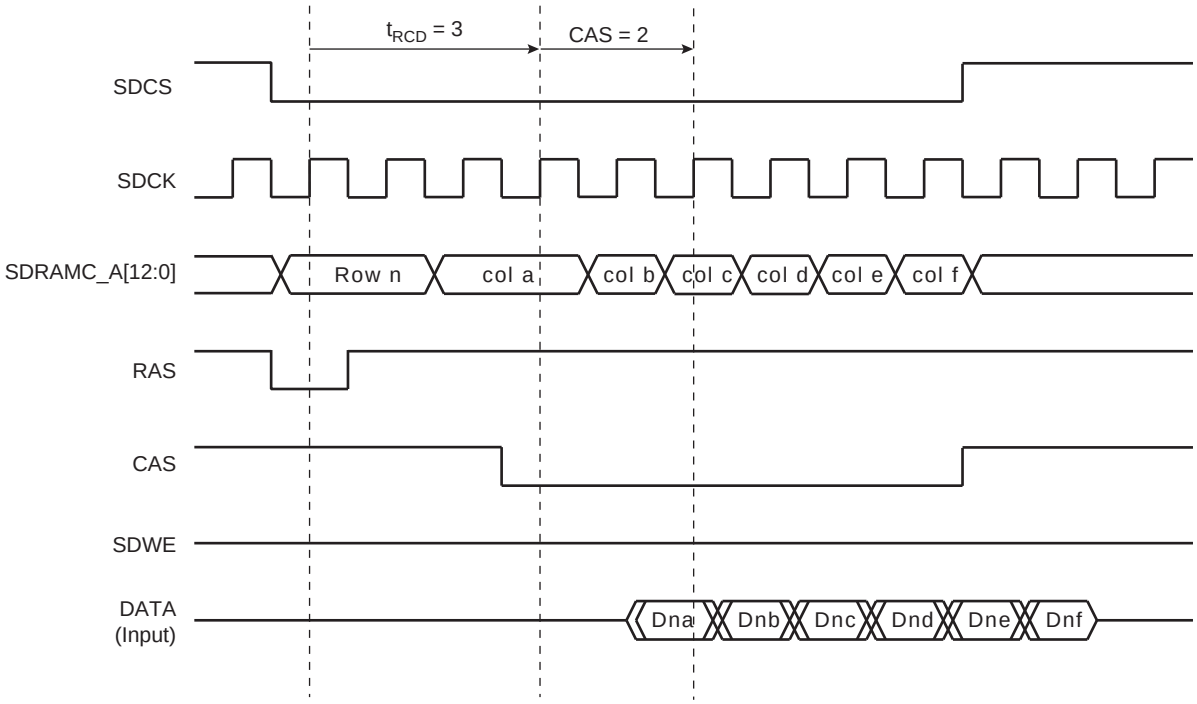
24.6.2 SDRAM Controller Read Cycle

The SDRAM Controller allows burst access, incremental burst of unspecified length or single access. In all cases, the SDRAM Controller keeps track of the active row in each bank, thus maximizing performance of the SDRAM. If row and bank addresses do not match the previous row/bank address, then the SDRAM controller automatically generates a precharge command, activates the new row and starts the read command. To comply with the SDRAM timing parameters, additional clock cycles on SDCK are inserted between precharge and active commands (t_{RP}) and between active and read command (t_{RCD}). These two parameters are set in the configuration register of the SDRAM Controller. After a read command, additional wait states are generated to comply with the CAS latency (1, 2 or 3 clock delays specified in the configuration register).

For a single access or an incremented burst of unspecified length, the SDRAM Controller anticipates the next access. While the last value of the column is returned by the SDRAM Controller on the bus, the SDRAM Controller anticipates the read to the next column and thus anticipates the CAS latency. This reduces the effect of the CAS latency on the internal bus.

For burst access of specified length (4, 8, 16 words), access is not anticipated. This case leads to the best performance. If the burst is broken (border, busy mode, etc.), the next access is handled as an incrementing burst of unspecified length.

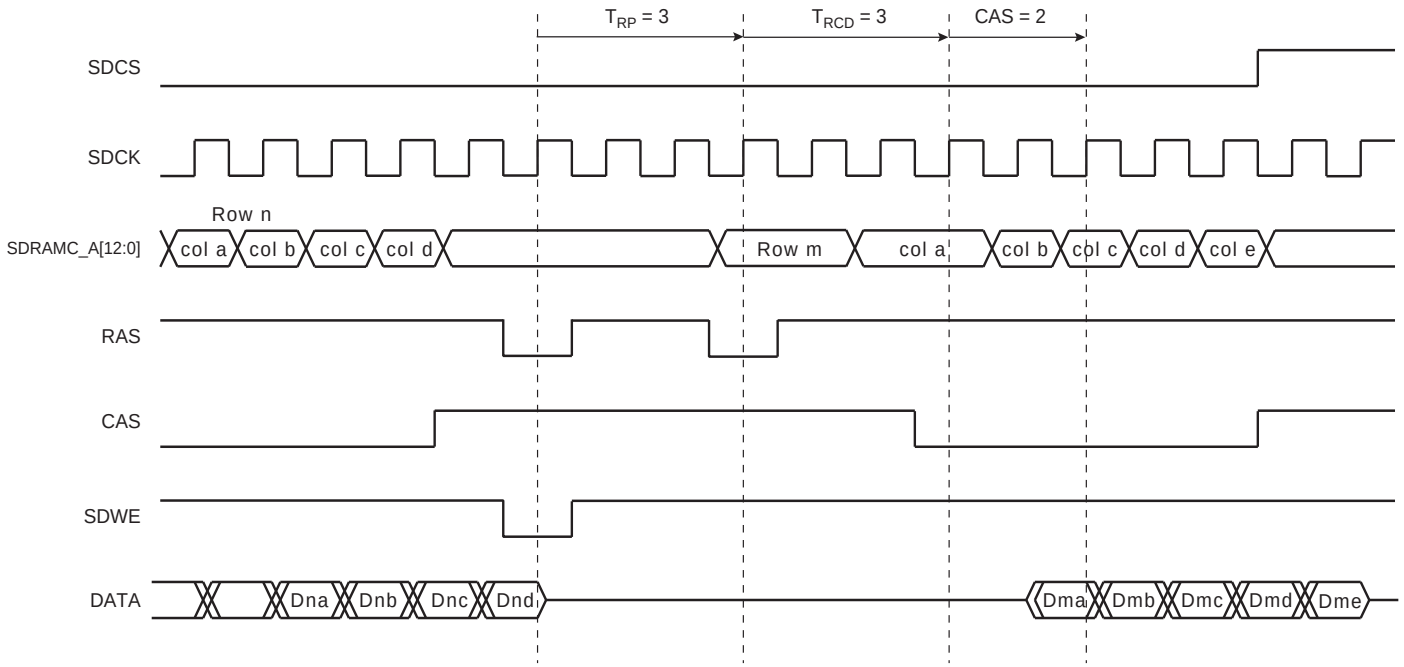
Figure 24-3. Read Burst SDRAM Access



24.6.3 Border Management

When the memory row boundary has been reached, an automatic page break is inserted. In this case, the SDRAM controller generates a precharge command, activates the new row and initiates a read or write command. To comply with SDRAM timing parameters, an additional clock cycle is inserted between the precharge/active (t_{RP}) command and the active/read (t_{RCD}) command. This is described in Figure 24-4 below.

Figure 24-4. Read Burst with Boundary Row Access



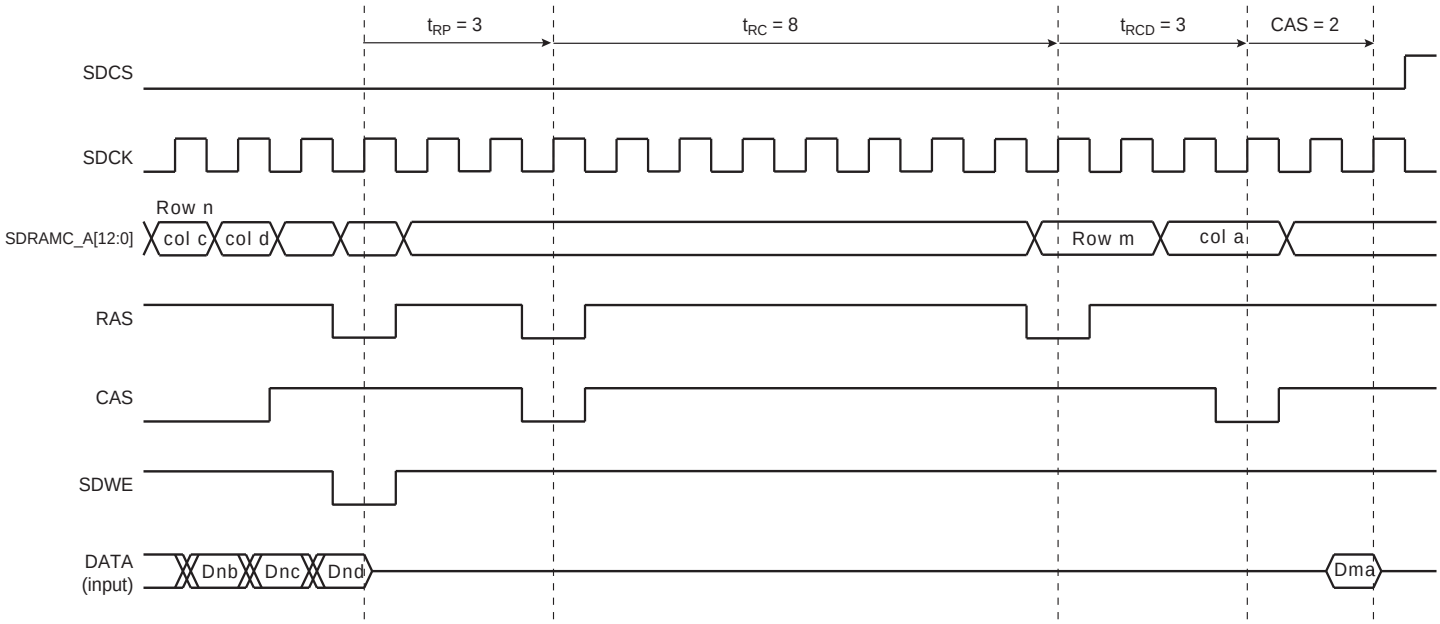
24.6.4 SDRAM Controller Refresh Cycles

An auto-refresh command is used to refresh the SDRAM device. Refresh addresses are generated internally by the SDRAM device and incremented after each auto-refresh automatically. The SDRAM Controller generates these auto-refresh commands periodically. An internal timer is loaded with the value in the register SDRAMC_TR that indicates the number of clock cycles between refresh cycles.

A refresh error interrupt is generated when the previous auto-refresh command did not perform. It is acknowledged by reading the Interrupt Status Register (SDRAMC_ISR).

When the SDRAM Controller initiates a refresh of the SDRAM device, internal memory accesses are not delayed. However, if the CPU tries to access the SDRAM, the slave indicates that the device is busy and the master is held by a wait signal. See Figure 24-5.

Figure 24-5. Refresh Cycle Followed by a Read Access



24.6.5 Power Management

Three low-power modes are available:

- Self-refresh Mode: The SDRAM executes its own Auto-refresh cycle without control of the SDRAM Controller. Current drained by the SDRAM is very low.
- Power-down Mode: Auto-refresh cycles are controlled by the SDRAM Controller. Between auto-refresh cycles, the SDRAM is in power-down. Current drained in Power-down mode is higher than in Self-refresh Mode.
- Deep Power-down Mode: (Only available with Mobile SDRAM) The SDRAM contents are lost, but the SDRAM does not drain any current.

The SDRAM Controller activates one low-power mode as soon as the SDRAM device is not selected. It is possible to delay the entry in self-refresh and power-down mode after the last access by programming a timeout value in the Low Power Register.

24.6.5.1 Self-refresh Mode

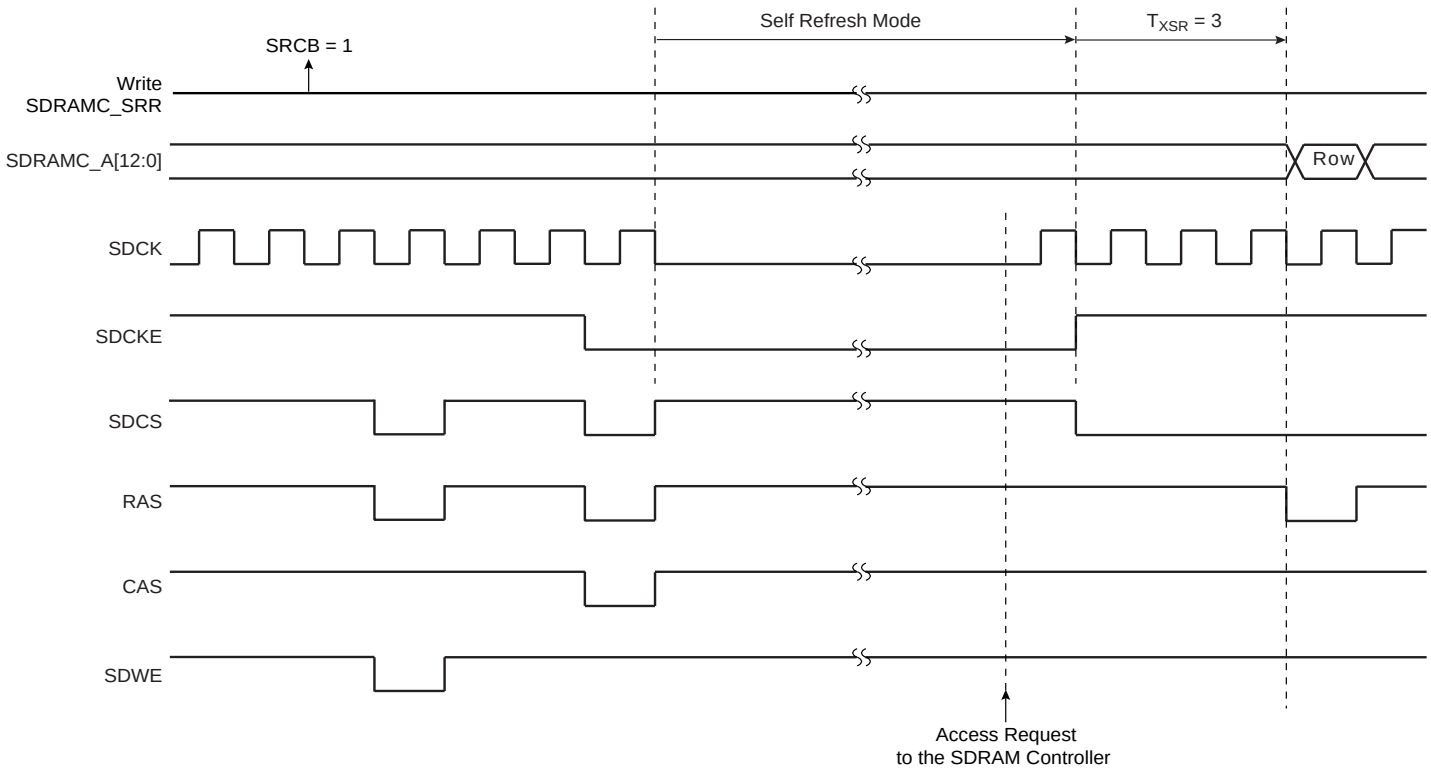
This mode is selected by programming the LPCB field to 1 in the SDRAMC Low Power Register. In self-refresh mode, the SDRAM device retains data without external clocking and provides its own internal clocking, thus performing its own auto-refresh cycles. All the inputs to the SDRAM device become “don’t care” except SDCKE, which remains low. As soon as the SDRAM device is selected, the SDRAM Controller provides a sequence of commands and exits self-refresh mode.

Some low-power SDRAMs (e.g., mobile SDRAM) can refresh only one quarter or a half quarter or all banks of the SDRAM array. This feature reduces the self-refresh current. To configure this feature, Temperature Compensated Self Refresh (TCSR), Partial Array Self Refresh (PASR) and Drive Strength (DS) parameters must be set in the Low Power Register and transmitted to the low-power SDRAM during initialization.

After initialization, as soon as PASR/DS/TCSR fields are modified and self-refresh mode is activated, the Extended Mode Register is accessed automatically and PASR/DS/TCSR bits are updated before entry into self-refresh mode. This feature is not supported when SDRAMC shares an external bus with another controller.

The SDRAM device must remain in self-refresh mode for a minimum period of t_{RAS} and may remain in self-refresh mode for an indefinite period. This is described in Figure 24-6.

Figure 24-6. Self-refresh Mode Behavior

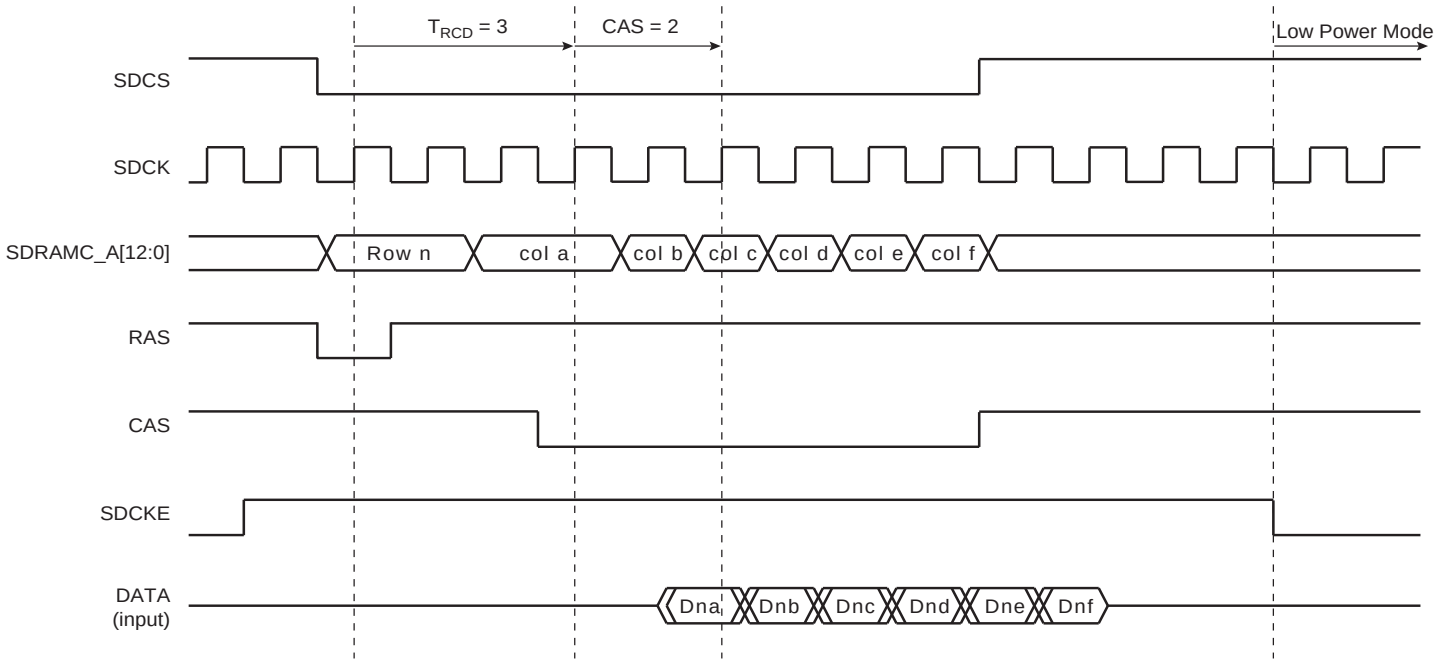


24.6.5.2 Low-power Mode

This mode is selected by programming the LPCB field to 2 in the SDRAMC Low Power Register. Power consumption is greater than in self-refresh mode. All the input and output buffers of the SDRAM device are deactivated except SDCKE, which remains low. In contrast to self-refresh mode, the SDRAM device cannot remain in low-power mode longer than the refresh period (64 ms for a whole device refresh operation). As no auto-refresh operations are performed by the SDRAM itself, the SDRAM Controller carries out the refresh operation. The exit procedure is faster than in self-refresh mode.

This is described in Figure 24-7.

Figure 24-7. Low-power Mode Behavior



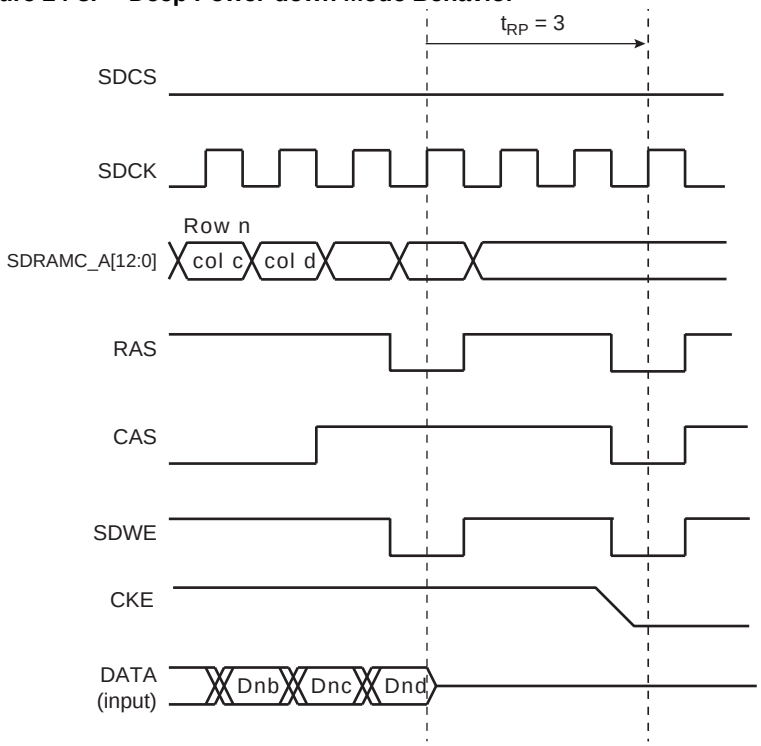
24.6.5.3 Deep Power-down Mode

This mode is selected by programming the LPCB field to 3 in the SDRAMC Low Power Register. When this mode is activated, all internal voltage generators inside the SDRAM are stopped and all data is lost.

When this mode is enabled, the application must not access to the SDRAM until a new initialization sequence is done (See “SDRAM Device Initialization” on page 395).

This is described in Figure 24-8.

Figure 24-8. Deep Power-down Mode Behavior



24.7 AHB SDRAM Controller (SDRAMC) User Interface

Table 24-6. Register Mapping

Offset	Register	Name	Access	Reset
0x00	SDRAMC Mode Register	SDRAMC_MR	Read-write	0x00000000
0x04	SDRAMC Refresh Timer Register	SDRAMC_TR	Read-write	0x00000000
0x08	SDRAMC Configuration Register	SDRAMC_CR	Read-write	0x852372C0
0x10	SDRAMC Low Power Register	SDRAMC_LPR	Read-write	0x0
0x14	SDRAMC Interrupt Enable Register	SDRAMC_IER	Write-only	–
0x18	SDRAMC Interrupt Disable Register	SDRAMC_IDR	Write-only	–
0x1C	SDRAMC Interrupt Mask Register	SDRAMC_IMR	Read-only	0x0
0x20	SDRAMC Interrupt Status Register	SDRAMC_ISR	Read-only	0x0
0x24	SDRAMC Memory Device Register	SDRAMC_MDR	Read-write	0x0
0x28	SDRAMC Configuration Register 1	SDRAMC_CR1	Read-write	0x02
0x2C	SDRAMC OCMS Register 1	SDRAMC_OCMS	Read-write	0x0000000
0x30 - 0xF8	Reserved	–	–	–
0x28 - 0xFC	Reserved	–	–	–

24.7.1 SDRAMC Mode Register

Name: SDRAMC_MR

Address: 0x400E0200

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
–	–	–	–	–	MODE		

- **MODE: SDRAMC Command Mode**

This field defines the command issued by the SDRAM Controller when the SDRAM device is accessed.

Value	Name	Description
000	NORMAL	Normal mode. Any access to the SDRAM is decoded normally. To activate this mode, command must be followed by a write to the SDRAM.
001	NOP	The SDRAM Controller issues a NOP command when the SDRAM device is accessed regardless of the cycle. To activate this mode, command must be followed by a write to the SDRAM.
010	ALLBANKS_PRECHARGE	The SDRAM Controller issues an “All Banks Precharge” command when the SDRAM device is accessed regardless of the cycle. To activate this mode, command must be followed by a write to the SDRAM.
011	LOAD_MODEREG	The SDRAM Controller issues a “Load Mode Register” command when the SDRAM device is accessed regardless of the cycle. To activate this mode, command must be followed by a write to the SDRAM.
100	AUTO_REFRESH	The SDRAM Controller issues an “Auto-Refresh” Command when the SDRAM device is accessed regardless of the cycle. Previously, an “All Banks Precharge” command must be issued. To activate this mode, command must be followed by a write to the SDRAM.
101	EXT_LOAD_MODEREG	The SDRAM Controller issues an “Extended Load Mode Register” command when the SDRAM device is accessed regardless of the cycle. To activate this mode, the “Extended Load Mode Register” command must be followed by a write to the SDRAM. The write in the SDRAM must be done in the appropriate bank; most low-power SDRAM devices use the bank 1.
110	DEEP_POWERDOWN	Deep power-down mode. Enters deep power-down mode.

24.7.2 SDRAMC Refresh Timer Register

Name: SDRAMC_TR

Address: 0x400E0204

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
–	–	–	–	COUNT			

7	6	5	4	3	2	1	0
COUNT							

- **COUNT: SDRAMC Refresh Timer Count**

This 12-bit field is loaded into a timer that generates the refresh pulse. Each time the refresh pulse is generated, a refresh burst is initiated. The SDRAM device requires a refresh every 15.625 µs or 7.81 µs. With a 100 MHz frequency, the Refresh Timer Counter Register must be set with the value 1562 (15.625 µs x 100 MHz) or 781 (7.81 µs x 100 MHz).

To refresh the SDRAM device, this 12-bit field must be written. If this condition is not satisfied, no refresh command is issued and no refresh of the SDRAM device is carried out.

24.7.3 SDRAMC Configuration Register

Name: SDRAMC_CR

Address: 0x400E0208

Access: Read-write

Reset: 0x852372C0

31	30	29	28	27	26	25	24
TXSR				TRAS			
23	22	21	20	19	18	17	16
TRCD				TRP			
15	14	13	12	11	10	9	8
TRC_TRFC				TWR			
7	6	5	4	3	2	1	0
DBW	CAS		NB	NR		NC	

Warning: bit 7 (DBW) must always be set to 1 when programming the SDRAMC_CR register.

- **NC: Number of Column Bits**

Reset value is 8 column bits.

Value	Name	Description
00	COL8	8 column bits
01	COL9	9 column bits
10	COL10	10 column bits
11	COL11	11 column bits

- **NR: Number of Row Bits**

Reset value is 11 row bits.

Value	Name	Description
00	ROW11	11 row bits
01	ROW12	12 row bits
10	ROW13	13 row bits

Values which are not listed in the table must be considered as “reserved”.

- **NB: Number of Banks**

Reset value is two banks.

Value	Name	Description
0	BANK2	2 banks
1	BANK4	4 banks

- **CAS: CAS Latency**

Reset value is two cycles.

In the SDRAMC, only a CAS latency of one, two and three cycles are managed.

Value	Name	Description
01	LATENCY1	1 cycle CAS latency
10	LATENCY2	2 cycle CAS latency
11	LATENCY3	3 cycle CAS latency

Values which are not listed in the table must be considered as “reserved”.

- **DBW: Data Bus Width**

Reset value is 16 bits

This field defines the Data Bus Width, which is 16 bits. It must be set to 1.

- **TWR: Write Recovery Delay**

Reset value is two cycles.

This field defines the Write Recovery Time in number of cycles. Number of cycles is between 0 and 15.

- **TRC_TRFC: Row Cycle Delay and Row Refresh Cycle**

Reset value is seven cycles.

This field defines two timings:

the delay (t_{RFC}) between two Refresh commands,

the delay (t_{RFC}) between Refresh command and an Activate command

and the delay (t_{RC}) between two Active commands in number of cycles.

The number of cycles is between 0 and 15. The end user will have to program max {t_{RC}, t_{RFC}}.

- **TRP: Row Precharge Delay**

Reset value is three cycles.

This field defines the delay between a Precharge Command and another Command in number of cycles. Number of cycles is between 0 and 15.

- **TRCD: Row to Column Delay**

Reset value is two cycles.

This field defines the delay between an Activate Command and a Read/Write Command in number of cycles. Number of cycles is between 0 and 15.

- **TRAS: Active to Precharge Delay**

Reset value is five cycles.

This field defines the delay between an Activate Command and a Precharge Command in number of cycles. Number of cycles is between 0 and 15.

- **TXSR: Exit Self Refresh to Active Delay**

Reset value is eight cycles.

This field defines the delay between SCKE set high and an Activate Command in number of cycles. Number of cycles is between 0 and 15.

24.7.4 SDRAMC Low Power Register

Name: SDRAMC_LPR

Address: 0x400E0210

Access: Read-write

Reset: 0x0

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

- **LPCB: Low-power Configuration Bits**

Value	Name	Description
00	DISABLED	Low Power Feature is inhibited: no Power-down, Self-refresh or Deep Power-down command is issued to the SDRAM device.
01	SELF_REFRESH	The SDRAM Controller issues a Self-refresh command to the SDRAM device, the SDCK clock is deactivated and the SDCKE signal is set low. The SDRAM device leaves the Self Refresh Mode when accessed and enters it after the access.
10	POWER_DOWN	The SDRAM Controller issues a Power-down Command to the SDRAM device after each access, the SDCKE signal is set to low. The SDRAM device leaves the Power-down Mode when accessed and enters it after the access.
11	DEEP_POWER_DOWN	The SDRAM Controller issues a Deep Power-down command to the SDRAM device. This mode is unique to low-power SDRAM.

- **PASR: Partial Array Self-refresh (only for low-power SDRAM)**

PASR parameter is transmitted to the SDRAM during initialization to specify whether only one quarter, one half or all banks of the SDRAM array are enabled. Disabled banks are not refreshed in self-refresh mode. This parameter must be set according to the SDRAM device specification.

After initialization, as soon as PASR field is modified and self-refresh mode is activated, the Extended Mode Register is accessed automatically and PASR bits are updated before entry in self-refresh mode.This feature is not supported when SDRAMC shares an external bus with another controller.

- **TCSR: Temperature Compensated Self-Refresh (only for low-power SDRAM)**

TCSR parameter is transmitted to the SDRAM during initialization to set the refresh interval during self-refresh mode depending on the temperature of the low-power SDRAM. This parameter must be set according to the SDRAM device specification.

After initialization, as soon as TCSR field is modified and self-refresh mode is activated, the Extended Mode Register is accessed automatically and TCSR bits are updated before entry in self-refresh mode.This feature is not supported when SDRAMC shares an external bus with another controller.

- **DS: Drive Strength (only for low-power SDRAM)**

DS parameter is transmitted to the SDRAM during initialization to select the SDRAM strength of data output. This parameter must be set according to the SDRAM device specification.

After initialization, as soon as DS field is modified and self-refresh mode is activated, the Extended Mode Register is accessed automatically and DS bits are updated before entry in self-refresh mode. This feature is not supported when SDRAMC shares an external bus with another controller.

- **TIMEOUT: Time to define when low-power mode is enable**

Value	Name	Description
00	LP_LAST_XFER	The SDRAM controller activates the SDRAM low-power mode immediately after the end of the last transfer.
01	LP_LAST_XFER_64	The SDRAM controller activates the SDRAM low-power mode 64 clock cycles after the end of the last transfer.
10	LP_LAST_XFER_128	The SDRAM controller activates the SDRAM low-power mode 128 clock cycles after the end of the last transfer.

Values which are not listed in the table must be considered as “reserved”.

24.7.5 SDRAMC Interrupt Enable Register

Name: SDRAMC_IER

Address: 0x400E0214

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
–	–	–	–	–	–	–	RES

- **RES: Refresh Error Status**

0: No effect.

1: Enables the refresh error interrupt.

24.7.6 SDRAMC Interrupt Disable Register

Name: SDRAMC_IDR

Address: 0x400E0218

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
–	–	–	–	–	–	–	RES

- **RES: Refresh Error Status**

0: No effect.

1: Disables the refresh error interrupt.

24.7.7 SDRAMC Interrupt Mask Register

Name: SDRAMC_IMR

Address: 0x400E021C

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
–	–	–	–	–	–	–	RES

- **RES: Refresh Error Status**

0: The refresh error interrupt is disabled.

1: The refresh error interrupt is enabled.

24.7.8 SDRAMC Interrupt Status Register

Name: SDRAMC_ISR

Address: 0x400E0220

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
–	–	–	–	–	–	–	RES

- **RES: Refresh Error Status**

0: No refresh error has been detected since the register was last read.

1: A refresh error has been detected since the register was last read.

24.7.9 SDRAMC Memory Device Register

Name: SDRAMC_MDR

Address: 0x400E0224

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
–	–	–	–	–	–	MD	

- **MD: Memory Device Type**

Value	Name	Description
00	SDRAM	SDRAM
01	LPSDRAM	Low-power SDRAM

Values which are not listed in the table must be considered as “reserved”.

24.7.10 SDRAMC Configuration 1 Register

Name: SDRAMC_CR1

Address: 0x400E0228

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
–	–	–	–	TMRD			

- **TMRD: Load Mode Register Command to Active or Refresh Command**

Reset Value is 2 cycles.

This field defines the delay between a Load mode register command and an active or refresh command in number of cycles. Number of cycles is between 0 and 15.

24.7.11 SDRAMC OCMS Register

Name: SDRAMC_OCMS

Address: 0x400E022C

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
							SDR_SE

- **SDR_SE: SDRAM Memory Controller Scrambling Enable**

0: Disable “Off Chip” Scrambling for SDR-SDRAM access.

1: Enable “Off Chip” Scrambling for SDR-SDRAM access.

Before scrambling SDR-SDRAM access, the end user must set to 1 the bit field SMSE (Static Memory Controller Scrambling Enable) in the SMC_OCMS register, located in the Static Memory Controller block.

25. Static Memory Controller (SMC)

25.1 Description

The External Bus Interface is designed to ensure the successful data transfer between several external devices and the Cortex-M3 based device. The External Bus Interface of the SAM3X 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 SMC generates the signals that control the access to external memory devices or peripheral devices. It has 8 Chip Selects and a 24-bit address bus. The 16-bit data bus can be configured to interface with 8- or 16-bit external devices. Separate read and write control signals allow for direct memory and peripheral interfacing. Read and write signal waveforms are fully parametrizable.

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 embeds a NAND Flash Controller (NFC). The NFC can handle automatic transfers, sending the commands and address cycles to the NAND Flash and transferring the contents of the page (for read and write) to the NFC SRAM. It minimizes the CPU overhead.

The SMC includes programmable hardware error correcting code with one bit error correction capability and supports two bits error detection. In order to improve overall system performance the DATA phase of the transfer can be DMA assisted.

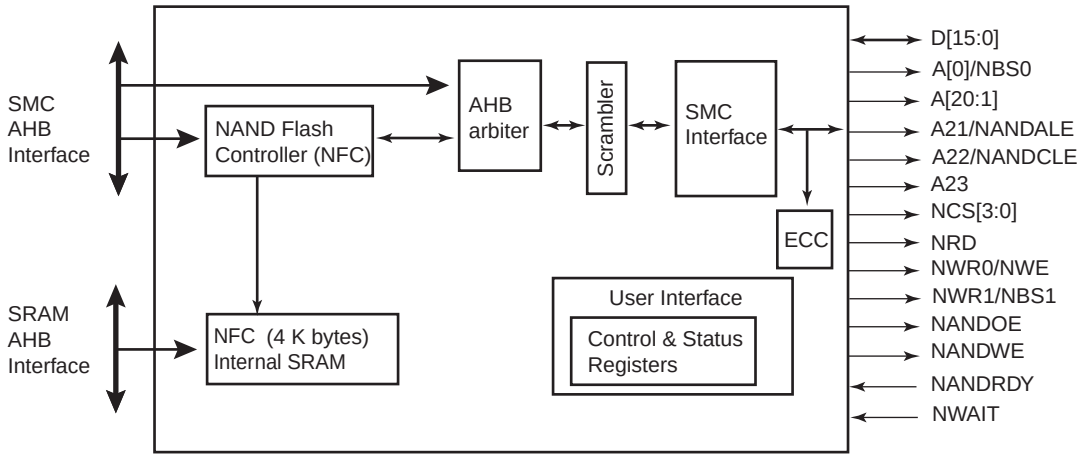
The External Data Bus can be scrambled/unscrambled by means of user keys.

25.2 Embedded Characteristics

- 16-Mbyte Address Space per Chip Select
- 8- or 16-bit Data Bus
- Word, Halfword, Byte Transfers
- Byte Write or Byte Select Lines
- 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
- NAND Flash Controller Supporting NAND Flash with Multiplexed Data/Address Buses
- Supports SLC NAND Flash Technology
- Hardware Configurable Number of Chip Selects from 1 to 8
- Programmable Timing on a per Chip Select Basis
- AHB Slave Interface
- Atmel APB Configuration Interface
- Programmable Flash Data Width 8 Bits or 16 Bits
- Supports Hardware Error Correcting Code (ECC), 1-bit Error Correction, 2-bit Error Detection
- Detection and Correction by Software
- Supports NAND Flash and SmartMedia™ Devices with 8- or 16-bit Data Path
- Supports NAND Flash/SmartMedia with Page Sizes of 528, 1056, 2112 and 4224 Bytes, Specified by Software
- **Supports 1-bit Correction for a Page of 512, 1024, 2048 and 4096 Bytes** with 8- or 16-bit Data Path
- Supports 1-bit Correction per 512 Bytes of Data for a Page Size of 512, 2048 and 4096 Bytes with 8-bit Data Path
- Supports 1-bit Correction per 256 Bytes of Data for a Page Size of 512, 2048 and 4096 Bytes with 8-bit Data Path

25.3 Block Diagram

Figure 25-1. Block Diagram



25.4 I/O Lines Description

Table 25-1. 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
NWR0/NWE	Write 0/Write Enable Signal	Output	Low
A0/NBS0	Address Bit 0/Byte 0 Select Signal	Output	Low
NWR1/NBS1	Write 1/Byte 1 Select Signal	Output	Low
A1	Address Bit 1	Output	Low
A[23:2]	Address Bus	Output	
D[15:0]	Data Bus	I/O	
NWAIT	External Wait Signal	Input	Low
NANDRDY	NAND Flash Ready/Busy	Input	
NANDWE	NAND Flash Write Enable	Output	Low
NANDOE	NAND Flash Output Enable	Output	Low
NANDALE	NAND Flash Address Latch Enable	Output	
NANDCLE	NAND Flash Command Latch Enable	Output	

25.5 Multiplexed Signals

Table 25-2. Static Memory Controller (SMC) Multiplexed Signals

Multiplexed Signals		Related Function
NWR0	NWE	Byte-write or byte-select access, see Figure 25-4 "Memory Connection for an 8-bit Data Bus" and Figure 25-5 "Memory Connection for a 16-bit Data Bus"
A0	NBS0	8-bit or 16-bit data bus, see Section 25.9.1 ”Data Bus Width”
A22	NANDCLE	NAND Flash Command Latch Enable
A21	NANDALE	NAND Flash Address Latch Enable
NWR1	NBS1	Byte-write or byte-select access, see Figure 25-4 and Figure 25-5
A1	–	8-/16-bit data bus, see Section 25.9.1 ”Data Bus Width” Byte-write or byte-select access, see Figure 25-4 and Figure 25-5

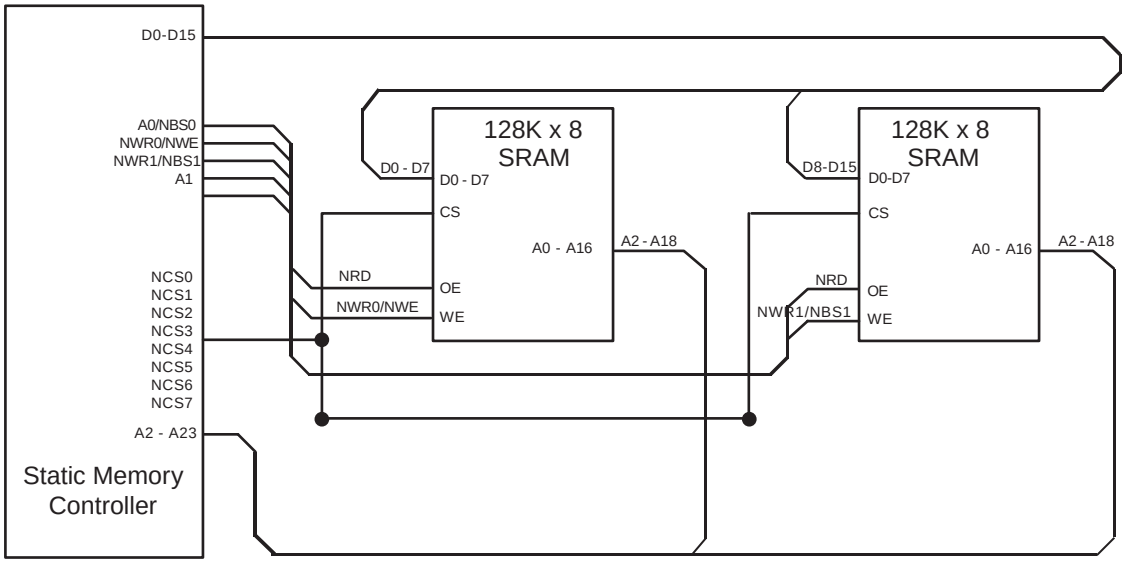
25.6 Application Example

25.6.1 Implementation Examples

For Hardware implementation examples, please refer to ATSAM3X-EK schematics which show examples of connection to an LCD module, PSRAM and NAND Flash.

25.6.2 Hardware Interface

Figure 25-2. SMC Connections to Static Memory Devices



25.7 Product Dependencies

25.7.1 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.

25.7.2 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.

25.7.3 Interrupt

The SMC has an interrupt line connected to the Nested Vector Interrupt Controller (NVIC). Handling the SMC interrupt requires programming the NVIC before configuring the SMC.

Table 25-3. Peripheral IDs

Instance	ID
SMC	9

25.8 External Memory Mapping

Table 25-4. External Memory Mapping

Address	Use	Access
0x60000000-0x60FFFFFF	Chip Select 0 (16 MB)	Read-write
0x61000000-0x61FFFFFF	Chip Select 1	Read-write
0x62000000-0x62FFFFFF	Chip Select 2	Read-write
0x63000000-0x63FFFFFF	Chip Select 3	Read-write
0x64000000-0x64FFFFFF	Chip Select 4	Read-write
0x65000000-0x65FFFFFF	Chip Select 5	Read-write
0x66000000-0x66FFFFFF	Chip Select 6	Read-write
0x67000000-0x67FFFFFF	Chip Select 7	Read-write
0x68000000-0x6FFFFFFF	NFC Command Registers(1)	Read-write

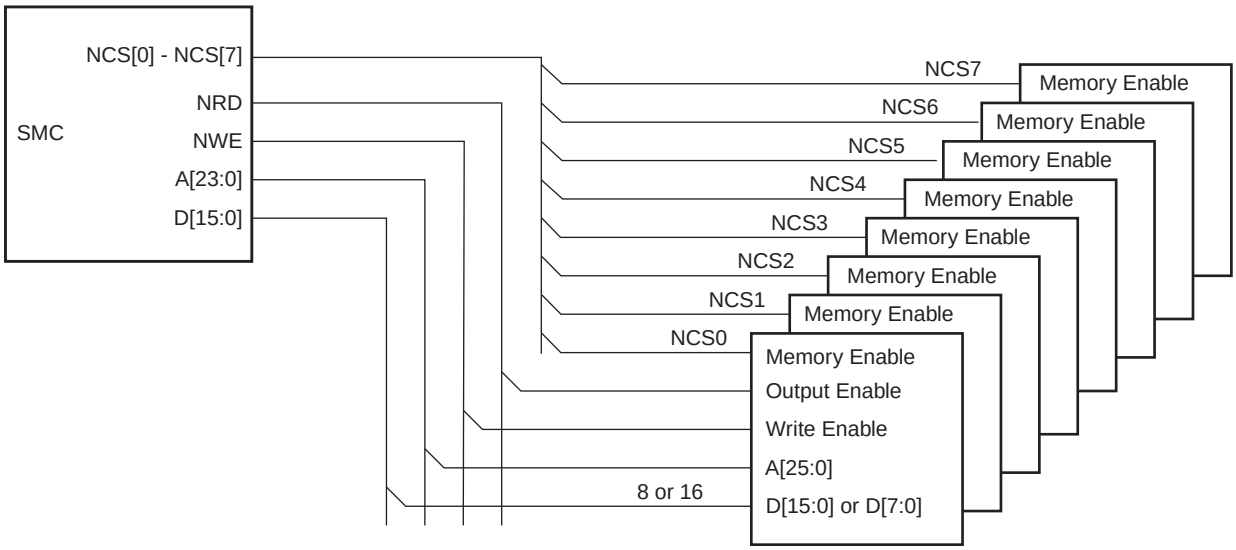
Note: 1. See Section 25.16.2 "NFC Control Registers", i.e., CMD_ADDR description.

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 25-3).

A[23:0] is only significant for 8-bit memory, A[23:1] is used for 16-bit memory.

Figure 25-3. Memory Connections for External Devices



25.9 Connection to External Devices

25.9.1 Data Bus Width

A data bus width of 8 or 16 bits can be selected for each chip select. This option is controlled by the field DBW in SMC_MODE (Mode Register) for the corresponding chip select.

Figure 25-4 shows how to connect a 512K x 8-bit memory on NCS2. Figure 25-5 shows how to connect a 512K x 16-bit memory on NCS2.

25.9.2 Byte Write or Byte Select Access

Each chip select with a 16-bit data bus can operate with one of two different types of write access: byte write or byte select access. This is controlled by the BAT field of the SMC_MODE register for the corresponding chip select.

Figure 25-4. Memory Connection for an 8-bit Data Bus

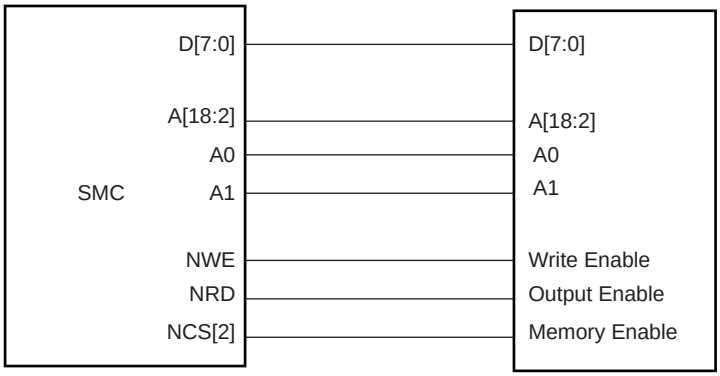
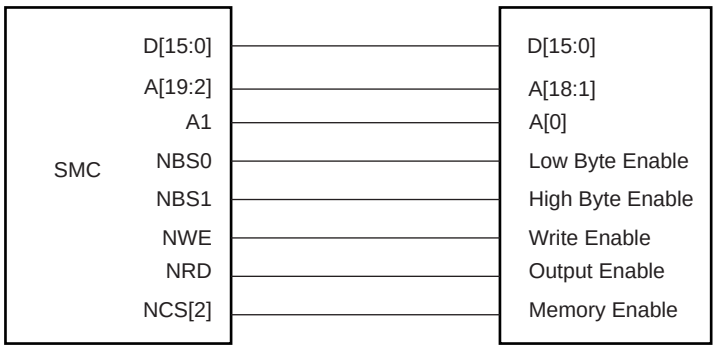


Figure 25-5. Memory Connection for a 16-bit Data Bus



25.9.2.1 Byte Write Access

Byte write access supports one byte write signal per byte of the data bus and a single read signal.

Note that the SMC does not allow boot in Byte Write Access mode.

- For 16-bit devices: the SMC provides NWR0 and NWR1 write signals for respectively, byte0 (lower byte) and byte1 (upper byte) of a 16-bit bus. One single read signal (NRD) is provided.

  Byte Write Access is used to connect 2 x 8-bit devices as a 16-bit memory.

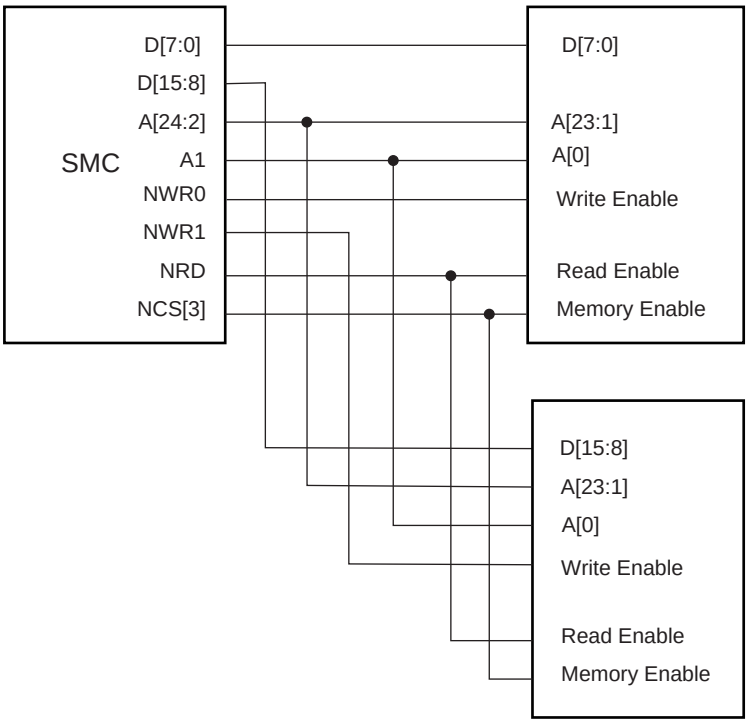
25.9.2.2 Byte Select Access

In this mode, read/write operations can be enabled/disabled at byte level. One byte-select line per byte of the data bus is provided. One NRD and one NWE signal control read and write.

- For 16-bit devices: the SMC provides NBS0 and NBS1 selection signals for respectively byte0 (lower byte) and byte1 (upper byte) of a 16-bit bus.

  Byte Select Access is used to connect one 16-bit device.

Figure 25-6. Connection of 2 x 8-bit Devices on a 16-bit Bus: Byte Write Option



25.9.2.3 Signal Multiplexing

Depending on the byte access type (BAT), only the write signals or the byte select signals are used. To save IOs at the external bus interface, control signals at the SMC interface are multiplexed. Table 25-5 shows signal multiplexing depending on the data bus width and the byte access type.

For 16-bit devices, bit A0 of address is unused. When Byte Select Option is selected, NWR1 is unused. When Byte Write option is selected, NBS0 is unused.

Table 25-5. SMC Multiplexed Signal Translation

Signal Name	16-bit Bus		8-bit Bus
Device Type	**1x16-bit**	**2 x 8-bit**	**1 x 8-bit**
Byte Access Type (BAT)	Byte Select	Byte Write	
NBS0_A0	NBS0		A0
NWE_NWR0	NWE	NWR0	NWE
NBS1_NWR1	NBS1	NWR1	
A1	A1	A1	A1

25.10 Standard Read and Write Protocols

In the following sections, the byte access type is not considered. Byte select lines (NBS0 to NBS1) always have the same timing as the A address bus. NWE represents either the NWE signal in byte select access type or one of the byte write lines (NWR0 to NWR1) in byte write access type. NWR0 to NWR3 have the same timings and protocol as NWE. In the same way, NCS represents one of the NCS[0..7] chip select lines.

25.10.1 Read Waveforms

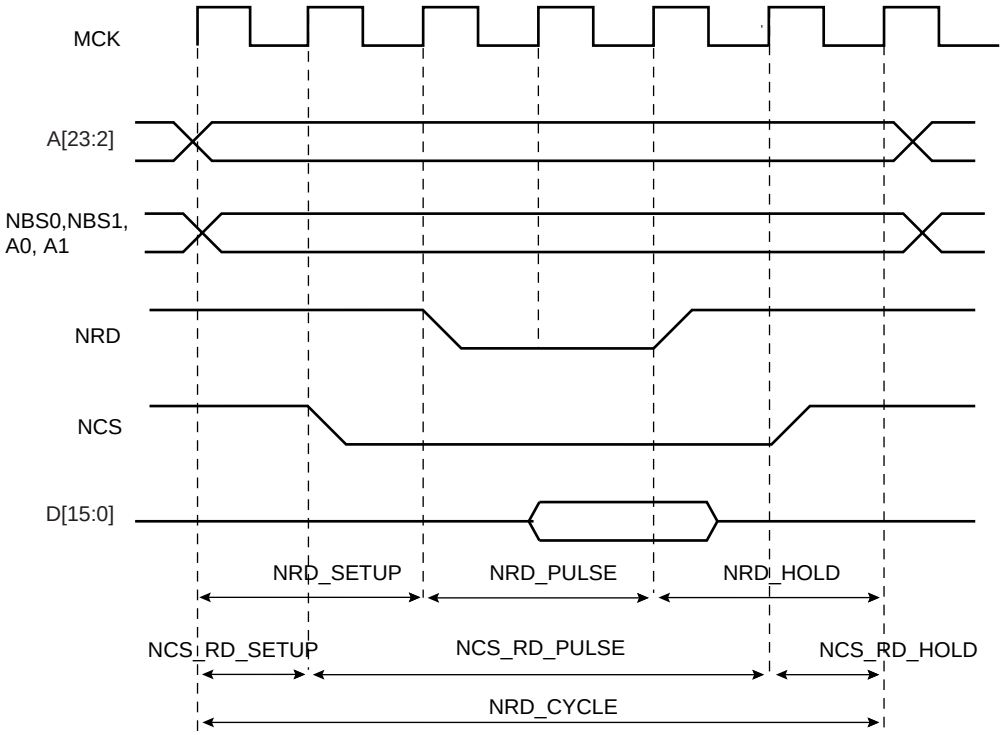
The read cycle is shown on Figure 25-7.

The read cycle starts with the address setting on the memory address bus, i.e.:

{A[23:2], A1, A0} for 8-bit devices

{A[23:2], A1} for 16-bit devices

Figure 25-7. Standard Read Cycle



25.10.1.1 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.

25.10.1.2 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.

25.10.1.3 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:

NRD_CYCLE = NRD_SETUP + NRD_PULSE + NRD_HOLD

= NCS_RD_SETUP + NCS_RD_PULSE + NCS_RD_HOLD

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, the 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:

NRD_HOLD = NRD_CYCLE - NRD SETUP - NRD PULSE

NCS_RD_HOLD = NRD_CYCLE - NCS_RD_SETUP - NCS_RD_PULSE

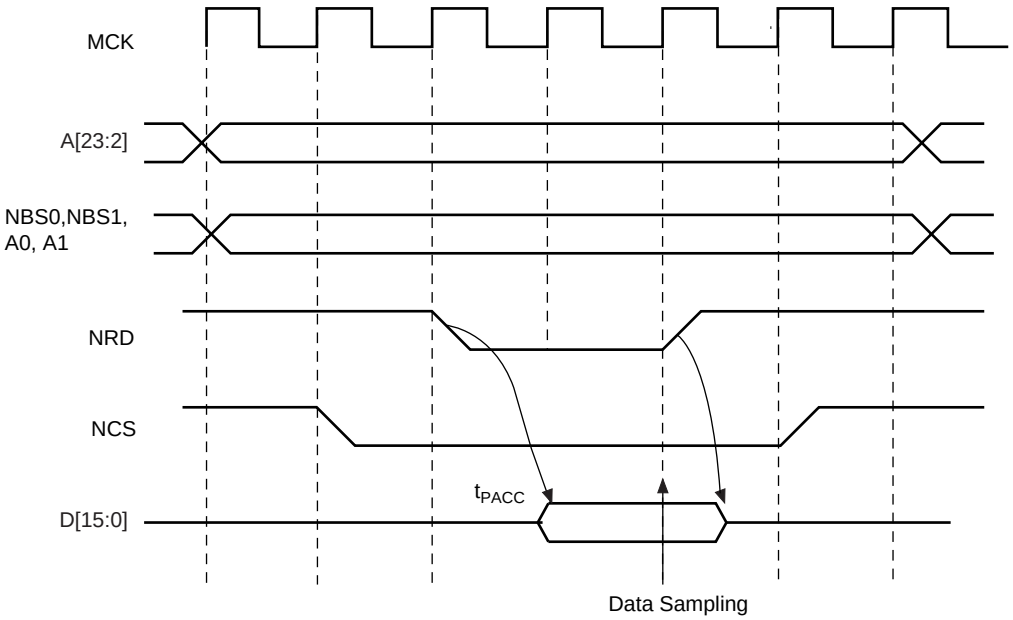
25.10.2 Read Mode

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.

25.10.2.1 Read is Controlled by NRD (READ_MODE = 1):

Figure 25-8 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.

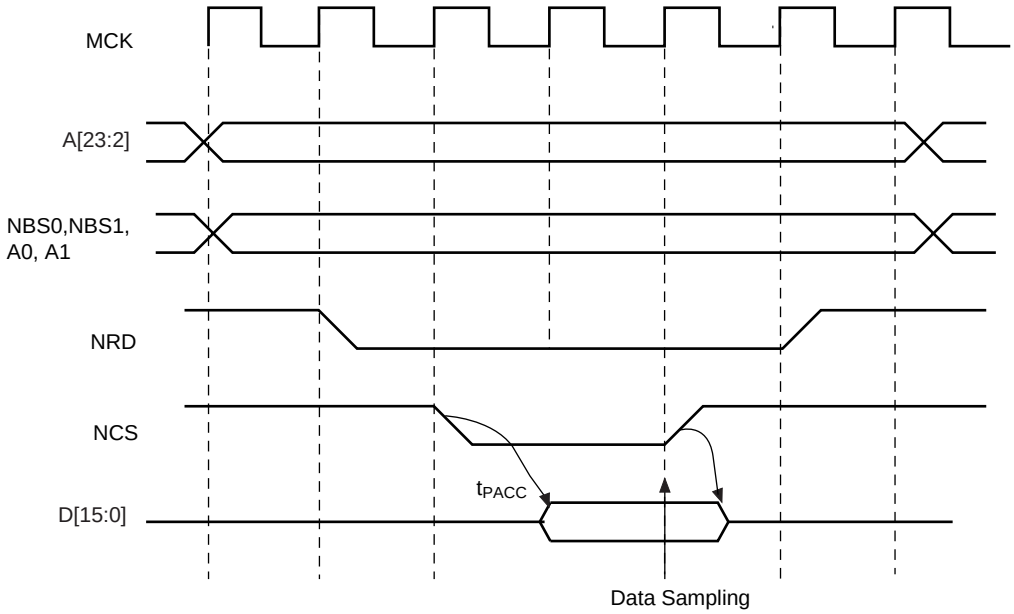
Figure 25-8. READ_MODE = 1: Data is sampled by SMC before the rising edge of NRD



25.10.2.2 Read is Controlled by NCS (READ_MODE = 0)

Figure 25-9 shows the typical read cycle. 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 25-9. READ_MODE = 0: Data is sampled by SMC before the rising edge of NCS



25.10.3 Write Waveforms

The write protocol is similar to the read protocol. It is depicted in Figure 25-10. The write cycle starts with the address setting on the memory address bus.

25.10.3.1 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.

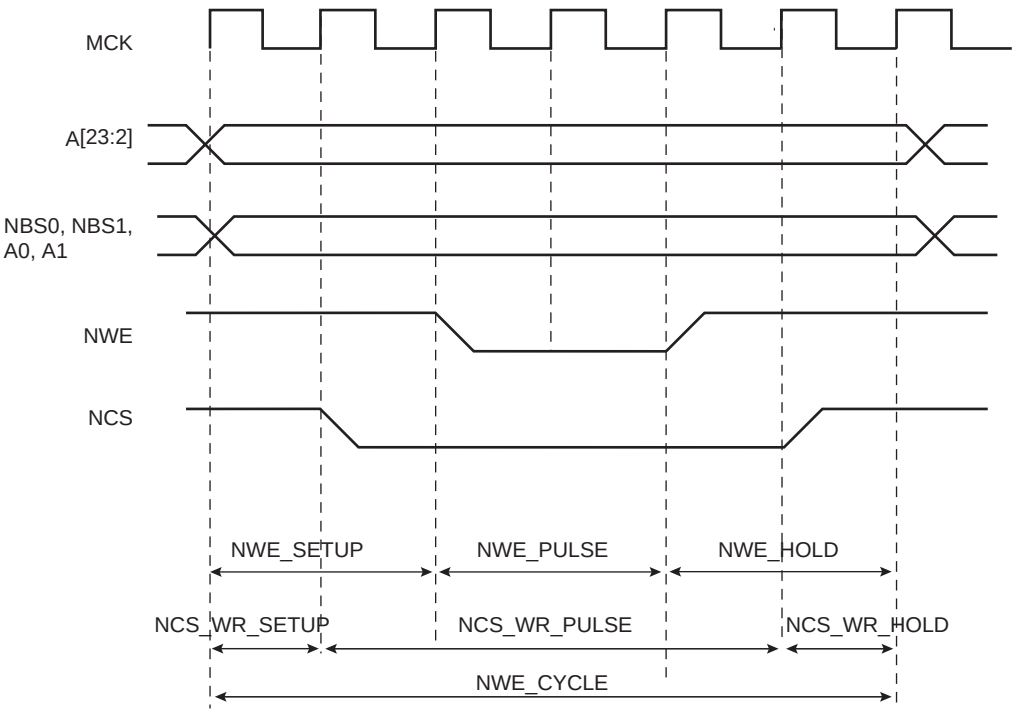
The NWE waveforms apply to all byte-write lines in Byte Write access mode: NWR0 to NWR3.

25.10.3.2 NCS Waveforms

The NCS signal waveforms in write operation are not the same as 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 25-10. Write Cycle



25.10.3.3 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:

NWE_CYCLE = NWE_SETUP + NWE_PULSE + NWE_HOLD

= NCS_WR_SETUP + NCS_WR_PULSE + NCS_WR_HOLD

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:

NWE_HOLD = NWE_CYCLE - NWE_SETUP - NWE_PULSE

NCS_WR_HOLD = NWE_CYCLE - NCS_WR_SETUP - NCS_WR_PULSE

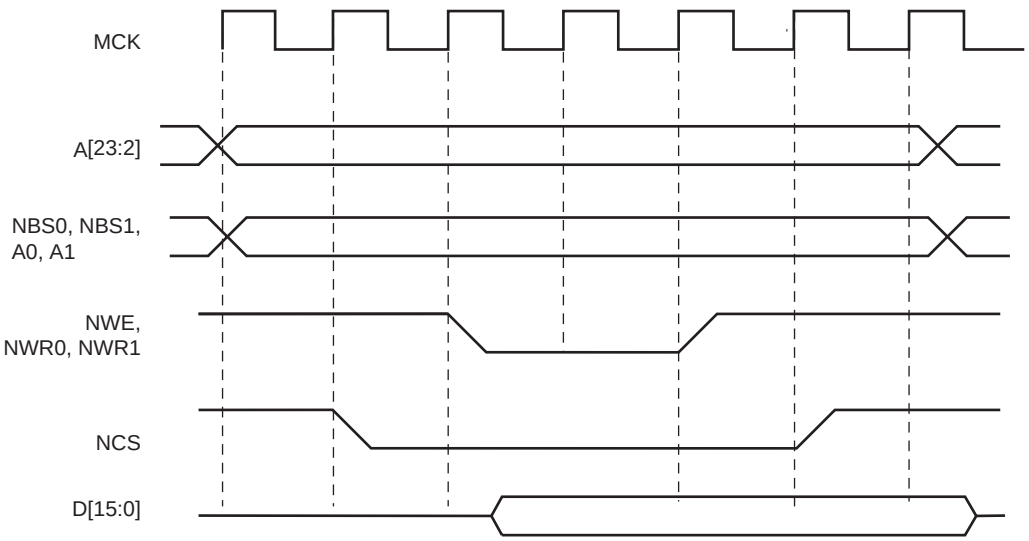
25.10.4 Write Mode

The WRITE_MODE parameter in the SMC_MODE register of the corresponding chip select indicates which signal controls the write operation.

25.10.4.1 Write is Controlled by NWE (WRITE_MODE = 1)

Figure 25-11 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 turned out after the NWE_SETUP time, and until the end of the write cycle, regardless of the programmed waveform on NCS.

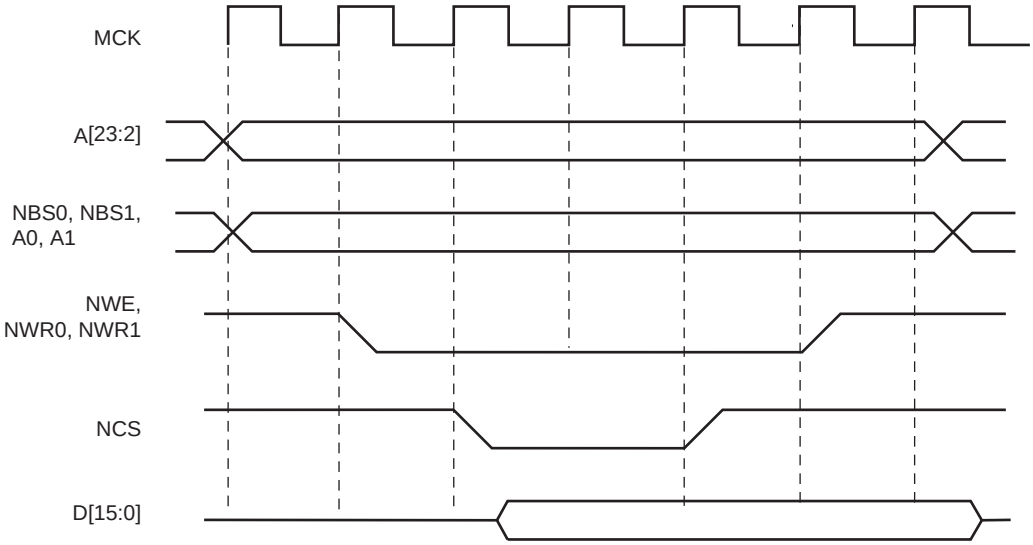
Figure 25-11. WRITE_MODE = 1. The write operation is controlled by NWE



25.10.4.2 Write is Controlled by NCS (WRITE_MODE = 0)

Figure 25-12 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 turned out after the NCS_WR_SETUP time, and until the end of the write cycle, regardless of the programmed waveform on NWE.

Figure 25-12. WRITE_MODE = 0. The write operation is controlled by NCS



25.10.5 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 25-6 shows how the timing parameters are coded and their permitted range.

Table 25-6. 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	128 ≤ ≤ 128+31
pulse [6:0]	7	256 x pulse[6] + pulse[5:0]	0 ≤ ≤ 63	256 ≤ ≤ 256+63
cycle [8:0]	9	256 x cycle[8:7] + cycle[6:0]	0 ≤ ≤ 127	256 ≤ ≤ 256+127 512 ≤ ≤ 512+127 768 ≤ ≤ 768+127

25.10.6 Reset Values of Timing Parameters

Table 25-7 gives the default value of timing parameters at reset.

Table 25-7. Reset Values of Timing Parameters

Register	Reset Value	Description
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

25.10.7 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.

25.10.7.1 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 theses 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.

25.10.7.2 For Write Operations

If a null hold value is programmed on NWE, the SMC can guarantee a positive hold of address, byte select lines, and NCS signal after the rising edge of NWE. This is true for WRITE_MODE = 1 only. See “Early Read Wait State” on page 436.

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.

25.11 Scrambling/Unscrambling Function

The external data bus D[15:0] 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.

One bit is dedicated to enable/disable NAND Flash scrambling and one bit is dedicated enable/disable scrambling the off chip SRAM. When at least one external SRAM is scrambled, the SMSC field must be set in the SMC_OCMS register.

When multiple chip selects (external SRAM) are handled, it is possible to configure the scrambling function per chip select using the OCMS field in the SMC_TIMINGS registers.

To scramble the NAND Flash contents, the SRSE field must be set in the SMC_OCMS register.

When NAND Flash memory content is scrambled, the on-chip SRAM page buffer associated for the transfer is also scrambled.

25.12 Automatic Wait States

Under certain circumstances, the SMC automatically inserts idle cycles between accesses to avoid bus contention or operation conflict.

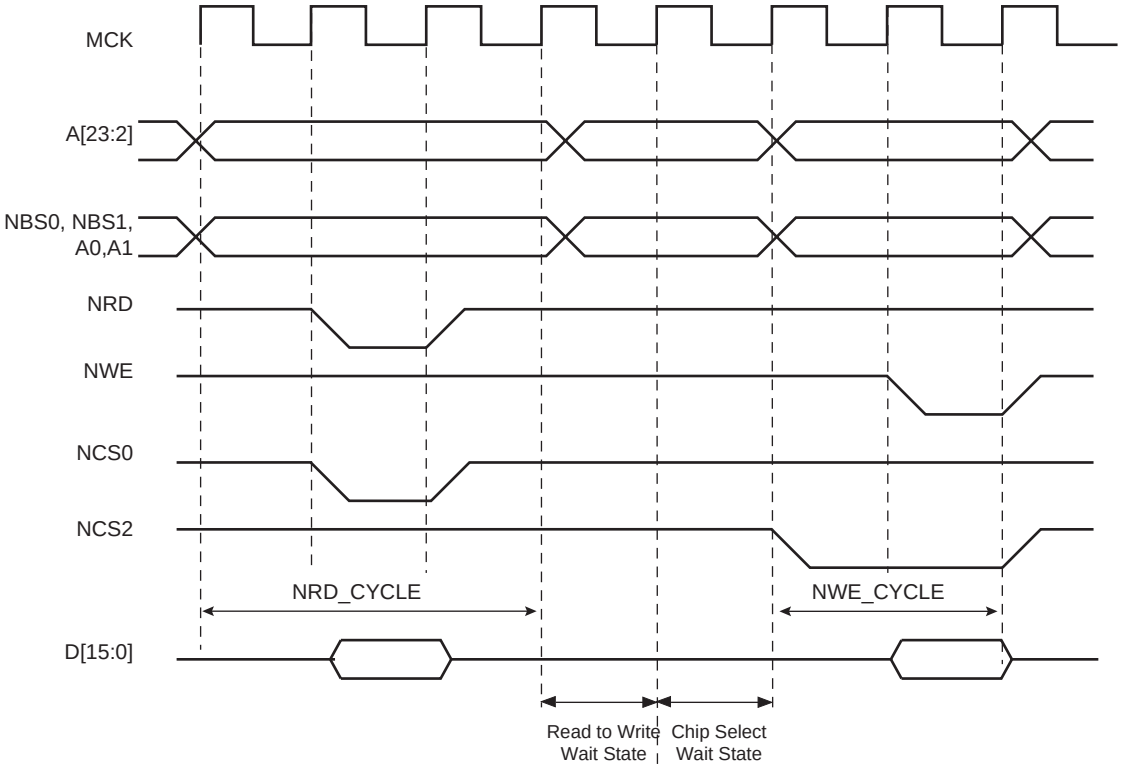
25.12.1 Chip Select Wait States

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: NBS0 to NBS1, NWR0 to NWR1, NCS[0..7], NRD lines are all set to 1.

Figure 25-13 illustrates a chip select wait state between access on Chip Select 0 and Chip Select 2.

Figure 25-13. Chip Select Wait State between a Read Access on NCS0 and a Write Access on NCS2



25.12.2 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 25-14).
- 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 25-15). 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, chip select and byte 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 25-16.

Figure 25-14. Early Read Wait State: Write with No Hold Followed by Read with No Setup

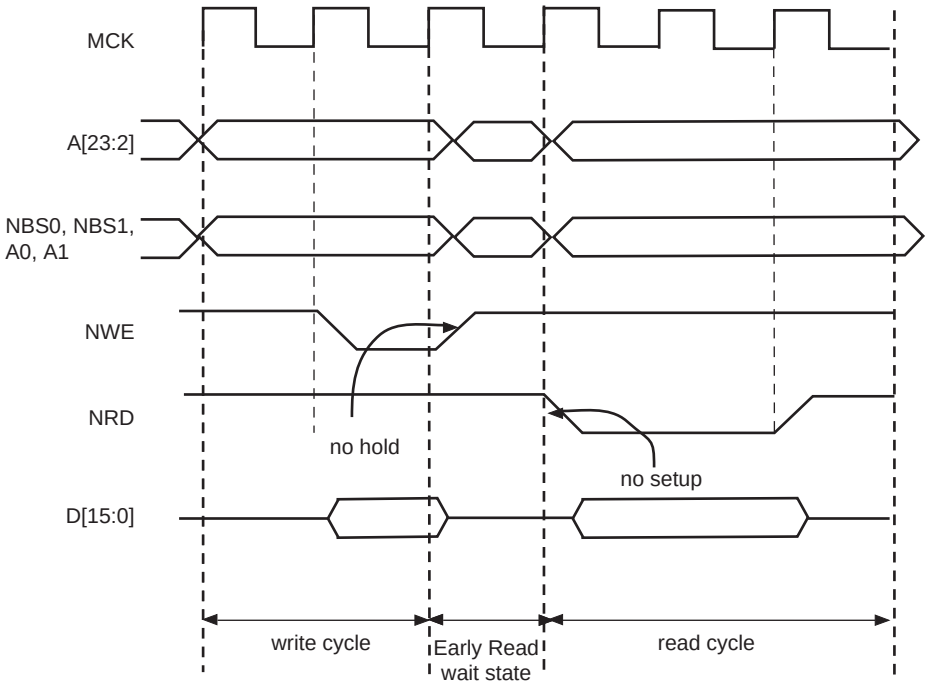


Figure 25-15. Early Read Wait State: NCS Controlled Write with No Hold Followed by a Read with No NCS Setup

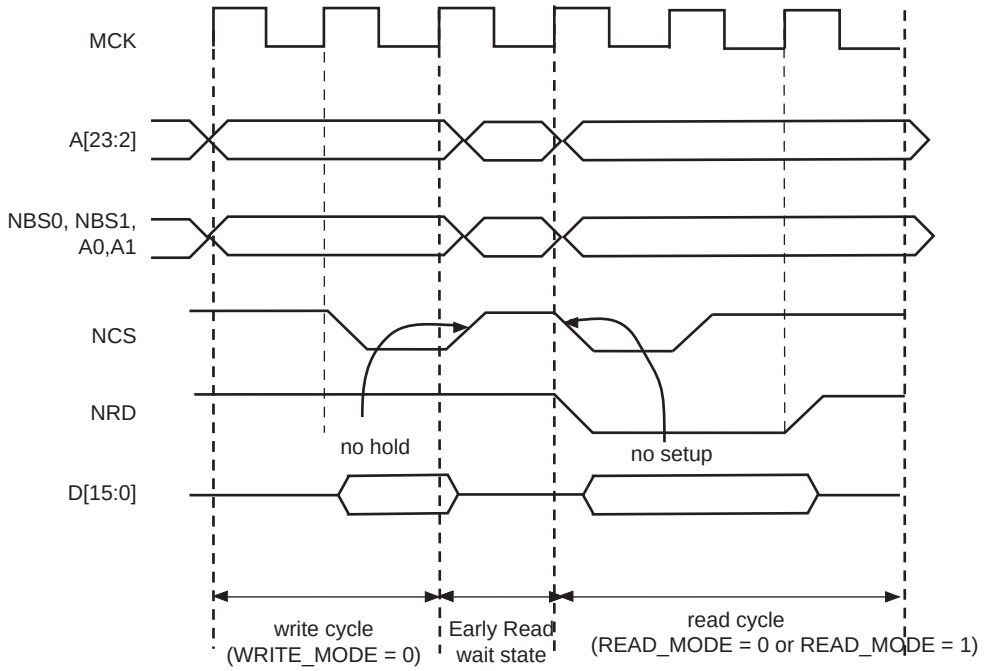
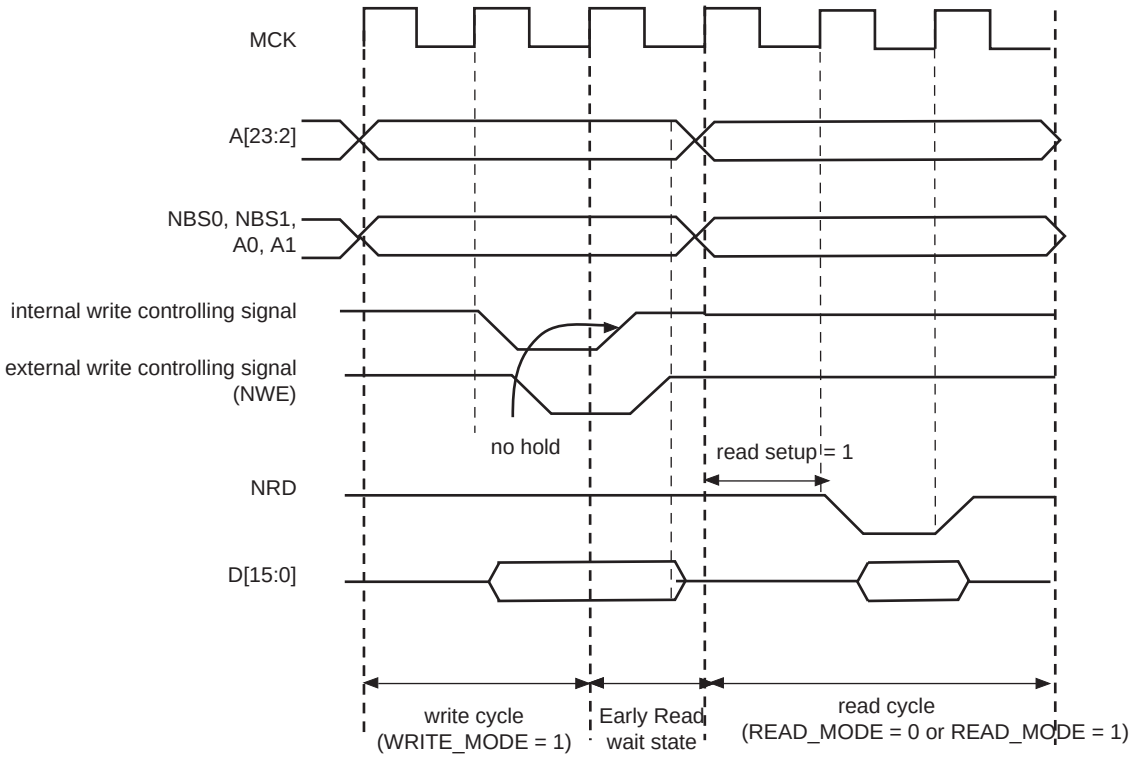


Figure 25-16. Early Read Wait State: NWE-controlled Write with No Hold Followed by a Read with one Set-up Cycle



25.12.3 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.

25.12.3.1 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 only the timing registers are modified (SMC_SETUP, SMC_PULSE, SMC_CYCLE registers) in the user interface, the user must validate the modification by writing the SMC_MODE register, even if no change was made on the mode parameters.

25.12.3.2 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 448).

25.12.4 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 25-13 on page 435.

25.13 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.

25.13.1 READ_MODE

Setting 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 25-17 illustrates the Data Float Period in NRD-controlled mode (READ_MODE =1), assuming a data float period of 2 cycles (TDF_CYCLES = 2). Figure 25-18 shows the read operation when controlled by NCS (READ_MODE = 0) and the TDF_CYCLES parameter equals 3.

Figure 25-17. TDF Period in NRD Controlled Read Access (TDF = 2)

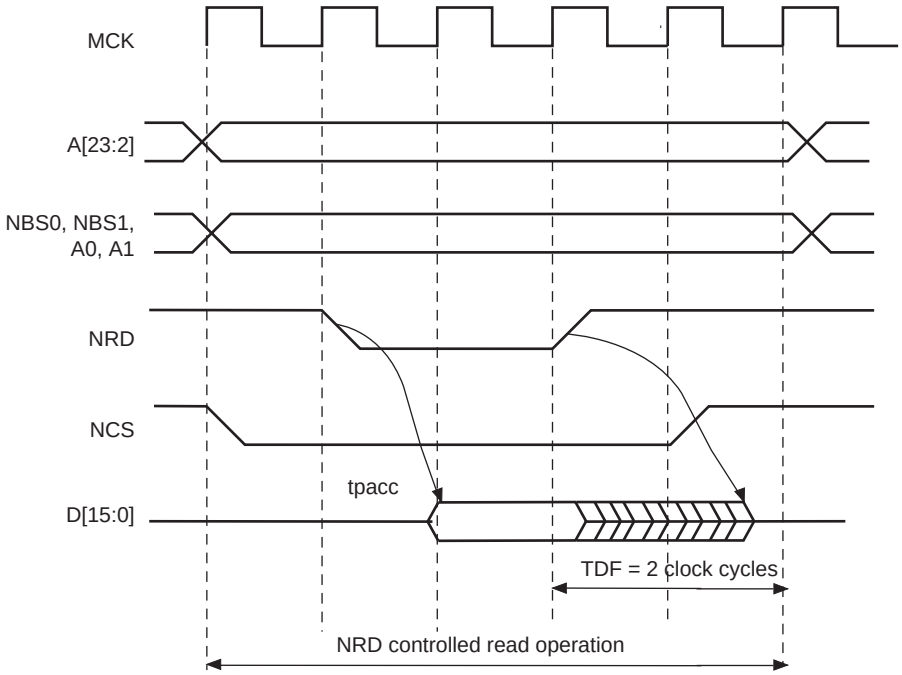
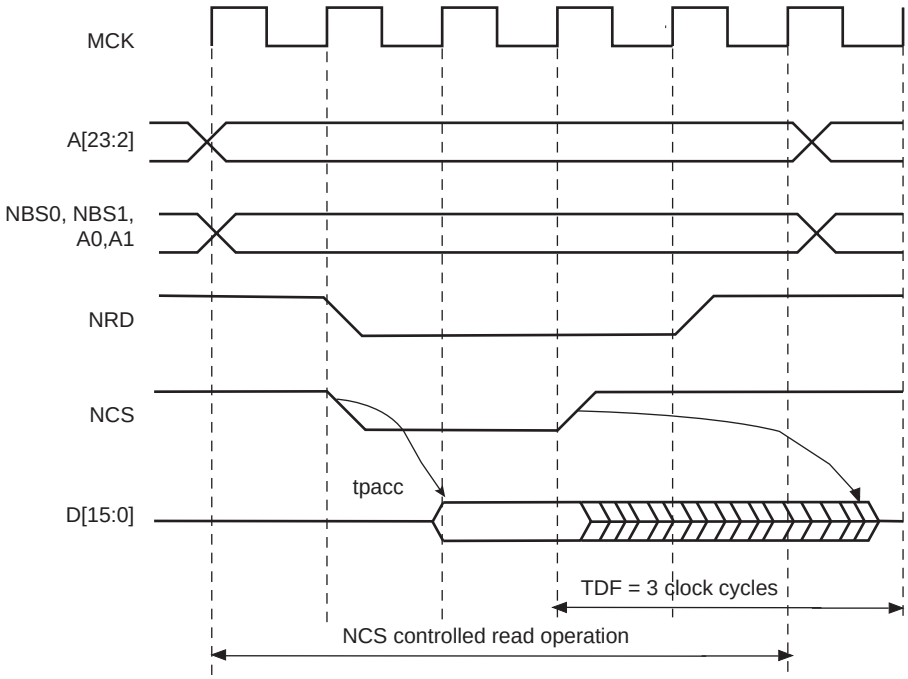


Figure 25-18. TDF Period in NCS Controlled Read Operation (TDF = 3)



25.13.2 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.

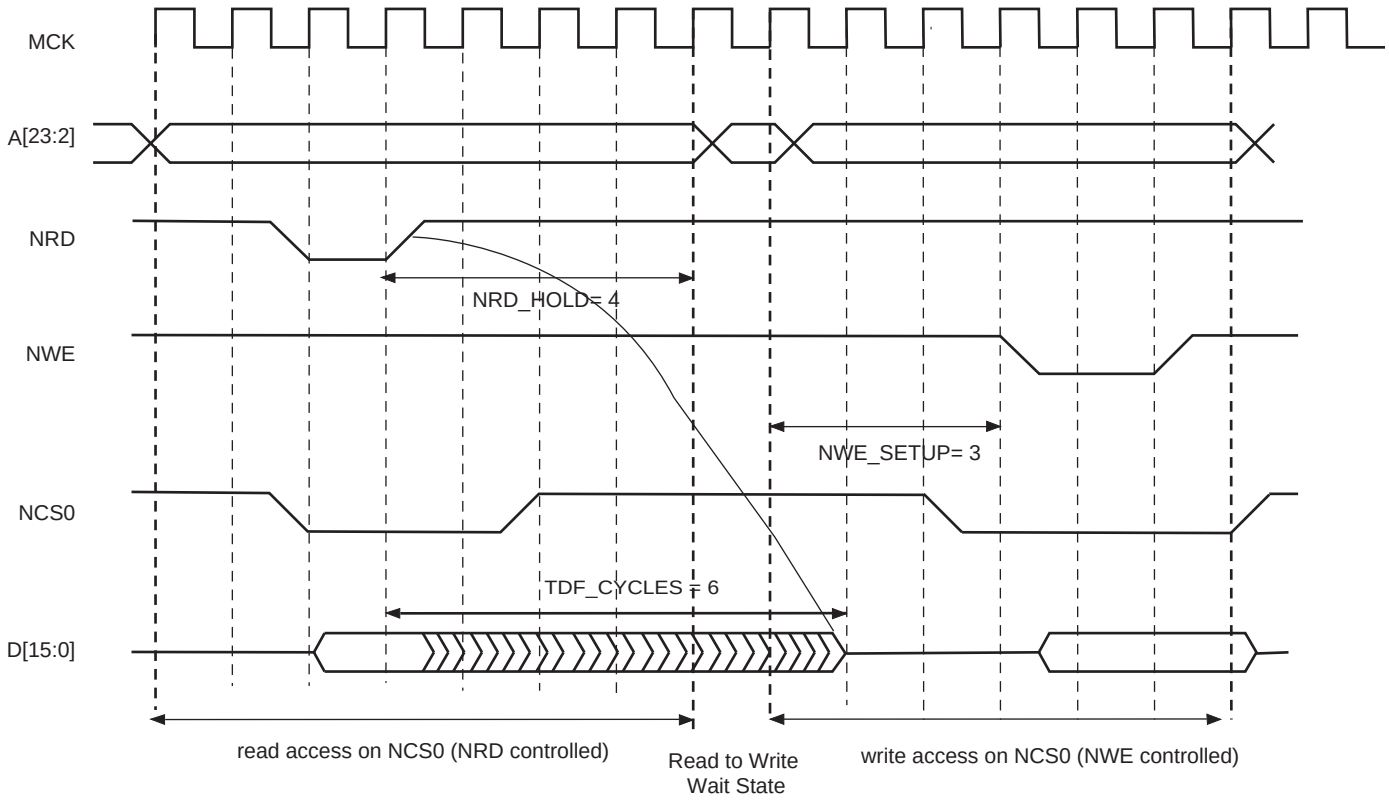
Figure 25-19 shows a read access controlled by NRD, followed by a write access controlled by NWE, on Chip Select 0. Chip Select 0 has been programmed with:

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 25-19. TDF Optimization: No TDF wait states are inserted if the TDF period is over when the next access begins



25.13.3 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 ends 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 25-20, Figure 25-21 and Figure 25-22 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 25-20. TDF Optimization Disabled (TDF Mode = 0). TDF wait states between 2 read accesses on different chip selects

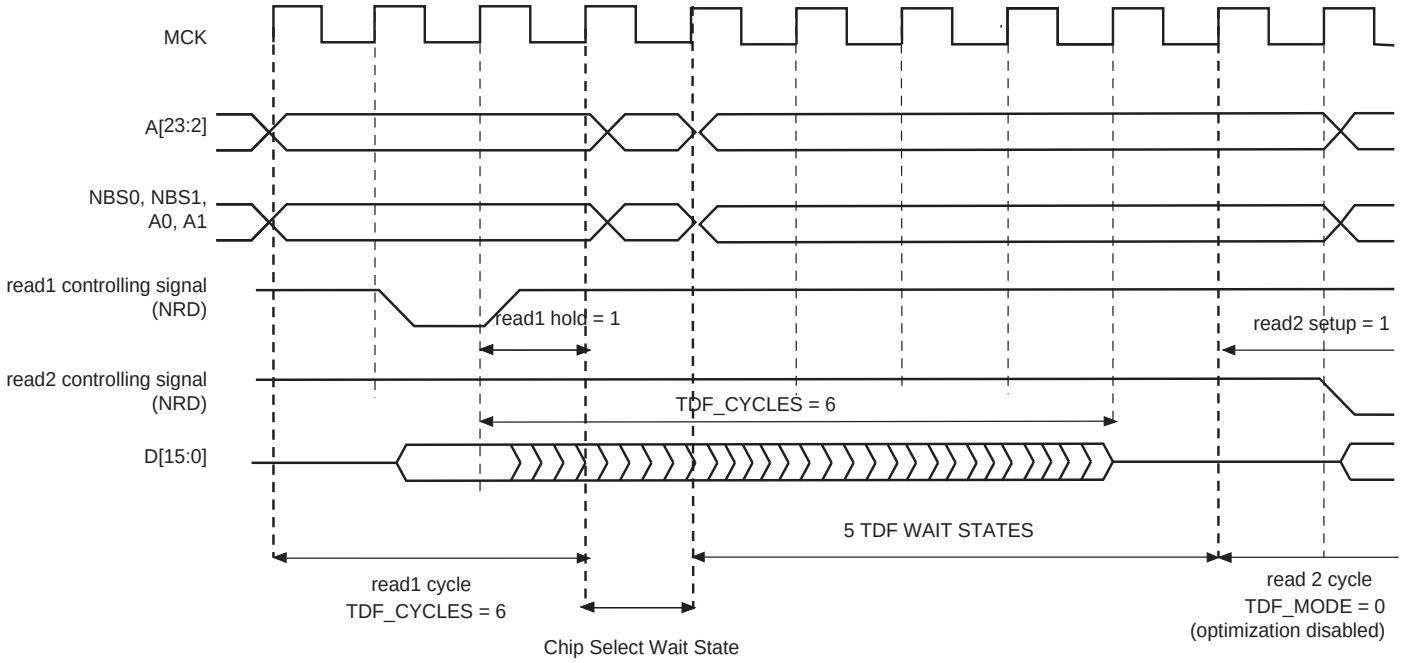


Figure 25-21. TDF Mode = 0: TDF wait states between a read and a write access on different chip selects

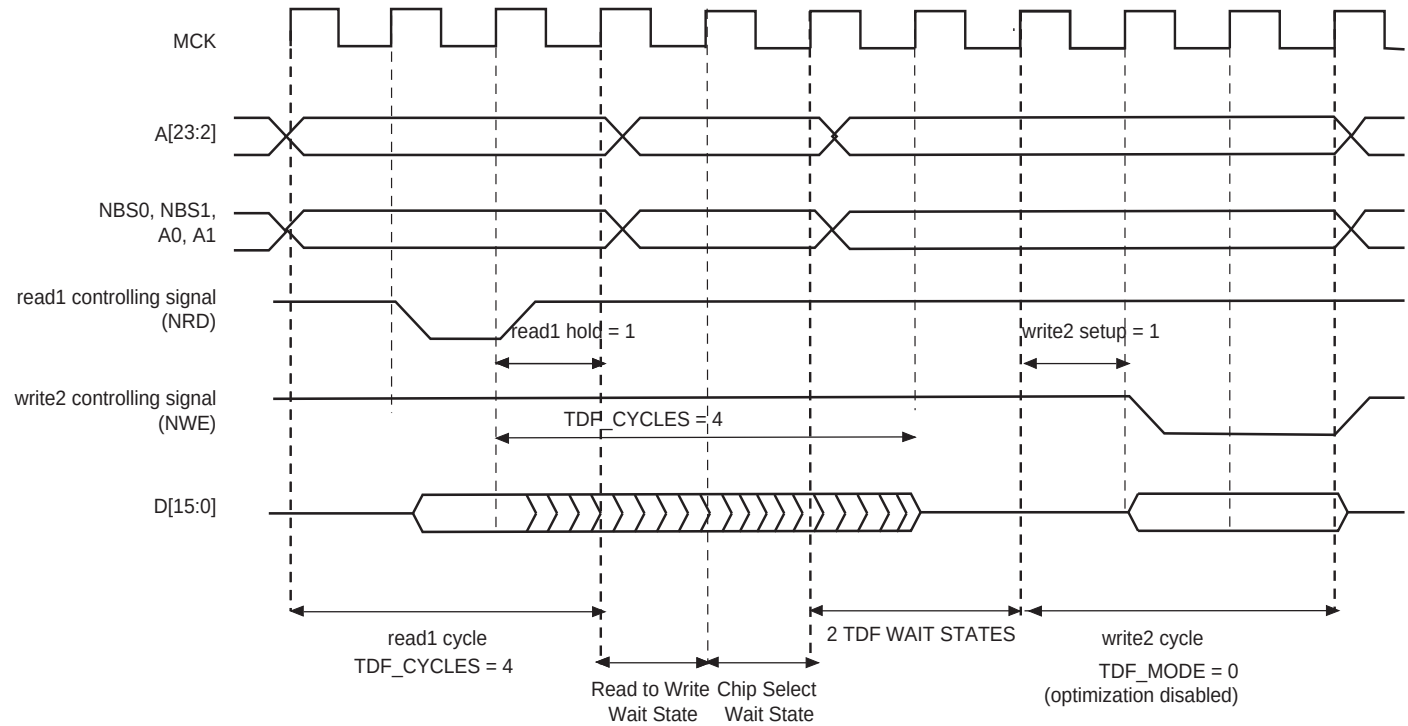
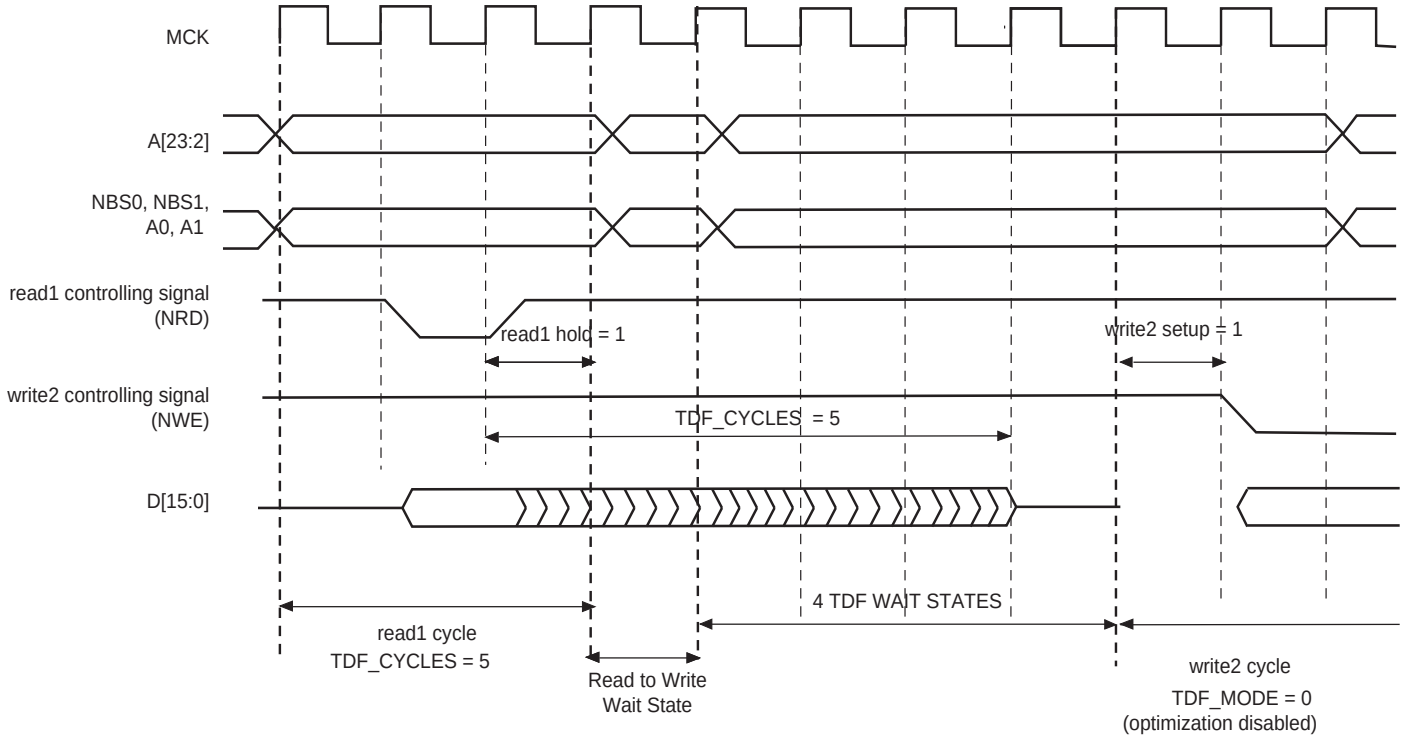


Figure 25-22. TDF Mode = 0: TDF wait states between read and write accesses on the same chip select



25.14 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 to “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.

25.14.1 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 Slow Clock Mode (“Slow Clock Mode” on page 448).

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.

25.14.2 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 25-23. 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 25-24.

Figure 25-23. Write Access with NWAIT Assertion in Frozen Mode (EXNW_MODE = 10)

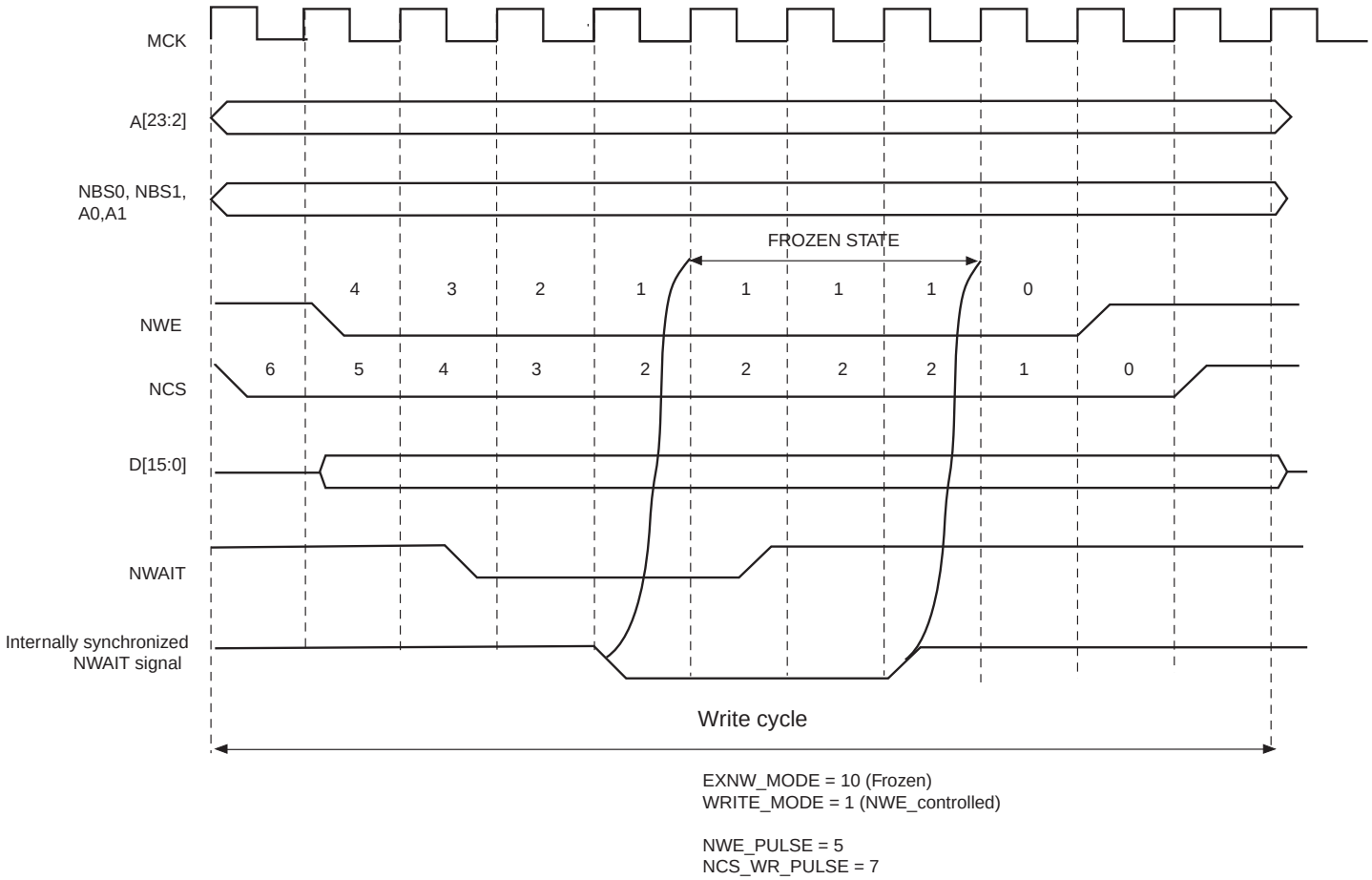
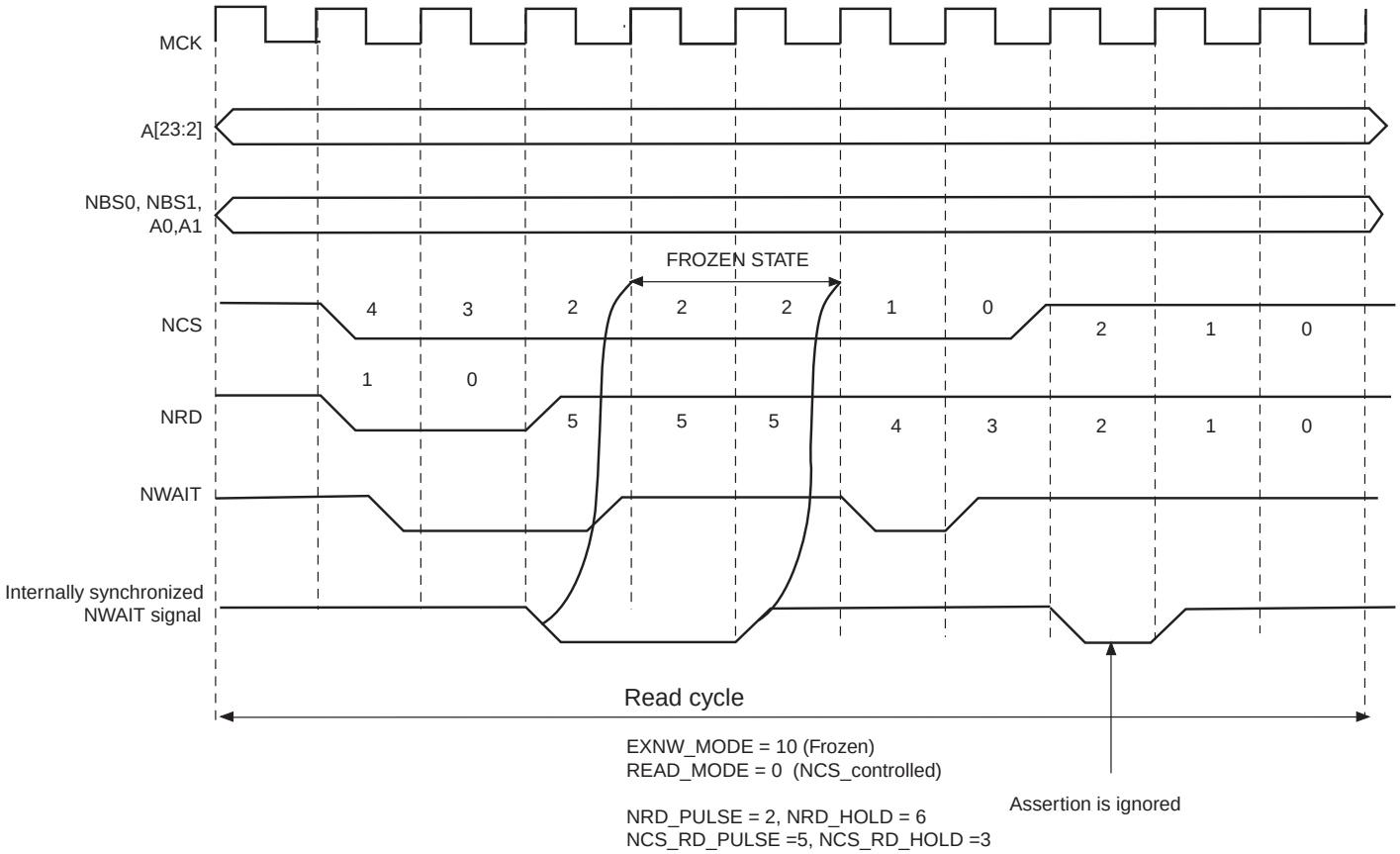


Figure 25-24. Read Access with NWAIT Assertion in Frozen Mode (EXNW_MODE = 10)



25.14.3 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 25-25 and Figure 25-26. 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 25-26.

Figure 25-25. NWAIT Assertion in Write Access: Ready Mode (EXNW_MODE = 11)

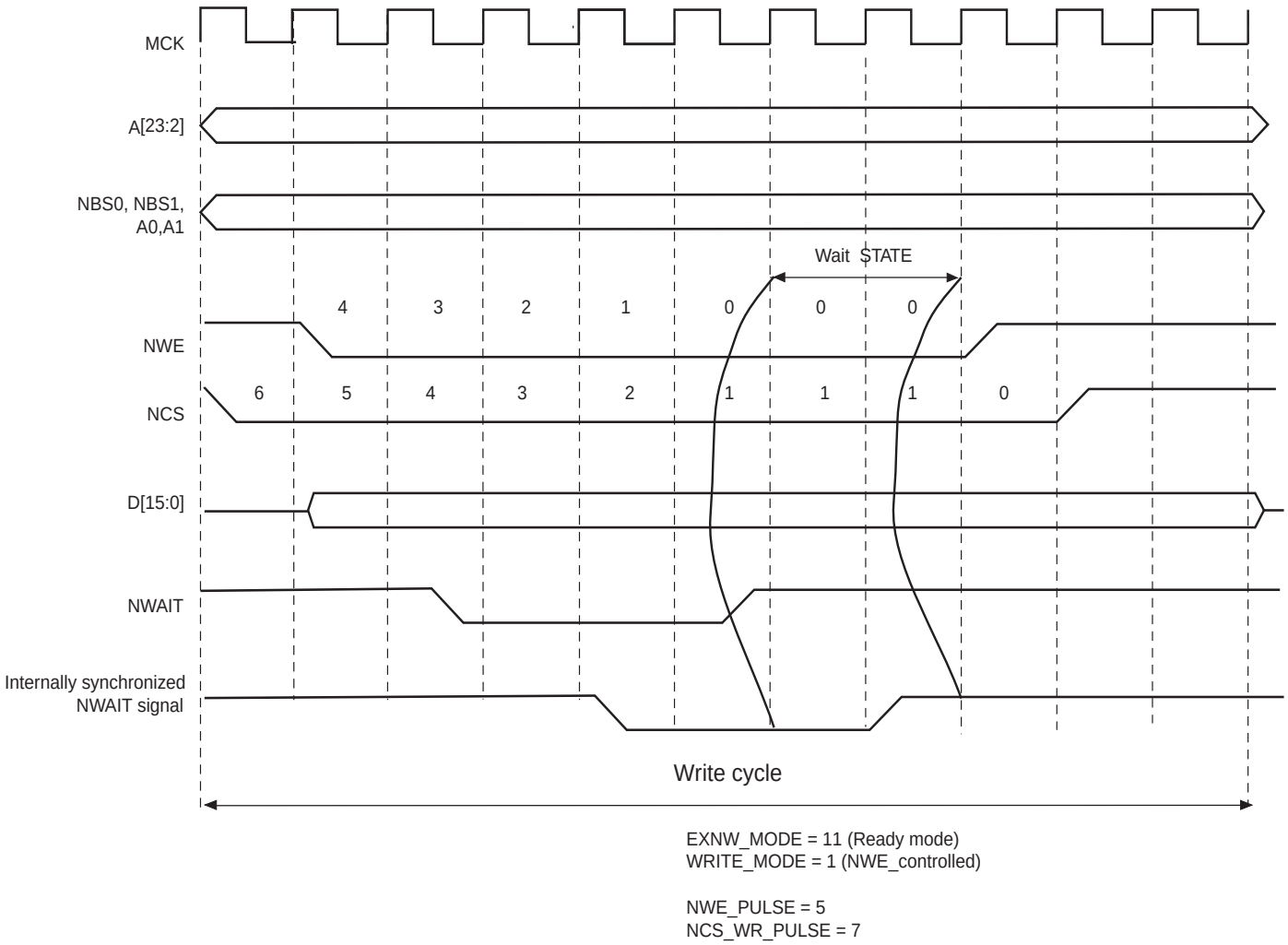
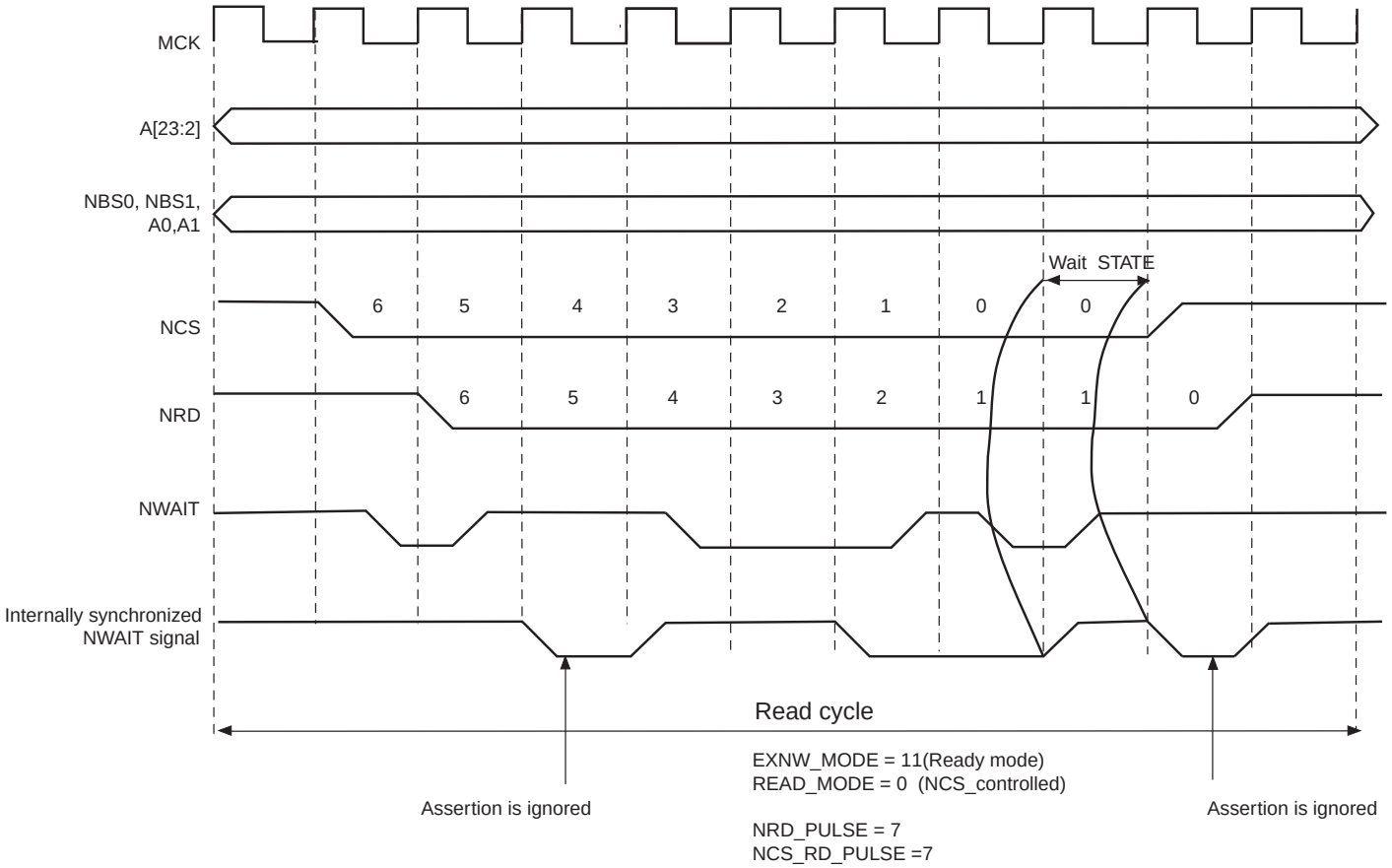


Figure 25-26. NWAIT Assertion in Read Access: Ready Mode (EXNW_MODE = 11)



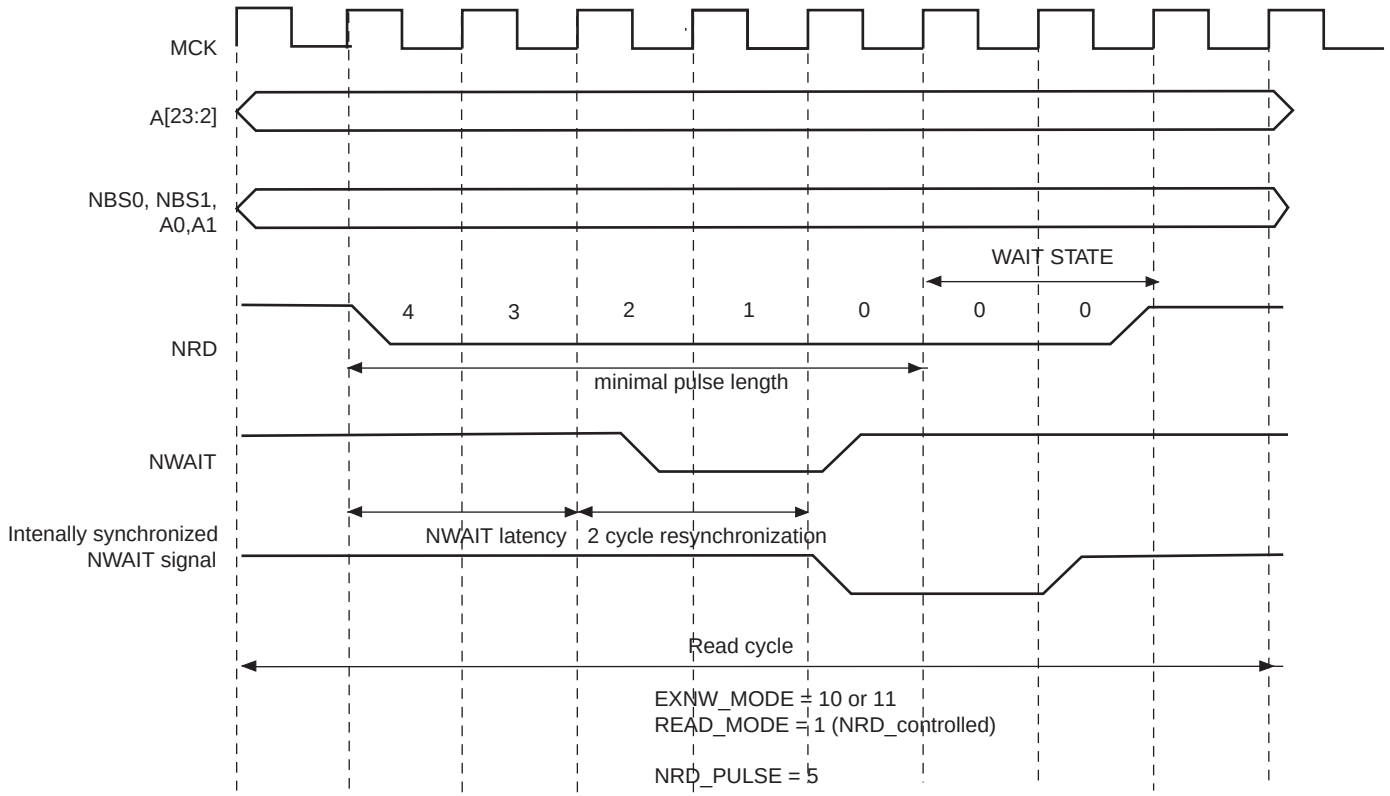
25.14.4 NWAIT Latency and Read/Write Timings

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 25-27.

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:

minimal pulse length = NWAIT latency + 2 resynchronization cycles + 1 cycle

Figure 25-27. NWAIT Latency



25.15 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.

25.15.1 Slow Clock Mode Waveforms

Figure 25-28 illustrates the read and write operations in slow clock mode. They are valid on all chip selects. Table 25-8 indicates the value of read and write parameters in slow clock mode.

Figure 25-28. Read/Write Cycles in Slow Clock Mode

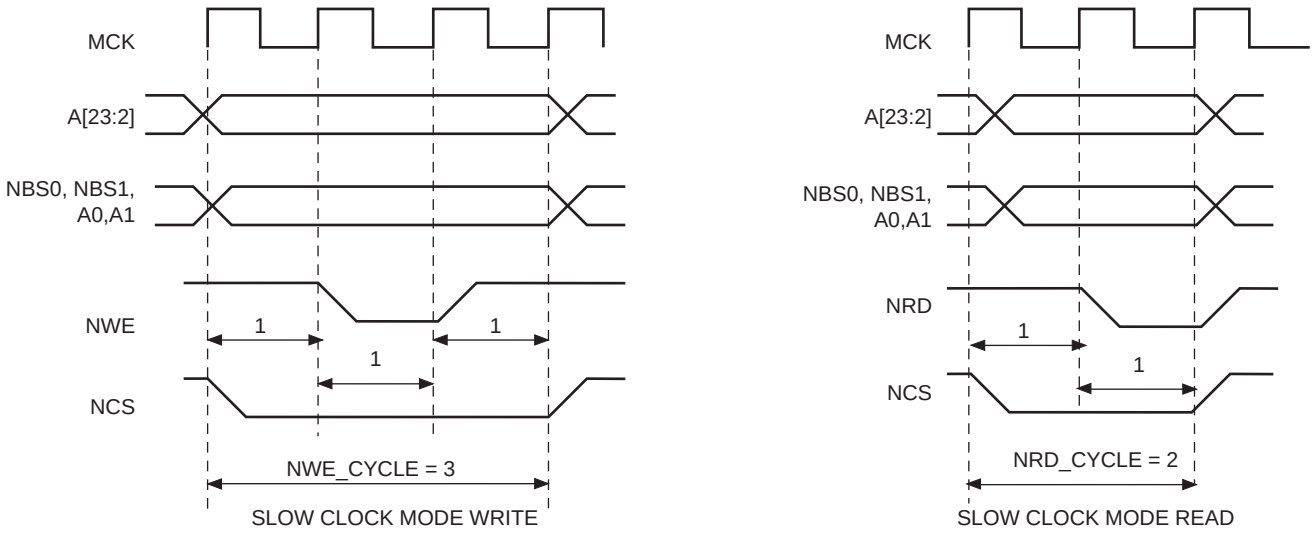


Table 25-8. Read and Write Timing Parameters in Slow Clock Mode

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

25.15.2 Switching from (to) Slow Clock Mode to (from) Normal Mode

When switching from slow clock mode to normal mode, the current slow clock mode transfer is completed at high clock rate, with the set of slow clock mode parameters. See Figure 25-29. The external device may not be fast enough to support such timings.

Figure 25-30 illustrates the recommended procedure to properly switch from one mode to the other.

Figure 25-29. Clock Rate Transition Occurs while the SMC is Performing a Write Operation

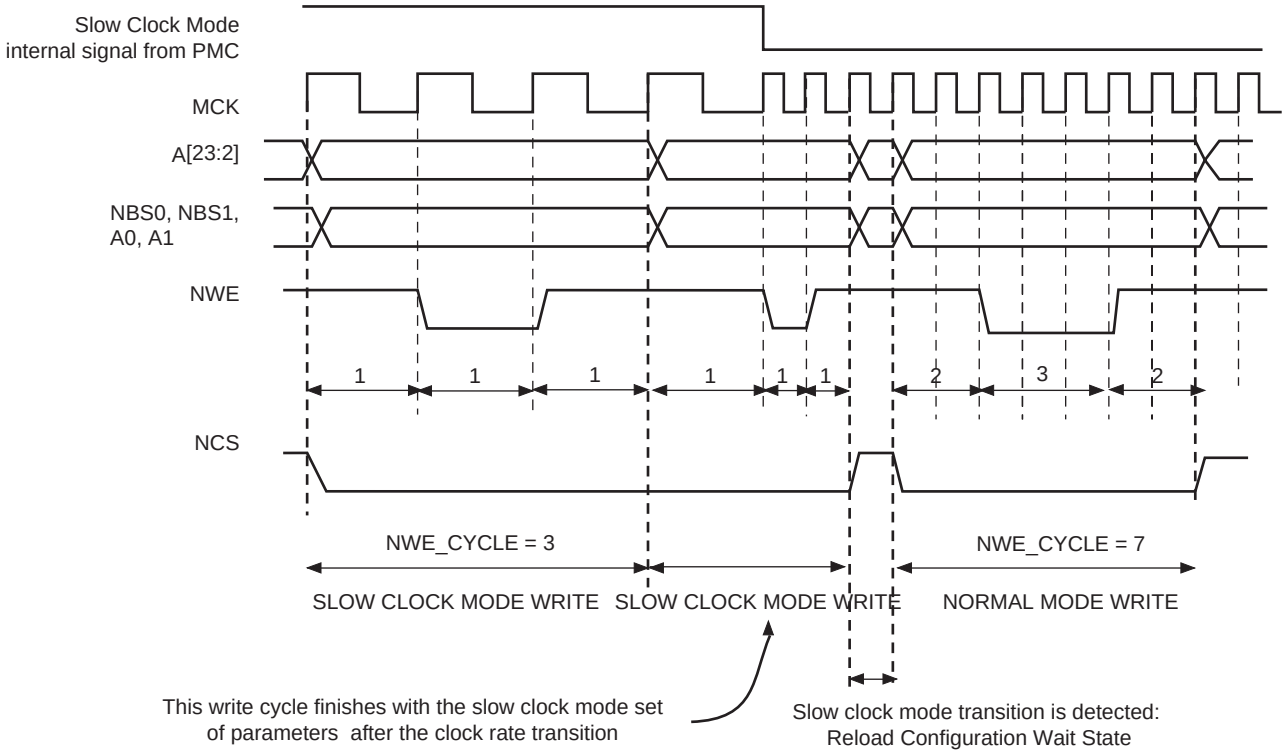
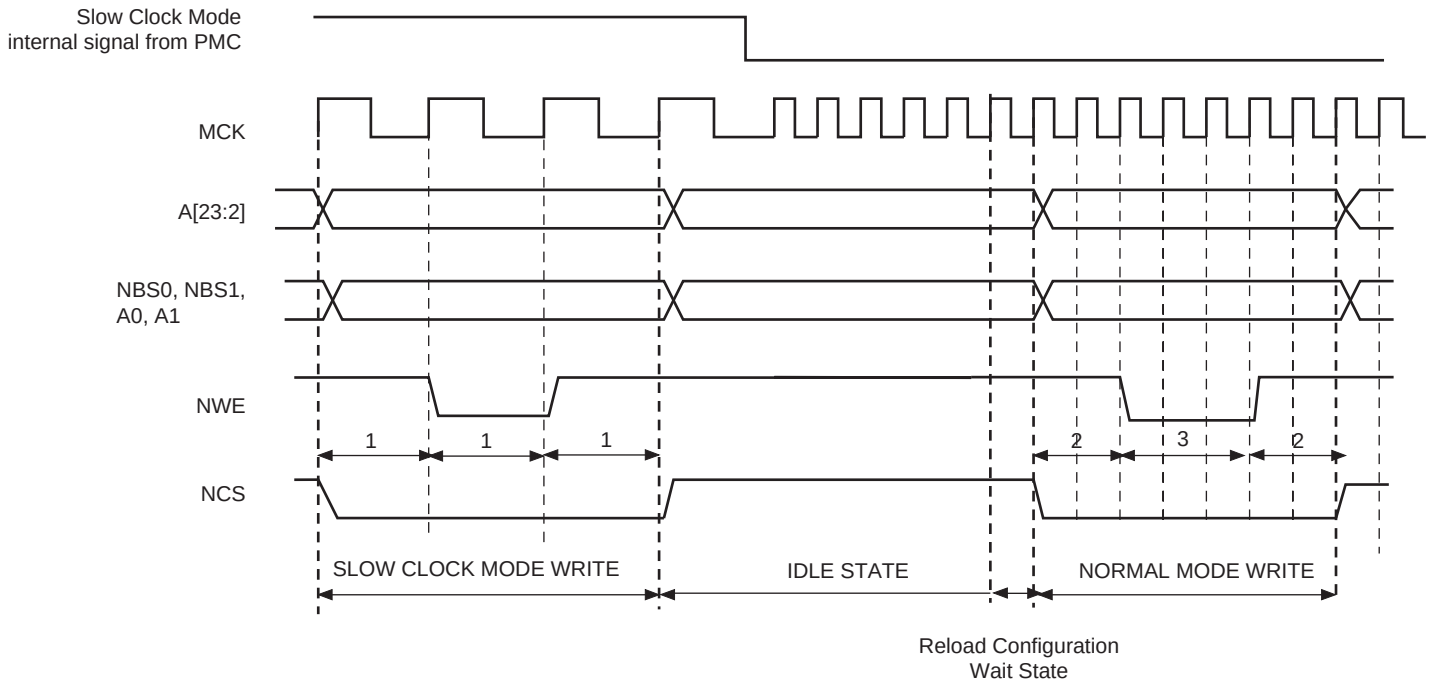


Figure 25-30. Recommended Procedure to Switch from Slow Clock Mode to Normal Mode or from Normal Mode to Slow Clock Mode



25.16 NAND Flash Controller Operations

25.16.1 NFC Overview

The NFC can handle automatic transfers, sending the commands and address to the NAND Flash and transferring the contents of the page (for read and write) to the NFC SRAM. It minimizes the CPU overhead.

25.16.2 NFC Control Registers

NAND Flash Read and NAND Flash Program operations can be performed through the NFC Command Registers. In order to minimize CPU intervention and latency, commands are posted in a command buffer. This buffer provides zero wait state latency. The detailed description of the command encoding scheme is explained below.

The NFC handles automatic transfer between the external NAND Flash and the chip via the NFC SRAM. It is done via NFC Command Registers.

The NFC Command Registers are very efficient to use. When writing to these registers:

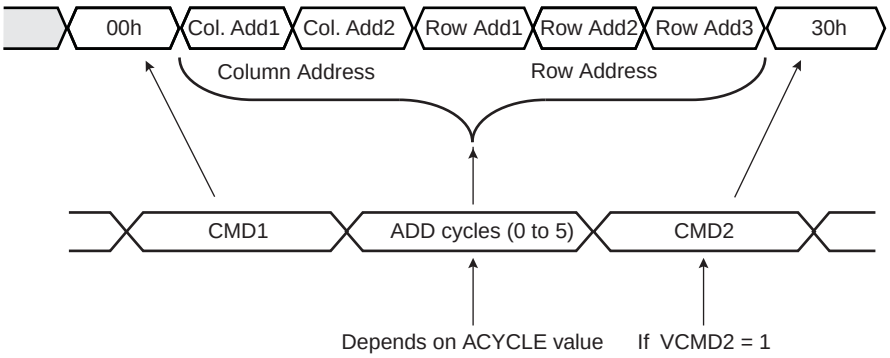
- the address of the register (NFCADDR_CMD) contains the command used,
- the data of the register (NFCDATA_ADDT) contains the address to be sent to the NAND Flash.

So, in one single access the command is sent and immediately executed by the NFC. Even two commands can be programed within a single access (CMD1, CMD2) depending on the VCMD2 value.

The NFC can send up to 5 Address cycles.

Figure 25-31 below shows a typical NAND Flash Page Read Command of a NAND Flash Memory and correspondence with NFC Address Command Register.

Figure 25-31. NFC/NAND Flash Access Example



For more details refer to “NFC Address Command” on page 452.

The NFC Command Registers can be found at address 0x68000000 - 0x6FFFFFFF. (See Table 25-4, “External Memory Mapping”.)

Reading the NFC command register (to any address) will give the status of the NFC. Especially useful to know if the NFC is busy, for example.

25.16.2.1 Building NFC Address Command Example.

The base address is made of address 0x60000000 + NFCCMD bit set = 0x68000000.

Page read operation example:

```
// Build the Address Command (NFCADDR_CMD)
AddressCommand = (0x60000000  |
                  NFCCMD=1 | // NFC Command Enable
                  NFCWR=0   |// NFC Read Data from NAND Flash
                  NFCEN=1    |    // NFC Enable.
                  CSID=1        |        // Chip Select ID = 1
                 ACYCLE= 5 |     // Number of address cycle.
                 VCMD2=1    |               // CMD2 is sent after Address Cycles
                 CMD2=0x30  |               // CMD2 = 30h
                 CMD1=0x0)    // CMD1 = Read Command = 00h

  // Set the Address for Cycle 0
  SMC_ADDR = Col. Add1

  // Write command with the Address Command built above
*AddressCommand = (Col. Add2   |// ADDR_CYCLE1
           Row Add1  | // ADDR_CYCLE2
           Row Add2  |// ADDR_CYCLE3
           Row Add3  )// ADDR_CYCLE4
```

25.16.2.2 NFC Address Command

Name: NFCADDR_CMD

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	NFCCMD	NFCWR	NFCEN	CSID

23	22	21	20	19	18	17	16
CSID		ACYCLE			VCMD2	CMD2	

15	14	13	12	11	10	9	8
CMD2						CMD1	

7	6	5	4	3	2	1	0
CMD1						–	–

- **CMD1: Command Register Value for Cycle 1**

If NFCCMD is set, when a read or write access occurs, the NFC sends this command.

- **CMD2: Command Register Value for Cycle 2**

If NFCCMD and VCMD2 field are set to one, the NFC sends this command after CMD1.

- **VCMD2: Valid Cycle 2 Command**

When set to true, the CMD2 field is issued after addressing cycle.

- **ACYCLE: Number of Address required for the current command**

When ACYCLE field is different from zero, ACYCLE Address cycles are performed after Command Cycle 1. The maximum number of cycles is 5.

- **CSID: Chip Select Identifier**

Chip select used

- **NFCEN: NFC Enable**

When set to true, the NFC will automatically read or write data after the command.

- **NFCWR: NFC Write Enable**

0: The NFC reads data from the NAND Flash.

1: The NFC writes data into the NAND Flash.

- **NFCCMD: NFC Command Enable**

If set to true, CMD indicates that the NFC shall execute the command encoded in the NFCADDR_CMD.

25.16.2.3 NFC Data Address

Name: NFCDATA_ADDT

Access: Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
ADDR_CYCLE4							

23	22	21	20	19	18	17	16
ADDR_CYCLE3							

15	14	13	12	11	10	9	8
ADDR_CYCLE2							

7	6	5	4	3	2	1	0
ADDR_CYCLE1							

- **ADDR_CYCLE1: NAND Flash Array Address Cycle 1**

When less than 5 address cycles are used ADDR_CYCLE1 is the first byte written to NAND Flash

When 5 address cycles are used ADDR_CYCLE1 is the second byte written to NAND Flash

- **ADDR_CYCLE2: NAND Flash Array Address Cycle 2**

When less than 5 address cycles are used ADDR_CYCLE2 is the second byte written to NAND Flash

When 5 address cycles are used ADDR_CYCLE2 is the third byte written to NAND Flash

- **ADDR_CYCLE3: NAND Flash Array Address Cycle 3**

When less than 5 address cycles are used ADDR_CYCLE3 is the third byte written to NAND Flash

When 5 address cycles are used ADDR_CYCLE3 is the fourth byte written to NAND Flash

- **ADDR_CYCLE4: NAND Flash Array Address Cycle 4**

When less than 5 address cycles are used ADDR_CYCLE4 is the fourth byte written to NAND Flash

When 5 address cycles are used ADDR_CYCLE4 is the fifth byte written to NAND Flash

Note that if 5 address cycles are used, the first address cycle is ADDR_CYCLE0. Please refer to SMC_ADDR register.

25.16.2.4 NFC DATA Status

Name: NFCDATA_Status

Access: Read

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	NFCBUSY	NFCWR	NFCEN	CSID

23	22	21	20	19	18	17	16
CSID		ACYCLE			VCMD2	CMD2	

15	14	13	12	11	10	9	8
CMD2						CMD1	

7	6	5	4	3	2	1	0
CMD1						–	–

- **CMD1: Command Register Value for Cycle 1**

When a Read or Write Access occurs, the Physical Memory Interface drives the IO bus with CMD1 field during the Command Latch cycle 1.

- **CMD2: Command Register Value for Cycle 2**

When VCMD2 field is set to true, the Physical Memory Interface drives the IO bus with CMD2 field during the Command Latch cycle 2.

- **VCMD2: Valid Cycle 2 Command**

When set to true, the CMD2 field is issued after addressing cycle.

- **ACYCLE: Number of Address required for the current command**

When ACYCLE field is different from zero, ACYCLE Address cycles are performed after Command Cycle 1.

- **CSID: Chip Select Identifier**

Chip select used

- **NFCEN: NFC Enable**

When set to true, The NFC is enabled.

- **NFCWR: NFC Write Enable**

0: The NFC is in read mode.

1: The NFC is in write mode.

- **NFCBUSY: NFC Busy Status Flag**

If set to true, it indicates that the NFC is busy.

25.16.3 NFC Initialization

Prior to any Command and Data Transfer, the SMC User Interface must be configured to meet the device timing requirements.

- Write enable Configuration

Use NWE_SETUP, NWE_PULSE and NWE_CYCLE to define the write enable waveform according to datasheet of the device.

Use TADL field in the SMC_TIMINGS register to configure the timing between the last address latch cycle and the first rising edge of WEN for data input.

Figure 25-32. Write Enable Timing Configuration

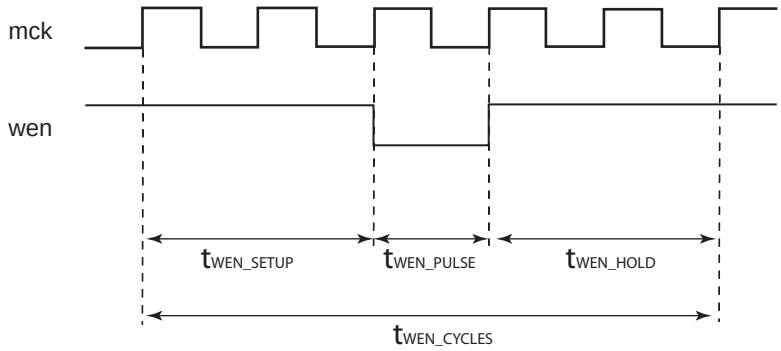
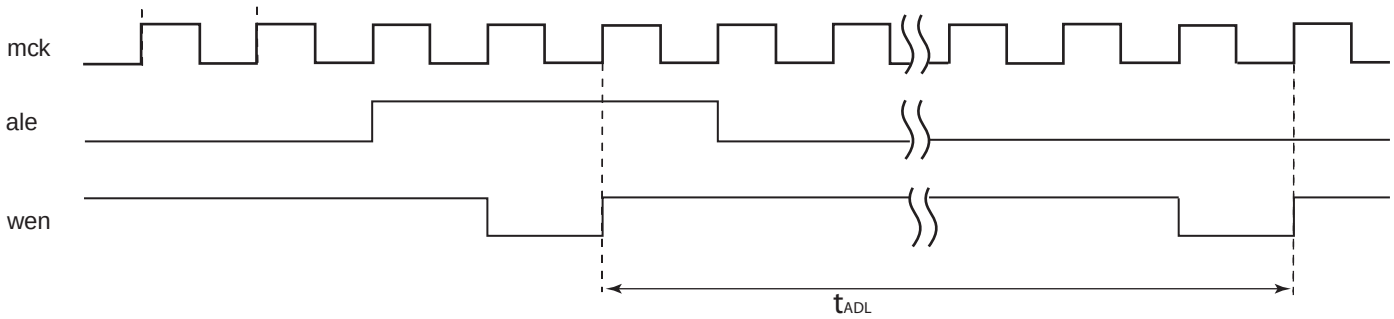


Figure 25-33. Write Enable Timing for NAND Flash Device Data Input Mode.



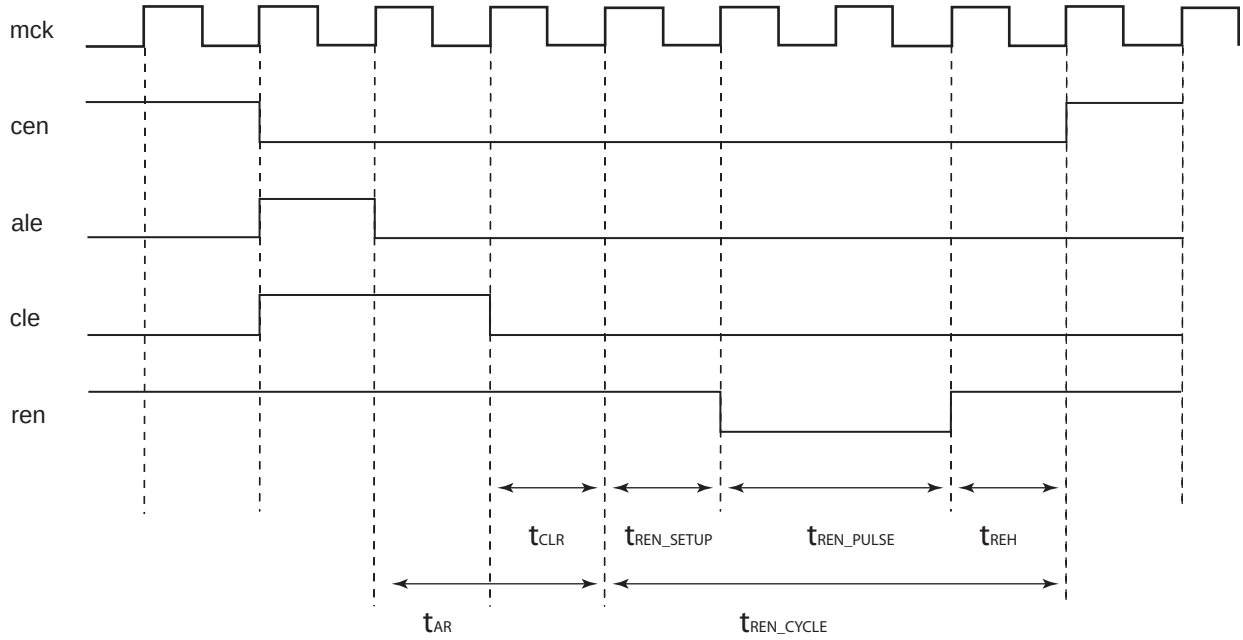
- Read Enable Configuration

Use NRD_SETUP, NRD_PULSE and NRD_CYCLE to define the read enable waveform according to the datasheet of the device.

Use TAR field in the SMC_TIMINGS register to configure the timings between address latch enable falling edge to read enable falling edge.

Use TCLR field in the SMC_TIMINGS register to configure the timings between the command latch enable falling edge to the read enable falling edge.

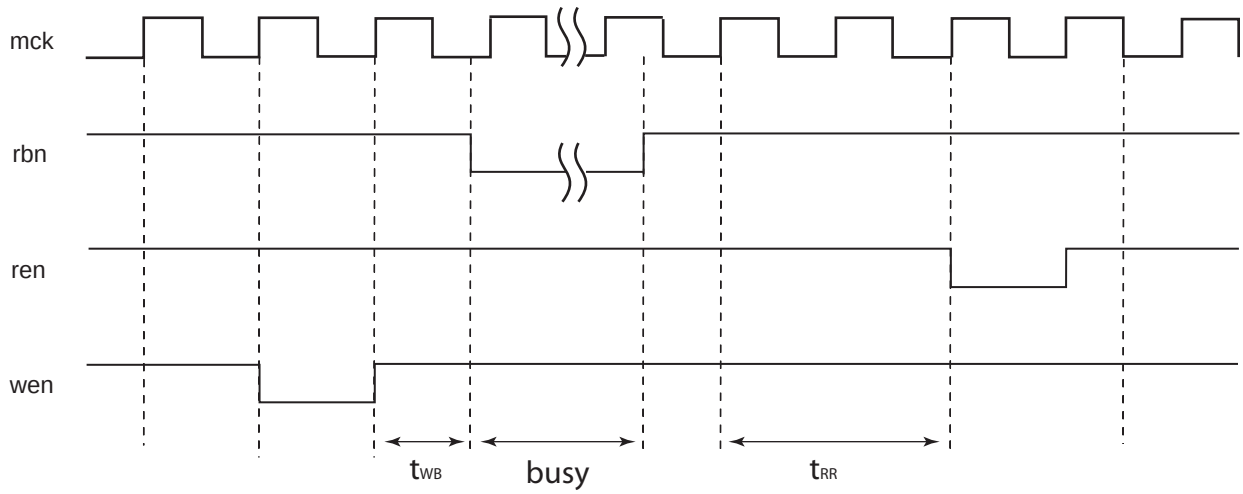
Figure 25-34. Read Enable Timing Configuration Working with NAND Flash Device



- Ready/Busy Signal Timing configuration working with a NAND Flash device

Use TWB field in SMC_TIMINGS register to configure the maximum elapsed time between the rising edge of wen signal and the falling edge of rbn signal. Use TRR field in the SMC_TIMINGS register to program the number of clock cycle between the rising edge of the rbn signal and the falling edge of ren signal.

Figure 25-35. Ready/Busy Timing Configuration

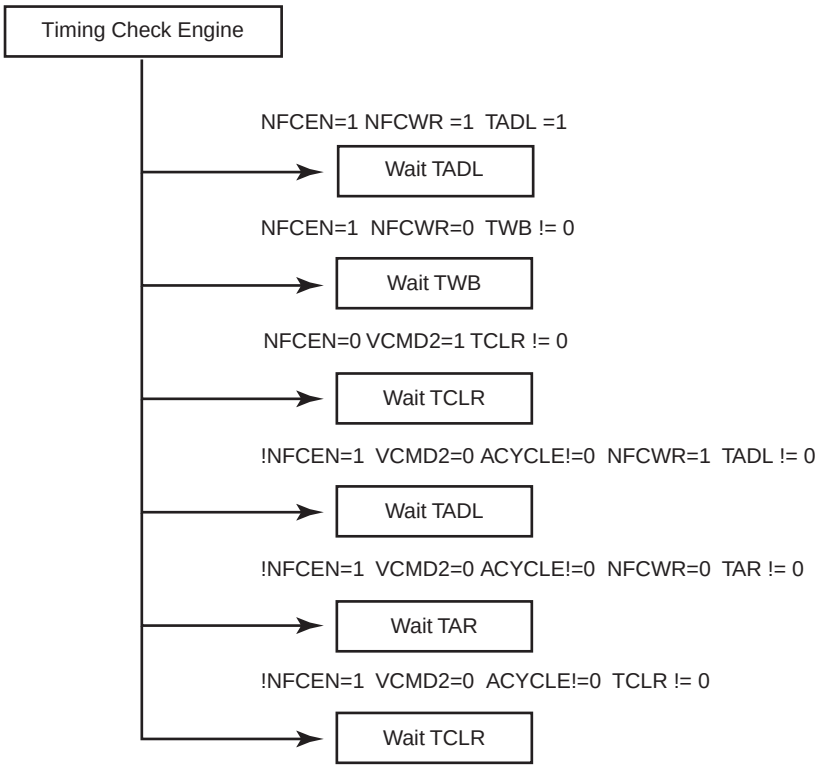


25.16.3.1 NAND Flash Controller Timing Engine

When the NFC Command register is written, the NFC issues a NAND Flash Command and optionally performs a data transfer between the NFC SRAM and the NAND Flash device. The NAND Flash Controller Timing Engine guarantees valid NAND Flash timings, depending on the set of parameters decoded from the address bus. These timings are defined in the SMC_TIMINGS register.

For information of the timing used depending on the command, see Figure 25-36:

Figure 25-36. NAND Flash Controller Timing Engine



See "NFC Address Command" register description and "SMC Timings Register".

25.16.4 NFC SRAM

25.16.4.1 NFC SRAM Mapping

If the NFC is used to read and write Data from and to the NAND Flash, the configuration depends on the page size. See Table 25-9and Table 25-10 for detailed mapping.

The NFC SRAM size is 4 Kbytes. The NFC can handle NAND Flash with a page size of 4 Kbytes or of course lower size (such as 2 Kbytes for example). In the case of 2 Kbytes or lower page size, the NFC SRAM can be split into several banks. The SMC_BANK field enables to select the bank used.

Note that a “ping-pong” mode (write or read to a bank while the NFC writes or reads to another bank) is not accessible with the NFC (using 2 different banks).

If the NFC is not used, the NFC SRAM can be used as general purpose by the application.

Table 25-9. NFC SRAM Mapping with NAND Flash Page Size of 2 Kbytes + 64 bytes

Offset	Use	Access
0x00000000-0x000001FF	Bank 0 Main Area Buffer 0	Read-write
0x00000200-0x000003FF	Bank 0 Main Area Buffer 1	Read-write
0x00000400-0x000005FF	Bank 0 Main Area Buffer 2	Read-write
0x00000600-0x000007FF	Bank 0 Main Area Buffer 3	Read-write
0x00000800-0x0000080F	Bank 0 Spare Area 0	Read-write
0x00000810-0x0000081F	Bank 0 Spare Area 1	Read-write
0x00000820-0x0000082F	Bank 0 Spare Area 2	Read-write
0x00000830-0x0000083F	Bank 0 Spare Area 3	Read-write
0x00000840-0x00000A3F	Bank 1 Main Area Buffer 0	Read-write
0x00000A40-0x00000C3F	Bank 1 Main Area Buffer 1	Read-write
0x00000C40-0x00000E3F	Bank 1 Main Area Buffer 2	Read-write
0x00000E40-0x0000103F	Bank 1 Main Area Buffer 3	Read-write
0x00001040-0x0000104F	Bank 1 Spare Area 0	Read-write
0x00001050-0x0000105F	Bank 1 Spare Area 1	Read-write
0x00001060-0x0000106F	Bank 1 Spare Area 2	Read-write
0x00001070-0x0000107F	Bank 1 Spare Area 3	Read-write
0x00001080-0x00001FFF	Reserved	–

Table 25-10. NFC SRAM Mapping with NAND Flash Page Size of 4 Kbytes + 128 bytes

Offset	Use	Access
0x00000000-0x000001FF	Bank 0 Main Area Buffer 0	Read-write
0x00000200-0x000003FF	Bank 0 Main Area Buffer 1	Read-write
0x00000400-0x000005FF	Bank 0 Main Area Buffer 2	Read-write
0x00000600-0x000007FF	Bank 0 Main Area Buffer 3	Read-write
0x00000800-0x000009FF	Bank 0 Main Area Buffer 4	Read-write
0x00000A00-0x00000BFF	Bank 0 Main Area Buffer 5	Read-write
0x00000C00-0x00000DFF	Bank 0 Main Area Buffer 6	Read-write
0x00000E00-0x00000FFF	Bank 0 Main Area Buffer 7	Read-write
0x00001000-0x0000100F	Bank 0 Spare Area 0	Read-write
0x00001010-0x0000101F	Bank 0 Spare Area 1	Read-write
0x00001020-0x0000102F	Bank 0 Spare Area 2	Read-write

Table 25-10. NFC SRAM Mapping with NAND Flash Page Size of 4 Kbytes + 128 bytes

Offset	Use	Access
0x00001030-0x0000103F	Bank 0 Spare Area 3	Read-write
0x00001040-0x0000104F	Bank 0 Spare Area 4	Read-write
0x00001050-0x0000105F	Bank 0 Spare Area 5	Read-write
0x00001060-0x0000106F	Bank 0 Spare Area 6	Read-write
0x00001070-0x0000107F	Bank 0 Spare Area 7	Read-write
0x00001080-0x00001FFF	Reserved	–

25.16.4.2 NFC SRAM Access Prioritization Algorithm

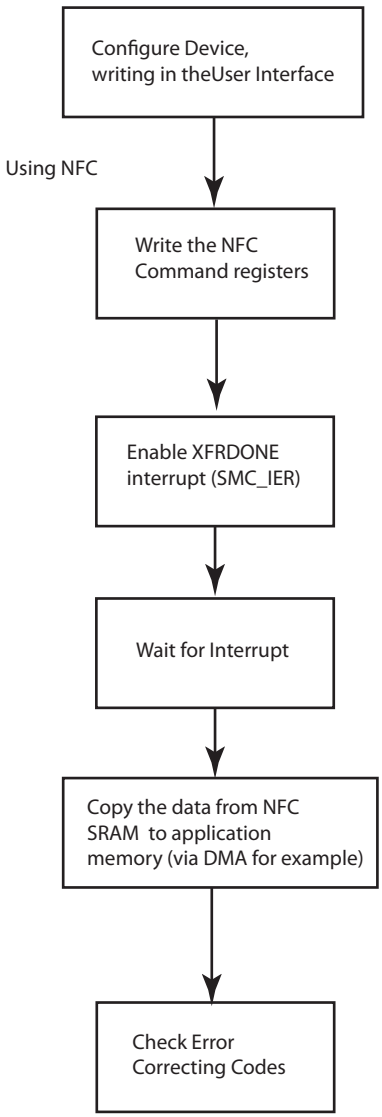
When the NAND Flash Controller (NFC) is reading from or writing to the NFC SRAM, the internal memory is no longer accessible. If an NFC SRAM access occurs when the NFC performs a read or write operation then the access is discarded. The write operation is not performed. The read operation returns undefined data. If this situation is encountered, the status flag AWB located in the NFC status Register is raised and indicates that a shared resource access violation has occurred.

25.16.5 NAND Flash Operations

This section describes the software operations needed to issue commands to the NAND Flash device and perform data transfers using NFC.

25.16.5.1 Page Read

Figure 25-37. Page Read Flow Chart

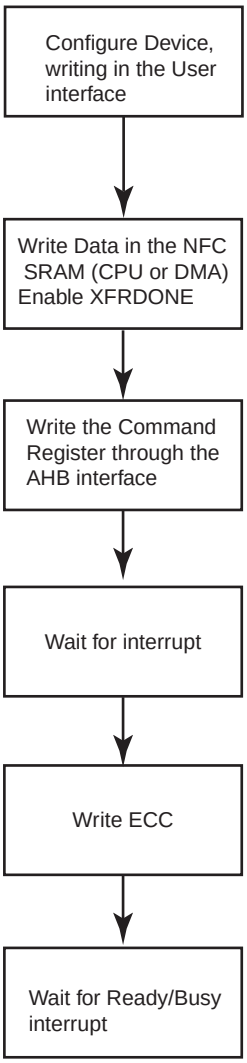


Note that instead of using the interrupt one can poll the NFCBUSY Flag.

For more information on the NFC Control Register, see Section 25.16.2.2 ”NFC Address Command”.

25.16.5.2 Program Page

Figure 25-38. Program Page Flow Chart



Writing the ECC can not be done using the NFC so it needs to be done “manually”.

Note that instead of using the interrupt one can poll the NFCBUSY Flag.

For more information on the NFC Control Register, see Section 25.16.2.2 ”NFC Address Command”.

25.17 SMC Error Correcting Code Functional Description

A page in NAND Flash and SmartMedia memories contains an area for main data and an additional area used for redundancy (ECC). The page is organized in 8-bit or 16-bit words. The page size corresponds to the number of words in the main area plus the number of words in the extra area used for redundancy.

Over time, some memory locations may fail to program or erase properly. In order to ensure that data is stored properly over the life of the NAND Flash device, NAND Flash providers recommend to utilize either 1 ECC per 256 bytes of data, 1 ECC per 512 bytes of data or 1 ECC for all of the page.

The only configurations required for ECC are the NAND Flash or the SmartMedia page size (528/2112/4224) and the type of correction wanted (1 ECC for all the page/1 ECC per 256 bytes of data /1 ECC per 512 bytes of data).

Page size is configured setting the PAGESIZE field in the ECC Mode Register (ECC_MR). Type of correction is configured setting the TYPCORRECT field in the ECC Mode Register (ECC_MR).

Note that there is a limitation when using 16-bit NAND Flash: only 1 ECC for all of page is possible. For 8-bit NAND Flash there is no limitation.

ECC is automatically computed as soon as a read (00h)/write (80h) command to the NAND Flash or the SmartMedia is detected. Read and write access must start at a page boundary.

ECC results are available as soon as the counter reaches the end of the main area. Values in the ECC Parity Registers (ECC_PR0 to ECC_PR15) are then valid and locked until a new start condition occurs (read/write command followed by address cycles).

25.17.1 Write Access

Once the Flash memory page is written, the computed ECC codes are available in the ECC Parity (ECC_PR0 to ECC_PR15) registers. The ECC code values must be written by the software application in the extra area used for redundancy. The number of write accesses in the extra area is a function of the value of the type of correction field. For example, for 1 ECC per 256 bytes of data for a page of 512 bytes, only the values of ECC_PR0 and ECC_PR1 must be written by the software application. Other registers are meaningless.

25.17.2 Read Access

After reading the whole data in the main area, the application must perform read accesses to the extra area where ECC code has been previously stored. Error detection is automatically performed by the ECC controller. Please note that it is mandatory to read consecutively the entire main area and the locations where Parity and NParity values have been previously stored to let the ECC controller perform error detection.

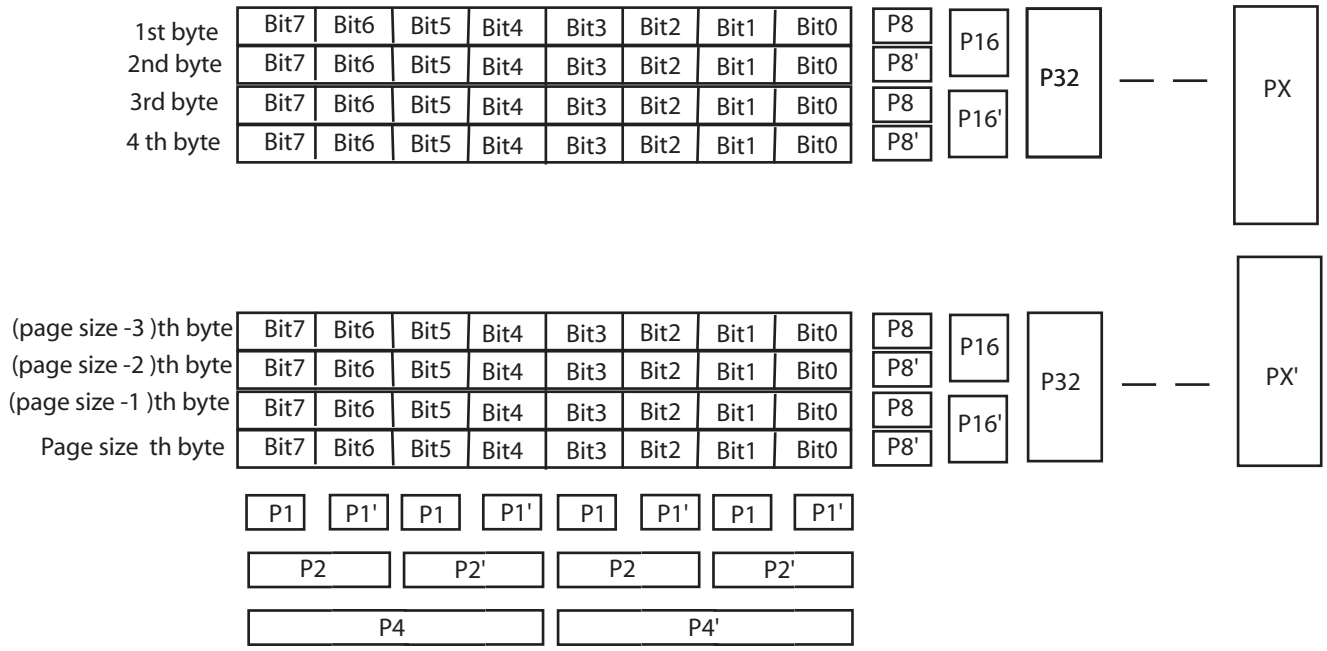
The application can check the ECC Status Registers (ECC_SR1/ECC_SR2) for any detected errors. It is up to the application to correct any detected error. ECC computation can detect four different circumstances:

- No error: XOR between the ECC computation and the ECC code stored at the end of the NAND Flash or SmartMedia page is equal to 0. No error flags in the ECC Status Registers (ECC_SR1/ECC_SR2).
- Recoverable error: Only the RECERR flags in the ECC Status registers (ECC_SR1/ECC_SR2) are set. The corrupted word offset in the read page is defined by the WORDADDR field in the ECC Parity Registers (ECC_PR0 to ECC_PR15). The corrupted bit position in the concerned word is defined in the BITADDR field in the ECC Parity Registers (ECC_PR0 to ECC_PR15).
- ECC error: The ECCERR flag in the ECC Status Registers (ECC_SR1/ECC_SR2) is set. An error has been detected in the ECC code stored in the Flash memory. The position of the corrupted bit can be found by the application performing an XOR between the Parity and the NParity contained in the ECC code stored in the Flash memory.
- Non correctable error: The MULERR flag in the ECC Status Registers (ECC_SR1/ECC_SR2) is set. Several unrecoverable errors have been detected in the Flash memory page.

ECC Status Registers, ECC Parity Registers are cleared when a read/write command is detected or a software reset is performed.

For Single-bit Error Correction and Double-bit Error Detection (SEC-DED) Hsiao code is used. 24-bit ECC is generated in order to perform one bit correction per 256 or 512 bytes for pages of 512/2048/4096 8-bit words. 32-bit ECC is generated in order to perform one bit correction per 512/1024/2048/4096 8- or 16-bit words. They are generated according to the schemes shown in Figure 25-39 and Figure 25-40.

Figure 25-39. Parity Generation for 512/1024/2048/4096 8-bit Words



Page size = 512 Px = 2048
Page size = 1024 Px = 4096
Page size = 2048 Px = 8192
Page size = 4096 Px = 16384

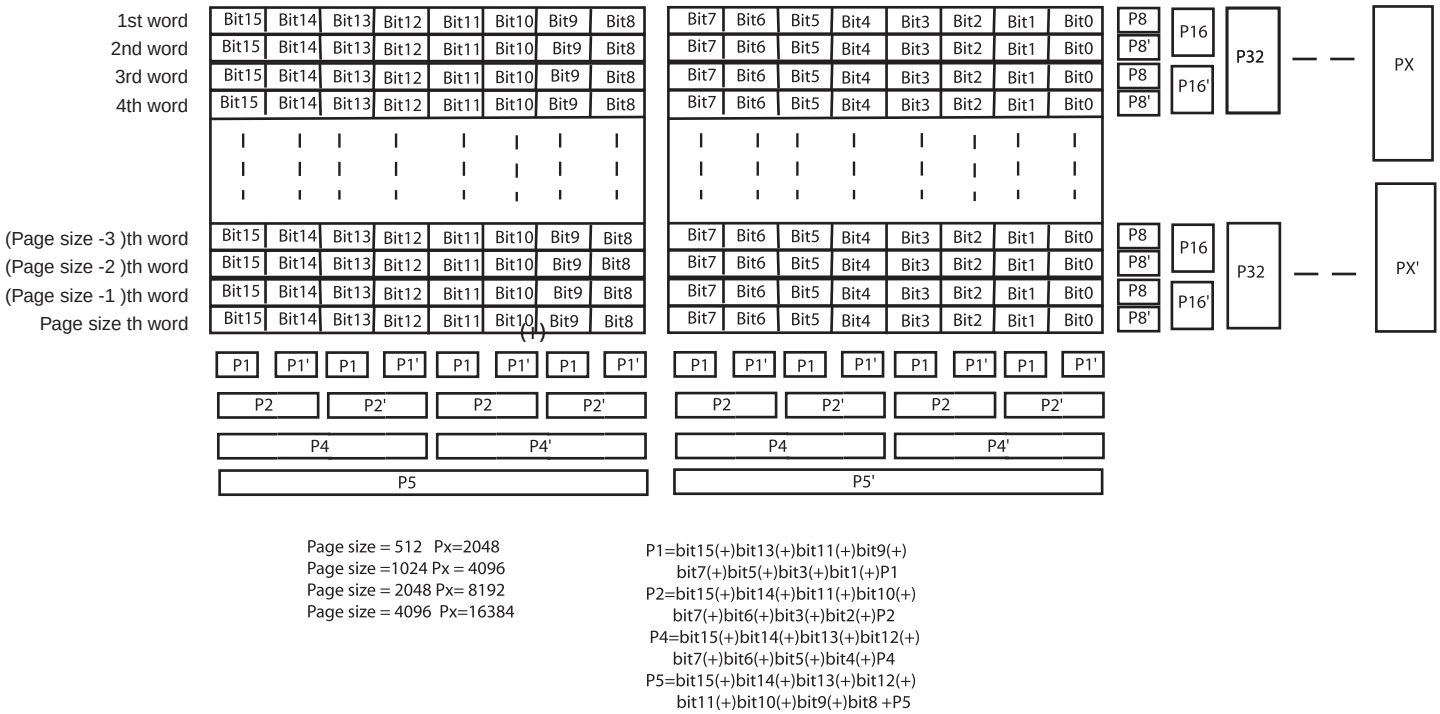
P1=bit7(+)bit5(+)bit3(+)bit1(+)P1
P2=bit7(+)bit6(+)bit3(+)bit2(+)P2
P4=bit7(+)bit6(+)bit5(+)bit4(+)P4
P1'=bit6(+)bit4(+)bit2(+)bit0(+)P1'
P2'=bit5(+)bit4(+)bit1(+)bit0(+)P2'
P4'=bit7(+)bit6(+)bit5(+)bit4(+)P4'

To calculate P8' to PX' and P8 to PX, apply the algorithm that follows.

Page size = 2^n

```
for i =0 to n
begin
 for (j = 0 to page_size_byte)
 begin
  if(j[i] ==1)
   P[2^(i+3)]=bit7(+)bit6(+)bit5(+)bit4(+)bit3(+)
              bit2(+)bit1(+)bit0(+)P[2^(i+3)]
  else
   P[2^(i+3)]'=bit7(+)bit6(+)bit5(+)bit4(+)bit3(+)
               bit2(+)bit1(+)bit0(+)P[2^(i+3)]'
  end
end
```

Figure 25-40. Parity Generation for 512/1024/2048/4096 16-bit Words



Page size = 512 Px=2048
Page size =1024 Px = 4096
Page size = 2048 Px= 8192
Page size = 4096 Px=16384

P1=bit15(+)bit13(+)bit11(+)bit9(+)
bit7(+)bit5(+)bit3(+)bit1(+)P1
P2=bit15(+)bit14(+)bit11(+)bit10(+)
bit7(+)bit6(+)bit3(+)bit2(+)P2
P4=bit15(+)bit14(+)bit13(+)bit12(+)
bit7(+)bit6(+)bit5(+)bit4(+)P4
P5=bit15(+)bit14(+)bit13(+)bit12(+)
bit11(+)bit10(+)bit9(+)bit8 +P5

To calculate P8' to PX' and P8 to PX, apply the algorithm that follows.

Page size = 2^n

```
for i =0 to n
begin
 for (j = 0 to page_size_word)
 begin
  if(j[i] ==1)
   P[2^(i+3)]= bit15(+)bit14(+)bit13(+)bit12(+)
               bit11(+)bit10(+)bit9(+)bit8(+)
               bit7(+)bit6(+)bit5(+)bit4(+)bit3(+)
               bit2(+)bit1(+)bit0(+)P[2^(n+3)]
  else
   P[2^(i+3)]'=bit15(+)bit14(+)bit13(+)bit12(+)
               bit11(+)bit10(+)bit9(+)bit8(+)
               bit7(+)bit6(+)bit5(+)bit4(+)bit3(+)
               bit2(+)bit1(+)bit0(+)P[2^(i+3)]'
  end
end
```

25.18 Static Memory Controller (SMC) User Interface

The SMC is programmed using the fields listed in Table 25-11. For each chip select a set of 4 registers is used to program the parameters of the external device. In Table 25-11, “CS_number” denotes chip select number. 16 bytes per chip select are required.

Table 25-11. Register Mapping

Offset	Register	Name	Access	Reset
0x000	SMC NFC Configuration Register	SMC_CFG	Read-write	0x0
0x004	SMC NFC Control Register	SMC_CTRL	Write-only	0x0
0x008	SMC NFC Status Register	SMC_SR	Read-only	0x0
0x00C	SMC NFC Interrupt Enable Register	SMC_IER	Write-only	0x0
0x010	SMC NFC Interrupt Disable Register	SMC_IDR	Write-only	0x0
0x014	SMC NFC Interrupt Mask Register	SMC_IMR	Read-only	0x0
0x018	SMC NFC Address Cycle Zero Register	SMC_ADDR	Read-write	0x0
0x01C	SMC Bank Address Register	SMC_BANK	Read-write	0x0
0x020	SMC ECC Control Register	SMC_ECC_CTRL	Write-only	0x0
0x024	SMC ECC Mode Register	SMC_ECC_MD	Read-write	0x0
0x028	SMC ECC Status 1 Register	SMC_ECC_SR1	Read-only	0x0
0x02C	SMC ECC Parity 0 Register	SMC_ECC_PR0	Read-only	0x0
0x030	SMC ECC parity 1 Register	SMC_ECC_PR1	Read-only	0x0
0x034	SMC ECC status 2 Register	SMC_ECC_SR2	Read-only	0x0
0x038	SMC ECC parity 2 Register	SMC_ECC_PR2	Read-only	0x0
0x03C	SMC ECC parity 3 Register	SMC_ECC_PR3	Read-only	0x0
0x040	SMC ECC parity 4 Register	SMC_ECC_PR4	Read-only	0x0
0x044	SMC ECC parity 5 Register	SMC_ECC_PR5	Read-only	0x0
0x048	SMC ECC parity 6 Register	SMC_ECC_PR6	Read-only	0x0
0x04C	SMC ECC parity 7 Register	SMC_ECC_PR7	Read-only	0x0
0x050	SMC ECC parity 8 Register	SMC_ECC_PR8	Read-only	0x0
0x054	SMC ECC parity 9 Register	SMC_ECC_PR9	Read-only	0x0
0x058	SMC ECC parity 10 Register	SMC_ECC_PR10	Read-only	0x0
0x05C	SMC ECC parity 11 Register	SMC_ECC_PR11	Read-only	0x0
0x060	SMC ECC parity 12 Register	SMC_ECC_PR12	Read-only	0x0
0x064	SMC ECC parity 13 Register	SMC_ECC_PR13	Read-only	0x0
0x068	SMC ECC parity 14 Register	SMC_ECC_PR14	Read-only	0x0
0x06C	SMC ECC parity 15 Register	SMC_ECC_PR15	Read-only	0x0
0x14*CS_number+0x070	SMC SETUP Register	SMC_SETUP	Read-write	0x01010101
0x14*CS_number+0x074	SMC PULSE Register	SMC_PULSE	Read-write	0x01010101
0x14*CS_number+0x078	SMC CYCLE Register	SMC_CYCLE	Read-write	0x00030003
0x14*CS_number+0x7C	SMC TIMINGS Register	SMC_TIMINGS	Read-write	0x00000000
0x14*CS_number+0x80	SMC MODE Register	SMC_MODE	Read-write	0x10000003

Table 25-11. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x110	SMC OCMS MODE Register	SMC_OCMS	Read-write	0x0
0x114	SMC KEY1 Register	SMC_KEY1	Write-only	0x0
0x118	SMC KEY2 Register	SMC_KEY2	Write-only	0x0
0x1E4	Write Protection Control Register	SMC_WPCR	Write-only	0x0
0x1E8	Write Protection Status Register	SMC_WPSR	Read-only	0x0
0x1FC	Reserved	–	–	–

25.18.1 SMC NFC Configuration Register

Name: SMC_CFG

Address: 0x400E0000

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	DTOMUL			DTOCYC			

15	14	13	12	11	10	9	8
–	–	RBEDGE	EDGECTRL	–	–	RSPARE	WSPARE

7	6	5	4	3	2	1	0
–	–	–	–	–	–	PAGESIZE	

- **PAGESIZE**

This field defines the page size of the NAND Flash device.

Value	Name	Description
0	PS512_16	Main area 512 Bytes + Spare area 16 Bytes = 528 Bytes
1	PS1024_32	Main area 1024 Bytes + Spare area 32 Bytes = 1056 Bytes
2	PS2048_64	Main area 2048 Bytes + Spare area 64 Bytes = 2112 Bytes
3	PS4096_128	Main area 4096 Bytes + Spare area 128 Bytes = 4224 Bytes

- **WSPARE: Write Spare Area**

0: The NFC skips the spare area in write mode.

1: The NFC writes both main area and spare area in write mode.

- **RSPARE: Read Spare Area**

0: The NFC skips the spare area in read mode.

1: The NFC reads both main area and spare area in read mode.

- **EDGECTRL: Rising/Falling Edge Detection Control**

0: Rising edge is detected.

1: Falling edge is detected.

- **RBEDGE: Ready/Busy Signal Edge Detection**

0: When set to zero, RB_EDGE fields indicate the level of the Ready/Busy lines.

1: When set to one, RB_EDGE fields indicate only transition on Ready/Busy lines.

- **DTOCYC: Data Timeout Cycle Number**

- **DTOMUL: Data Timeout Multiplier**

These fields determine the maximum number of Master Clock cycles that the SMC waits until the detection of a rising edge on Ready/Busy signal.

Multiplier is defined by DTOMUL as shown in the following table:

Value	Name	Description
0	X1	DTOCYC
1	X16	DTOCYC x 16
2	X128	DTOCYC x 128
3	X256	DTOCYC x 256
4	X1024	DTOCYC x 1024
5	X4096	DTOCYC x 4096
6	X65536	DTOCYC x 65536
7	X1048576	DTOCYC x 1048576

If the data timeout set by DTOCYC and DTOMUL has been exceeded, the Data Timeout Error flag (DTOE) in the SMC Status Register (SMC_SR) raises.

25.18.2 SMC NFC Control Register

Name: SMC_CTRL

Address: 0x400E0004

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
–	–	–	–	–	–	NFCDIS	NFCEN

- **NFCEN: NAND Flash Controller Enable**
- **NFCDIS: NAND Flash Controller Disable**

25.18.3 SMC NFC Status Register

Name: SMC_SR

Address: 0x400E0008

Access: Read-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	RB_EDGE0
23	22	21	20	19	18	17	16
NFCASE	AWB	UNDEF	DTOE	–	–	CMDDONE	XFRDONE
15	14	13	12	11	10	9	8
–	NFCSID			NFCWR	–	–	NFCBUSY
7	6	5	4	3	2	1	0
–	–	RB_FALL	RB_RISE	–	–	–	SMCSTS

- **SMCSTS: NAND Flash Controller status (this field cannot be reset)**

0: NAND Flash Controller is disabled.

1: NAND Flash Controller is enabled.

- **RB_RISE: Selected Ready Busy Rising Edge Detected**

When set to one, this flag indicates that a rising edge on Ready/Busy Line has been detected. This flag is reset after a status read operation. The Ready/Busy line selected is the decoding of the set NFCCSID, RBNSEL fields.

- **RB_FALL: Selected Ready Busy Falling Edge Detected**

When set to one, this flag indicates that a falling edge on Ready/Busy Line has been detected. This flag is reset after a status read operation. The Ready/Busy line is selected through the decoding of the set NFCSID, RBNSEL fields.

- **NFCBUSY: NFC Busy (this field cannot be reset)**

When set to one this flag indicates that the Controller is activated and accesses the memory device.

- **NFCWR: NFC Write/Read Operation (this field cannot be reset)**

When a command is issued, this field indicates the current Read or Write Operation. This field can be manually updated with the use of the SMC_CTRL register.

- **NFCSID: NFC Chip Select ID (this field cannot be reset)**

When a command is issued, this field indicates the value of the targeted chip select. This field can be manually updated with the use of the SMC_CTRL register.

- **XFRDONE: NFC Data Transfer Terminated**

When set to one, this flag indicates that the NFC has terminated the Data Transfer. This flag is reset after a status read operation.

- **CMDDONE: Command Done**

When set to one, this flag indicates that the NFC has terminated the Command. This flag is reset after a status read operation.

- **DTOE: Data Timeout Error**

When set to one this flag indicates that the Data timeout set be by DTOMUL and DTOCYC has been exceeded. This flag is reset after a status read operation.

- **UNDEF: Undefined Area Error**

When set to one this flag indicates that the processor performed an access in an undefined memory area. This flag is reset after a status read operation.

- **AWB: Accessing While Busy**

If set to one this flag indicates that an AHB master has performed an access during the busy phase. This flag is reset after a status read operation.

- **NFCASE: NFC Access Size Error**

If set to one, this flag indicates that an illegal access has been detected in the NFC Memory Area. Only Word Access is allowed within the NFC memory area. This flag is reset after a status read operation.

- **RB_EDGEx: Ready/Busy Line x Edge Detected**

If set to one, this flag indicates that an edge has been detected on the Ready/Busy Line x. Depending on the EDGE CTRL field located in the SMC_MODE register, only rising or falling edge is detected. This flag is reset after a status read operation.

25.18.4 SMC NFC Interrupt Enable Register

Name: SMC_IER

Address: 0x400E000C

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	RB_EDGE0
23	22	21	20	19	18	17	16
NFCASE	AWB	UNDEF	DTOE	–	–	CMDDONE	XFRDONE
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	RB_FALL	RB_RISE	–	–	–	–

- **RB_RISE: Ready Busy Rising Edge Detection Interrupt Enable**
- **RB_FALL: Ready Busy Falling Edge Detection Interrupt Enable**
- **XFRDONE: Transfer Done Interrupt Enable**
- **CMDDONE: Command Done Interrupt Enable**
- **DTOE: Data Timeout Error Interrupt Enable**
- **UNDEF: Undefined Area Access Interrupt Enable**
- **AWB: Accessing While Busy Interrupt Enable**
- **NFCASE: NFC Access Size Error Interrupt Enable**
- **RB_EDGEx: Ready/Busy Line x Interrupt Enable**

25.18.5 SMC NFC Interrupt Disable Register

Name: SMC_IDR

Address: 0x400E0010

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	RB_EDGE0
23	22	21	20	19	18	17	16
NFCASE	AWB	UNDEF	DTOE	–	–	CMDDONE	XFRDONE
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	RB_FALL	RB_RISE	–	–	–	–

- **RB_RISE: Ready Busy Rising Edge Detection Interrupt Disable**
- **RB_FALL: Ready Busy Falling Edge Detection Interrupt Disable**
- **XFRDONE: Transfer Done Interrupt Disable**
- **CMDDONE: Command Done Interrupt Disable**
- **DTOE: Data Timeout Error Interrupt Disable**
- **UNDEF: Undefined Area Access Interrupt Disable**
- **AWB: Accessing While Busy Interrupt Disable**
- **NFCASE: NFC Access Size Error Interrupt Disable**
- **RB_EDGEx: Ready/Busy Line x Interrupt Disable**

25.18.6 SMC NFC Interrupt Mask Register

Name: SMC_IMR

Address: 0x400E0014

Access: Read-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	RB_EDGE0
23	22	21	20	19	18	17	16
NFCASE	AWB	UNDEF	DTOE	–	–	CMDDONE	XFRDONE
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	RB_FALL	RB_RISE	–	–	–	–

- **RB_RISE: Ready Busy Rising Edge Detection Interrupt Mask**
- **RB_FALL: Ready Busy Falling Edge Detection Interrupt Mask**
- **XFRDONE: Transfer Done Interrupt Mask**
- **CMDDONE: Command Done Interrupt Mask**
- **DTOE: Data Timeout Error Interrupt Mask**
- **UNDEF: Undefined Area Access Interrupt Mask5**
- **AWB: Accessing While Busy Interrupt Mask**
- **NFCASE: NFC Access Size Error Interrupt Mask**
- **RB_EDGEx: Ready/Busy Line x Interrupt Mask**

25.18.7 SMC NFC Address Cycle Zero Register

Name: SMC_ADDR

Address: 0x400E0018

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
ADDR_CYCLE0							

- **ADDR_CYCLE0: NAND Flash Array Address cycle 0**

When 5 address cycles are used, ADDR_CYCLE0 is the first byte written to NAND Flash (used by the NFC).

25.18.8 SMC NFC Bank Register

Name: SMC_BANK

Address: 0x400E001C

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
–	–	–	–	–	BANK		

- **BANK: Bank Identifier**

Number of the Bank used

25.18.9 SMC ECC Control Register

Name: SMC_ECC_CTRL

Address: 0x400E0020

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	RST

- **RST: Reset ECC**
- **SWRST: Software Reset**

25.18.10 SMC ECC MODE Register

Name: SMC_ECC_MD

Address: 0x400E0024

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
–	–	TYPCORREC		–	–	ECC_PAGESIZE	

- **ECC_PAGESIZE**

This field defines the page size of the NAND Flash device.

Value	Name	Description
0	PS512_16	Main area 512 Bytes + Spare area 16 Bytes = 528 Bytes
1	PS1024_32	Main area 1024 Bytes + Spare area 32 Bytes = 1056 Bytes
2	PS2048_64	Main area 2048 Bytes + Spare area 64 Bytes = 2112 Bytes
3	PS4096_128	Main area 4096 Bytes + Spare area 128 Bytes = 4224 Bytes

- **TYPCORREC: Type of Correction**

Value	Name	Description
0	CPAGE	1 bit correction for a page of 512/1024/2048/4096 Bytes (for 8 or 16-bit NAND Flash)
1	C256B	1 bit correction for 256 Bytes of data for a page of 512/2048/4096 bytes (for 8-bit NAND Flash only)
2	C512B	1 bit correction for 512 Bytes of data for a page of 512/2048/4096 bytes (for 8-bit NAND Flash only)

25.18.11 SMC ECC Status Register 1

Name: SMC_ECC_SR1

Address: 0x400E0028

Access: Read-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	ECCERR7	ECCERR7	RECERR7	–	ECCERR6	ECCERR6	RECERR6
23	22	21	20	19	18	17	16
–	ECCERR5	ECCERR5	RECERR5	–	ECCERR4	ECCERR4	RECERR4
15	14	13	12	11	10	9	8
–	MULERR3	ECCERR3	RECERR3	–	MULERR2	ECCERR2	RECERR2
7	6	5	4	3	2	1	0
–	MULERR1	ECCERR1	RECERR1	–	ECCERR0	ECCERR0	RECERR0

- **RECERR0: Recoverable Error**

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected.

- **ECCERR0: ECC Error**

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

If TYPECORRECT = 0, read both ECC Parity 0 and ECC Parity 1 registers, the error occurred at the location which contains a 1 in the least significant 16 bits; else read ECC Parity 0 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR0: Multiple Error**

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR1: Recoverable Error in the page between the 256th and the 511th bytes or the 512nd and the 1023rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected.

- **ECCERR1: ECC Error in the page between the 256th and the 511th bytes or between the 512nd and the 1023rd bytes**

Fixed to 0 if TYPECORREC = 0

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 1 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR1: Multiple Error in the page between the 256th and the 511th bytes or between the 512nd and the 1023rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR2: Recoverable Error in the page between the 512nd and the 767th bytes or between the 1024th and the 1535th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise, multiple uncorrected errors were detected.

- **ECCERR2: ECC Error in the page between the 512nd and the 767th bytes or between the 1024th and the 1535th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 2 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR2: Multiple Error in the page between the 512nd and the 767th bytes or between the 1024th and the 1535th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR3: Recoverable Error in the page between the 768th and the 1023rd bytes or between the 1536th and the 2047th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected.

- **ECCERR3: ECC Error in the page between the 768th and the 1023rd bytes or between the 1536th and the 2047th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 3 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR3: Multiple Error in the page between the 768th and the 1023rd bytes or between the 1536th and the 2047th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR4: Recoverable Error in the page between the 1024th and the 1279th bytes or between the 2048th and the 2559th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected.

- **ECCERR4: ECC Error in the page between the 1024th and the 1279th bytes or between the 2048th and the 2559th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 4 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR4: Multiple Error in the page between the 1024th and the 1279th bytes or between the 2048th and the 2559th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR5: Recoverable Error in the page between the 1280th and the 1535th bytes or between the 2560th and the 3071st bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected

- **ECCERR5: ECC Error in the page between the 1280th and the 1535th bytes or between the 2560th and the 3071st bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 5 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR5: Multiple Error in the page between the 1280th and the 1535th bytes or between the 2560th and the 3071st bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR6: Recoverable Error in the page between the 1536th and the 1791st bytes or between the 3072nd and the 3583rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected.

- **ECCERR6: ECC Error in the page between the 1536th and the 1791st bytes or between the 3072nd and the 3583rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 6 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR6: Multiple Error in the page between the 1536th and the 1791st bytes or between the 3072nd and the 3583rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR7: Recoverable Error in the page between the 1792nd and the 2047th bytes or between the 3584th and the 4095th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise, multiple uncorrected errors were detected.

- **ECCERR7: ECC Error in the page between the 1792nd and the 2047th bytes or between the 3584th and the 4095th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 7 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR7: Multiple Error in the page between the 1792nd and the 2047th bytes or between the 3584th and the 4095th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

25.18.12 SMC ECC Status Register 2

Name: SMC_ECC_SR2

Address: 0x400E0034

Access: Read-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	ECCERR15	ECCERR15	RECERR15	–	ECCERR14	ECCERR14	RECERR14
23	22	21	20	19	18	17	16
–	ECCERR13	ECCERR13	RECERR13	–	ECCERR12	ECCERR12	RECERR12
15	14	13	12	11	10	9	8
–	MULERR11	ECCERR11	RECERR11	–	MULERR10	ECCERR10	RECERR10
7	6	5	4	3	2	1	0
–	MULERR9	ECCERR9	RECERR9	–	ECCERR8	ECCERR8	RECERR8

- **RECERR8: Recoverable Error in the page between the 2048th and the 2303rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected

- **ECCERR8: ECC Error in the page between the 2048th and the 2303rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 8 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR8: Multiple Error in the page between the 2048th and the 2303rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR9: Recoverable Error in the page between the 2304th and the 2559th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected.

- **ECCERR9: ECC Error in the page between the 2304th and the 2559th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 9 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR9: Multiple Error in the page between the 2304th and the 2559th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR10: Recoverable Error in the page between the 2560th and the 2815th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise, multiple uncorrected errors were detected.

- **ECCERR10: ECC Error in the page between the 2560th and the 2815th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 10 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR10: Multiple Error in the page between the 2560th and the 2815th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR11: Recoverable Error in the page between the 2816th and the 3071st bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise, multiple uncorrected errors were detected

- **ECCERR11: ECC Error in the page between the 2816th and the 3071st bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 11 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR11: Multiple Error in the page between the 2816th and the 3071st bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR12: Recoverable Error in the page between the 3072nd and the 3327th bytes**

Fixed to 0 if TYPECORREC = 0

0: No Errors Detected

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected

- **ECCERR12: ECC Error in the page between the 3072nd and the 3327th bytes**

Fixed to 0 if TYPECORREC = 0

0: No Errors Detected

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 12 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR12: Multiple Error in the page between the 3072nd and the 3327th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR13: Recoverable Error in the page between the 3328th and the 3583rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise multiple uncorrected errors were detected.

- **ECCERR13: ECC Error in the page between the 3328th and the 3583rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 13 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR13: Multiple Error in the page between the 3328th and the 3583rd bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR14: Recoverable Error in the page between the 3584th and the 3839th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise, multiple uncorrected errors were detected.

- **ECCERR14: ECC Error in the page between the 3584th and the 3839th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 14 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR14: Multiple Error in the page between the 3584th and the 3839th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

- **RECERR15: Recoverable Error in the page between the 3840th and the 4095th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: Errors Detected. If MUL_ERROR is 0, a single correctable error was detected. Otherwise, multiple uncorrected errors were detected.

- **ECCERR15: ECC Error in the page between the 3840th and the 4095th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Errors Detected.

1: A single bit error occurred in the ECC bytes.

Read ECC Parity 15 register, the error occurred at the location which contains a 1 in the least significant 24 bits.

- **MULERR15: Multiple Error in the page between the 3840th and the 4095th bytes**

Fixed to 0 if TYPECORREC = 0.

0: No Multiple Errors Detected.

1: Multiple Errors Detected.

25.18.13 SMC ECC Parity Register 0 for a Page of 512/1024/2048/4096 Bytes

Name: SMC_ECC_PR0

Address: 0x400E002C

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
WORDADDR							
7	6	5	4	3	2	1	0
WORDADDR				BITADDR			

Once the entire main area of a page is written with data, the register content must be stored at any free location of the spare area.

- **BITADDR: Bit Address**

During a page read, this value contains the corrupted bit offset where an error occurred, if a single error was detected. If multiple errors were detected, this value is meaningless.

- **WORDADDR: Word Address**

During a page read, this value contains the word address (8-bit or 16-bit word depending on the memory plane organization).where an error occurred, if a single error was detected. If multiple errors were detected, this value is meaningless.

25.18.14 SMC ECC Parity Register 1 for a Page of 512/1024/2048/4096 Bytes

Name: SMC_ECC_PR1

Address: 0x400E0030

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
NPARITY							
7	6	5	4	3	2	1	0
NPARITY							

- **NPARITY: Parity N**

25.18.15 SMC ECC Parity Registers for 1 ECC per 512 Bytes for a Page of 512/2048/4096 Bytes, 9-bit Word

Name: SMC_ECC_PRx [x=0..7] (W9BIT)

Address: 0x400E0038 [2] .. 0x400E006C [15]

Access: Read-only

Reset: 0x00000000

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

Once the entire main area of a page is written with data, the register content must be stored at any free location of the spare area.

- **BITADDR: Corrupted Bit Address in the Page between (i x 512) and ((i + 1) x 512) - 1) Bytes**

During a page read, this value contains the corrupted bit offset where an error occurred, if a single error was detected. If multiple errors were detected, this value is meaningless.

- **WORDADDR: Corrupted Word Address in the Page between (i x 512) and ((i + 1) x 512) - 1) Bytes**

During a page read, this value contains the word address (9-bit word) where an error occurred, if a single error was detected. If multiple errors were detected, this value is meaningless.

- **NPARITY: Parity N**

25.18.16 SMC ECC Parity Registers for 1 ECC per 256 Bytes for a Page of 512/2048/4096 Bytes, 8-bit Word

Name: SMC_ECC_PRx [x=0..15] (W8BIT)

Address: 0x400E0038 [2] .. 0x400E006C [15]

Access: Read-only

Reset: 0x00000000

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

Once the entire main area of a page is written with data, the register content must be stored at any free location of the spare area.

- **BITADDR: Corrupted Bit Address in the page between (i x 256) and ((i + 1) x 512) - 1) Bytes**
- **During a page read, this value contains the corrupted bit offset where an error occurred, if a single error was detected. If multiple errors were detected, this value is meaningless.**
- **WORDADDR: Corrupted Word Address in the page between (i x 256) and ((i + 1) x 512) - 1) Bytes**

During a page read, this value contains the word address (8-bit word) where an error occurred, if a single error was detected. If multiple errors were detected, this value is meaningless.

- **NPARITY: Parity N**

25.18.17 SMC Setup Register

Name: SMC_SETUPx [x=0..7]

Address: 0x400E0070 [0], 0x400E0084 [1], 0x400E0098 [2], 0x400E00AC [3], 0x400E00C0 [4], 0x400E00D4 [5], 0x400E00E8 [6], 0x400E00FC [7]

Access: Write-only

Reset: 0x00000000

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:

NWE setup length = (128 * NWE_SETUP[5] + NWE_SETUP[4:0]) clock cycles.

- **NCS_WR_SETUP: NCS Setup length in Write access**

In write access, the NCS signal setup length is defined as:

NCS setup length = (128 * NCS_WR_SETUP[5] + NCS_WR_SETUP[4:0]) clock cycles.

- **NRD_SETUP: NRD Setup length**

The NRD signal setup length is defined as:

NRD setup length = (128 * NRD_SETUP[5] + NRD_SETUP[4:0]) clock cycles.

- **NCS_RD_SETUP: NCS Setup length in Read access**

In Read access, the NCS signal setup length is defined as:

NCS setup length = (128 * NCS_RD_SETUP[5] + NCS_RD_SETUP[4:0]) clock cycles.

25.18.18 SMC Pulse Register

Name: SMC_PULSEx [x=0..7]

Address: 0x400E0074 [0], 0x400E0088 [1], 0x400E009C [2], 0x400E00B0 [3], 0x400E00C4 [4], 0x400E00D8 [5], 0x400E00EC [6], 0x400E0100 [7]

Access: Write-only

Reset: 0x00000000

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:

NWE pulse length = (256 * NWE_PULSE[6]+NWE_PULSE[5:0]) clock cycles.

The NWE pulse must be at least one clock cycle.

- **NCS_WR_PULSE: NCS Pulse Length in WRITE Access**

In Write access, The NCS signal pulse length is defined as:

NCS pulse length = (256 * NCS_WR_PULSE[6] + NCS_WR_PULSE[5:0]) clock cycles.

the NCS pulse must be at least one clock cycle.

- **NRD_PULSE: NRD Pulse Length**

The NRD signal pulse length is defined as:

NRD pulse length = (256 * NRD_PULSE[6] + NRD_PULSE[5:0]) clock cycles.

The NRD pulse width must be as least 1 clock cycle.

- **NCS_RD_PULSE: NCS Pulse Length in READ Access**

In READ mode, The NCS signal pulse length is defined as:

NCS pulse length = (256 * NCS_RD_PULSE[6] + NCS_RD_PULSE[5:0]) clock cycles.

25.18.19 SMC Cycle Register

Name: SMC_CYCLEx [x=0..7]

Address: 0x400E0078 [0], 0x400E008C [1], 0x400E00A0 [2], 0x400E00B4 [3], 0x400E00C8 [4], 0x400E00DC [5], 0x400E00F0 [6], 0x400E0104 [7]

Access: Read-write

Reset: 0x00000000

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.

25.18.20 SMC Timings Register

Name: SMC_TIMINGSx [x=0..7]

Address: 0x400E007C [0], 0x400E0090 [1], 0x400E00A4 [2], 0x400E00B8 [3], 0x400E00CC [4], 0x400E00E0 [5], 0x400E00F4 [6], 0x400E0108 [7]

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
NFSEL	RBNSEL			TWB			
23	22	21	20	19	18	17	16
–	–	–	–	TRR			
15	14	13	12	11	10	9	8
–	–	–	OCMS	TAR			
7	6	5	4	3	2	1	0
TADL				TCLR			

- **TCLR: CLE to REN Low Delay**

Command Latch Enable falling edge to Read Enable falling edge timing.

Latch Enable Falling to Read Enable Falling = (TCLR[3] * 64) + TCLR[2:0] clock cycles.

- **TADL: ALE to Data Start**

Last address latch cycle to the first rising edge of WEN for data input.

Last address latch to first rising edge of WEN = (TADL[3] * 64) + TADL[2:0] clock cycles.

- **TAR: ALE to REN Low Delay**

Address Latch Enable falling edge to Read Enable falling edge timing.

Address Latch Enable to Read Enable = (TAR[3] * 64) + TAR[2:0] clock cycles.

- **OCMS: Off Chip Memory Scrambling Enable**

When set to one, the memory scrambling is activated.

- **TRR: Ready to REN Low Delay**

Ready/Busy signal to Read Enable falling edge timing.

Read to REN = (TRR[3] * 64) + TRR[2:0] clock cycles.

- **TWB: WEN High to REN to Busy**

Write Enable rising edge to Ready/Busy falling edge timing.

Write Enable to Read/Busy = (TWB[3] * 64) + TWB[2:0] clock cycles.

- **RBNSEL: Ready/Busy Line Selection**

This field indicates the selected Ready/Busy Line from the RBN bundle.

- **NFSEL: NAND Flash Selection**

If this bit is set to one, the chip select is assigned to NAND Flash write enable and read enable lines drive the Error Correcting Code module.

25.18.21 SMC Mode Register

Name: SMC_MODEx [x=0..7]

Address: 0x400E0080 [0], 0x400E0094 [1], 0x400E00A8 [2], 0x400E00BC [3], 0x400E00D0 [4], 0x400E00E4 [5], 0x400E00F8 [6], 0x400E010C [7]

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	TDF_MODE	TDF_CYCLES			

15	14	13	12	11	10	9	8
–	–	–	DBW	–	–	–	BAT

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.

0: The Read operation is controlled by the NCS signal.

- **WRITE_MODE**

1: The Write operation is controlled by the NWE signal.

0: The Write operation is controller by the NCS signal.

- **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 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.

- **BAT: Byte Access Type**

This field is used only if DBW defines a 16-bit data bus.

- 1: Byte write access type:
  - Write operation is controlled using NCS, NWR0, NWR1.
  - Read operation is controlled using NCS and NRD.
- 0: Byte select access type:
  - Write operation is controlled using NCS, NWE, NBS0, NBS1.
  - Read operation is controlled using NCS, NRD, NBS0, NBS1.

- **DBW: Data Bus Width**

Value	Name	Description
0	BIT_8	8-bit bus
1	BIT_16	16-bit bus

- **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.

25.18.22 SMC OCMS Register

Name: SMC_OCMS

Address: 0x400E0110

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
–	–	–	–	–	–	SRSE	SMSE

- **SMSE: Static Memory Controller Scrambling Enable**

0: Disable “Off Chip” Scrambling for SMC access.

1: Enable “Off Chip” Scrambling for SMC access. (If OCMS field is set to 1 in the relevant SMC_TIMINGS register.)

- **SRSE: SRAM Scrambling Enable**

0: Disable SRAM Scrambling for SRAM access.

1: Enable SRAM Scrambling for SRAM access. (If OCMS field is set to 1 in the relevant SMC_TIMINGS register.)

25.18.23 SMC OCMS Key1 Register

Name: SMC_KEY1

Address: 0x400E0114

Access: Write Once

Reset: 0x00000000

31	30	29	28	27	26	25	24
KEY1							

23	22	21	20	19	18	17	16
KEY1							

15	14	13	12	11	10	9	8
KEY1							

7	6	5	4	3	2	1	0
KEY1							

- **KEY1: Off Chip Memory Scrambling (OCMS) Key Part 1**

When Off Chip Memory Scrambling is enabled by setting the SMC_OMCS and SMC_TIMINGS registers in accordance, the data scrambling depends on KEY1 and KEY2 values.

25.18.24 SMC OCMS Key2 Register

Name: SMC_KEY2

Address: 0x400E0118

Access: Write Once

Reset: 0x00000000

31	30	29	28	27	26	25	24
KEY2							

23	22	21	20	19	18	17	16
KEY2							

15	14	13	12	11	10	9	8
KEY2							

7	6	5	4	3	2	1	0
KEY2							

- **KEY2: Off Chip Memory Scrambling (OCMS) Key Part 2**

When Off Chip Memory Scrambling is enabled by setting the SMC_OMCS and SMC_TIMINGS registers in accordance, the data scrambling depends on KEY2 and KEY1 values.

25.18.25 SMC Write Protection Control

Name: SMC_WPCR

Address: 0x400E01E4

Access: Write-only

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

- **WP_EN: Write Protection Enable**

0: Disables the Write Protection if WP_KEY corresponds.

1: Enables the Write Protection if WP_KEY corresponds.

- **WP_KEY: Write Protection KEY password**

Should be written at value **0x534D43** (ASCII code for “SMC”). Writing any other value in this field has no effect.

25.18.26 SMC Write Protection Status

Name: SMC_WPSR

Address: 0x400E01E8

Access: Read-only

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

- **WP_VS: Write Protection Violation Status**

0: No Write Protect Violation has occurred since the last read of the SMC_WPSR register.

1: A Write Protect Violation has 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 WP_VSRC.

- **WP_VSRC: Write Protection Violation SouRCe**

WP_VSRC field Indicates the Register offset where the last violation occurred.

26. Peripheral DMA Controller (PDC)

26.1 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.

26.2 Embedded Characteristics

- Handles data transfer between peripherals and memories
- Low bus arbitration overhead
  - One Master Clock cycle needed for a transfer from memory to peripheral
  - Two Master Clock cycles needed for a transfer from peripheral to memory
- Next Pointer management for reducing interrupt latency requirement

The Peripheral DMA Controller handles transfer requests from the channel according to the following priorities (Low to High priorities):

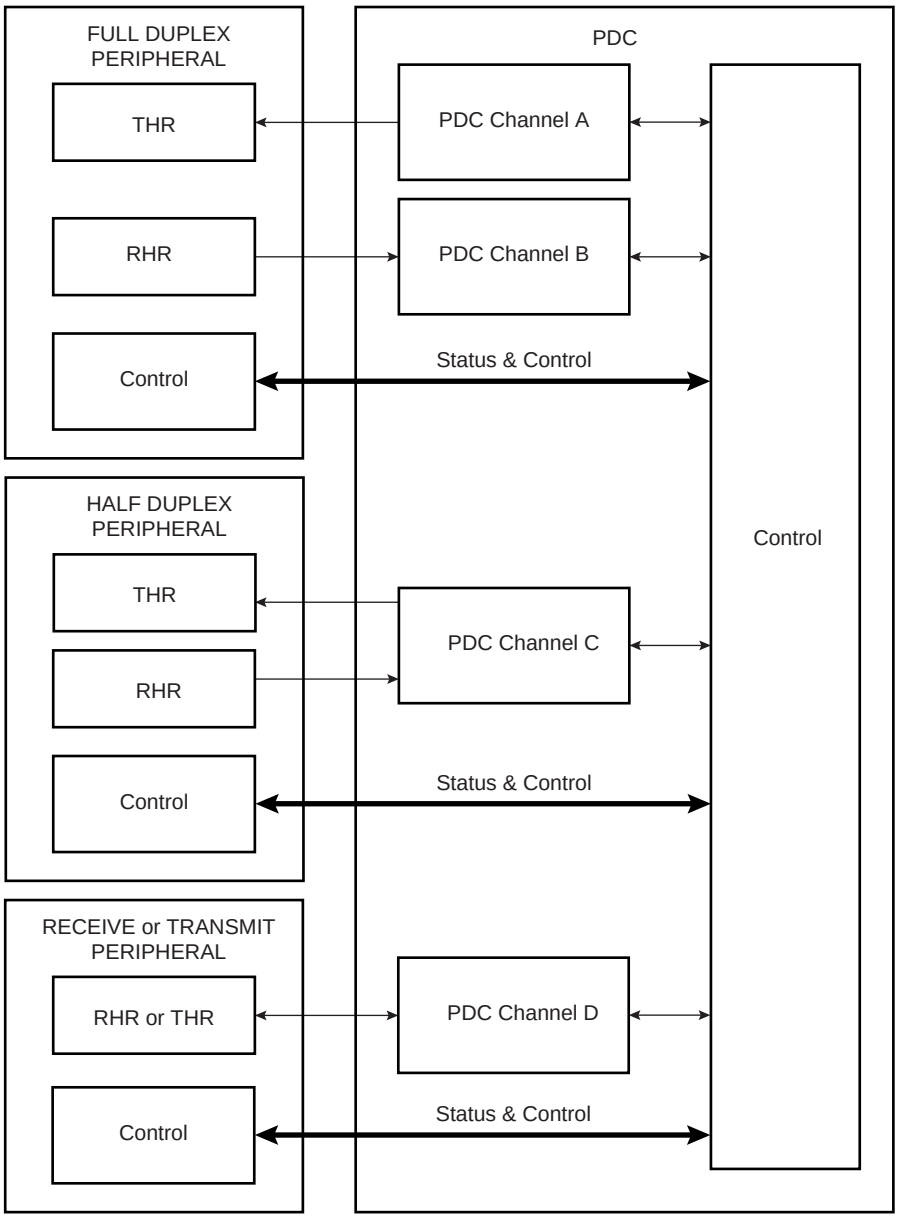
Table 26-1. Peripheral DMA Controller

Instance Name	Channel T/R	217 Pins[1]	144 Pins	100 Pins
DAC	Transmit	X	X	X
PWM	Transmit	X	X	X
TWI1	Transmit	X	X	X
TWI0	Transmit	X	X	X
USART3	Transmit	X	X	N/A
USART2	Transmit	X	X	X
USART1	Transmit	X	X	X
USART0	Transmit	X	X	X
UART	Transmit	X	X	X
ADC	Receive	X	X	X
TWI1	Receive	X	X	X
TWI0	Receive	X	X	X
USART3	Receive	X	X	N/A
USART2	Receive	X	X	X
USART1	Receive	X	X	X
USART0	Receive	X	X	X
UART	Receive	X	X	X

Note: 1.Corresponds to 217-pin SAM3X8H. This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

26.3 Block Diagram

Figure 26-1. Block Diagram



26.4 Functional Description

26.4.1 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 26.4.3 and to the associated peripheral user interface.

26.4.2 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.

26.4.3 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 then 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.

26.4.4 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.

26.4.5 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.

26.4.5.1 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.

26.4.5.2 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.

26.4.5.3 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.

26.4.5.4 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.

26.5 Peripheral DMA Controller (PDC) User Interface

Table 26-2. Register Mapping

Offset	Register	Name	Access	Reset
0x100	Receive Pointer Register	PERIPH[(1)]_RPR	Read-write	0
0x104	Receive Counter Register	PERIPH_RCR	Read-write	0
0x108	Transmit Pointer Register	PERIPH_TPR	Read-write	0
0x10C	Transmit Counter Register	PERIPH_TCR	Read-write	0
0x110	Receive Next Pointer Register	PERIPH_RNPR	Read-write	0
0x114	Receive Next Counter Register	PERIPH_RNCR	Read-write	0
0x118	Transmit Next Pointer Register	PERIPH_TNPR	Read-write	0
0x11C	Transmit Next Counter Register	PERIPH_TNCR	Read-write	0
0x120	Transfer Control Register	PERIPH_PTCR	Write-only	0
0x124	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.)

26.5.1 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.

26.5.2 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

26.5.3 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.

26.5.4 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

26.5.5 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.

26.5.6 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.

26.5.7 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.

26.5.8 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.

26.5.9 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.

26.5.10 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.

27. Clock Generator

27.1 Description

The Clock Generator User Interface is embedded within the Power Management Controller and is described in Section 28.15 ”Power Management Controller (PMC) User Interface”. However, the Clock Generator registers are named CKGR_.

27.2 Embedded Characteristics

The Clock Generator is made up of:

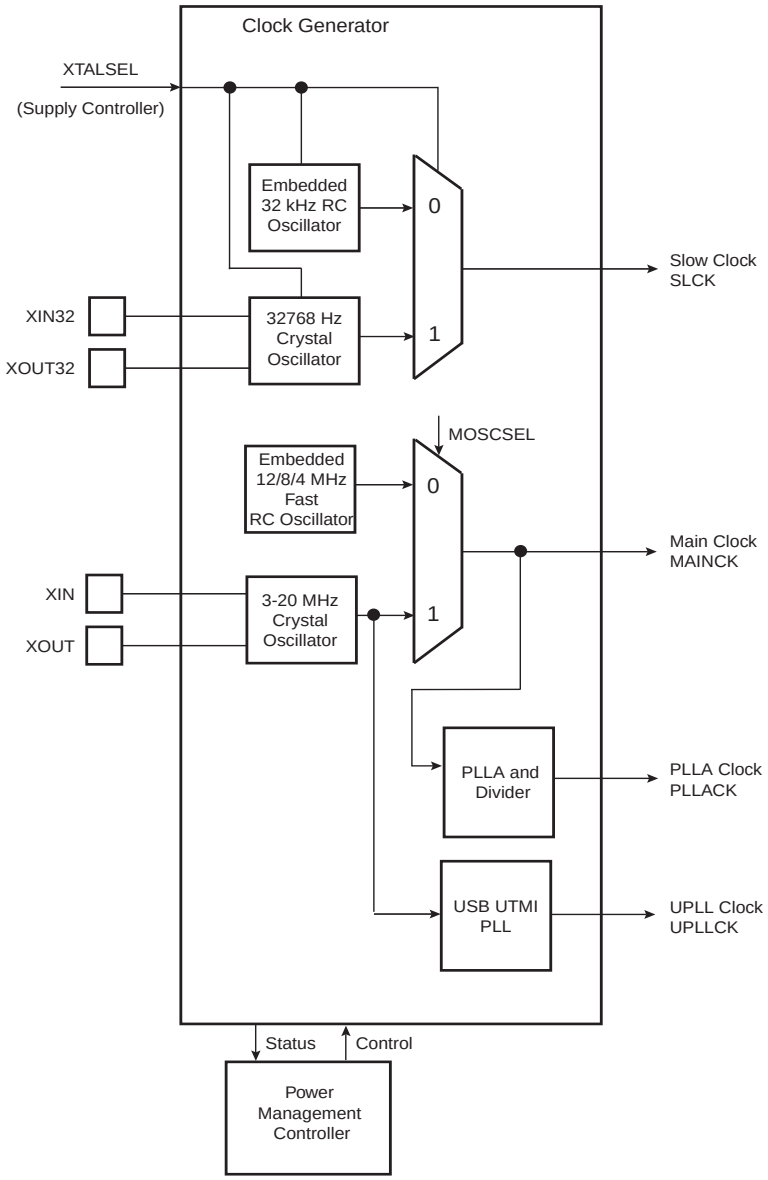
- A Low Power 32,768 Hz Slow Clock Oscillator with bypass mode.
- A Low Power RC Oscillator
- A 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator, which can be bypassed.
- A factory programmed Fast RC Oscillator. 3 output frequencies can be selected: 4, 8 or 12 MHz. By default 4MHz is selected.
- A 480 MHz UTMI PLL, providing a clock for the USB High Speed Controller.
- A 96 to 192 MHz programmable PLL (input from 8 to 16 MHz), capable of providing the clock MCK to the processor and to the peripherals.

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 96 to 192 MHz programmable PLL (PLLA).
- UPLLCK is the output of the 480 MHz UTMI PLL (UPLL).

27.3 Block Diagram

Figure 27-1. Clock Generator Block Diagram



27.4 Slow Clock

The Supply Controller embeds a slow clock generator that is supplied with the VDDBU power supply. As soon as the VDDBU 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 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).

27.4.1 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.

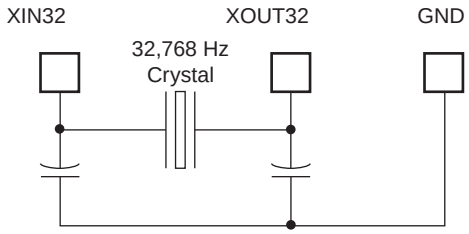
It can be disabled via the XTALSEL bit in the Supply Controller Control Register (SUPC_CR).

27.4.2 Slow Clock Crystal Oscillator

The Clock Generator integrates a 32,768 Hz low-power oscillator. In order to use this oscillator, the XIN32 and XOUT32 pins must be connected to a 32,768 Hz crystal. Two external capacitors must be wired as shown in Figure 27-2. 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 27-2. 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 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) 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 MOSCSEL bit of CKGR_MOR, is done.

Coming back on the RC oscillator is only possible by shutting down the VDDBU power supply. If the user does not need the crystal oscillator, the XIN32 and XOUT32 pins can 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.

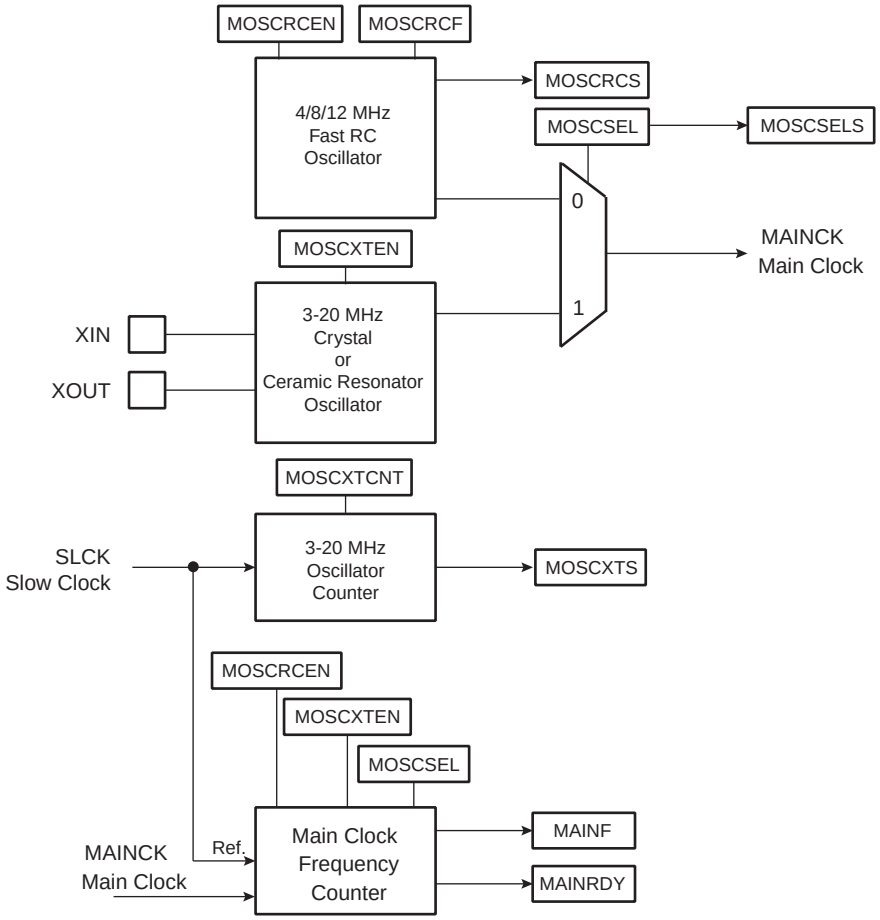
The user can set the Slow Clock Crystal Oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin under these conditions are given in the product electrical characteristics section.

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).

27.5 Main Clock

Figure 27-3 shows the Main Clock block diagram.

Figure 27-3. 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 startup.
- 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator which can be bypassed.

27.5.1 4/8/12 MHz 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 8 and 12 MHz frequencies are calibrated in production. Note that is not the case for the 4 MHz frequency.

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 MHz, 8 MHz or 12 MHz are available. It can be done through MOSCRCF bits in CKGR_MOR. When changing this frequency 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, PLLACKor UPLLCK.

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.

27.5.2 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 startup time of the oscillator. This startup time depends on the crystal frequency connected to the oscillator.

When the MOSCXTEN bit and the MOSCXTCNT 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 MOSCXTCNT value. Since the MOSCXTCNT value is coded with 8 bits, the maximum startup 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.

27.5.3 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 startup 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, PLLACKor UPLLCK 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.

27.5.4 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

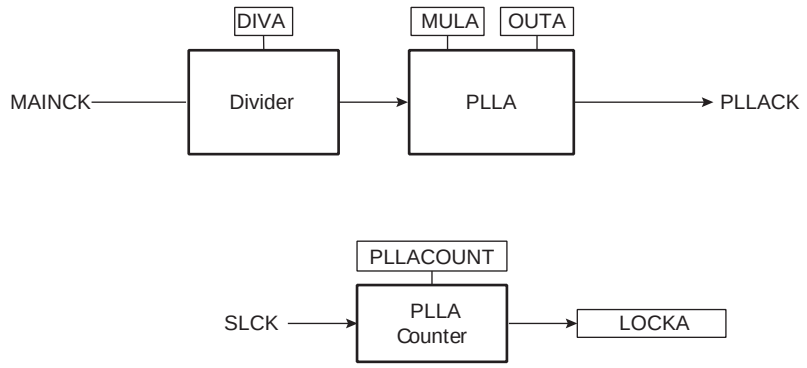
Then, at the 16th falling edge of Slow Clock, the MAINFRDY bit in the Clock Generator Main Clock Frequency Register (CKGR_MCFR) is set and the counter stops counting. Its value can be read in the MAINF field of CKGR_MCFR and gives the number of Main Clock cycles during 16 periods of Slow Clock, so that 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.

27.6 Divider and PLL Block

The device features one Divider/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 27-4 shows the block diagram of the dividers and PLL blocks.

Figure 27-4. Dividers and PLL Block Diagram



27.6.1 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, 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.

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 (PLLACOUNT) 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.

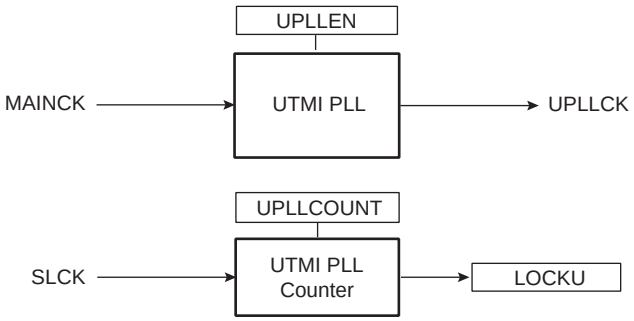
The user must:

- 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_IER.
- Disable and then enable the PLL (LOCK in PMC_IDR and PMC_IER).
- Wait for PLLRDY.
- Switch back to PLL.

27.7 UTMI Phase Lock Loop Programming

The source clock of the UTMI PLL is the 3-20 MHz crystal oscillator. A 12 MHz crystal is needed to use the USB.

Figure 27-5. UTMI PLL Block Diagram



Whenever the UTMI PLL is enabled by writing UPLLEN in CKGR_UCKR, the LOCKU bit in PMC_SR is automatically cleared. The values written in the PLLCOUNT field in CKGR_UCKR are loaded in the UTMI PLL counter. The UTMI PLL counter then decrements at the speed of the Slow Clock divided by 8 until it reaches 0. At this time, the LOCKU 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 UTMI PLL transient time into the PLLCOUNT field.

28. Power Management Controller (PMC)

28.1 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-M3 Processor.

The Supply Controller selects between the 32 kHz RC oscillator or the crystal oscillator. The unused oscillator is disabled automatically so that power consumption is optimized.

By default, at startup 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.

28.2 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) is automatically switched off when the processor enters Sleep Mode.
- Free running processor Clock (FCLK)
- the Cortex-M3 SysTick external clock
- UDP Clock (UDPCK), required by USB Device Port operations.
- Peripheral Clocks, typically MCK, provided to the embedded peripherals (USART, SSC, SPI, TWI, TC, HSMCI, etc.) and independently controllable. Some of the peripherals can be configured to be driven by MCK divided by 2, 4.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.

The Power Management Controller also provides the following operations on clocks:

- a main crystal oscillator clock failure detector.
- a frequency counter on main clock .

28.3 Block Diagram

Figure 28-1. General Clock Block Diagram

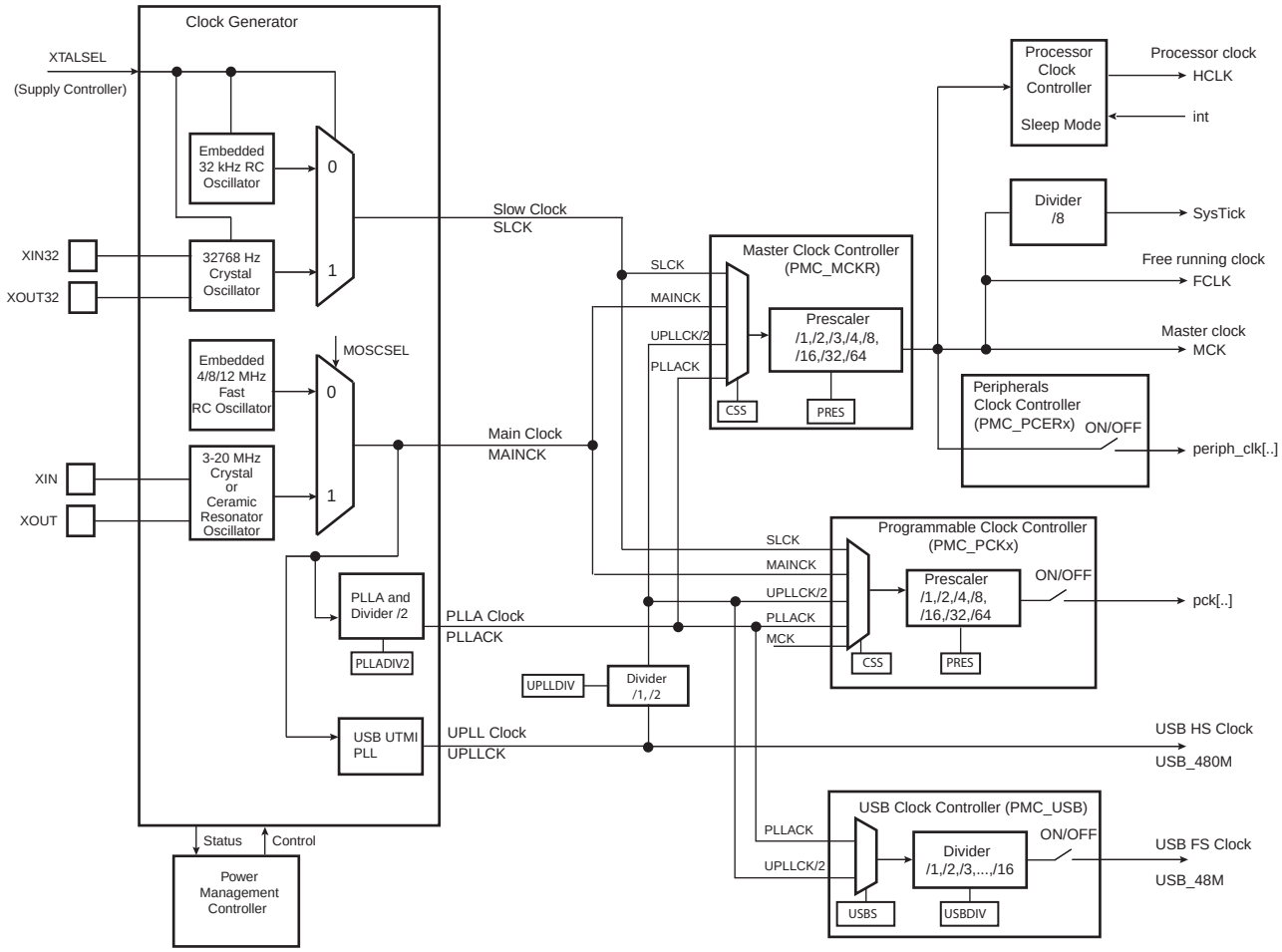
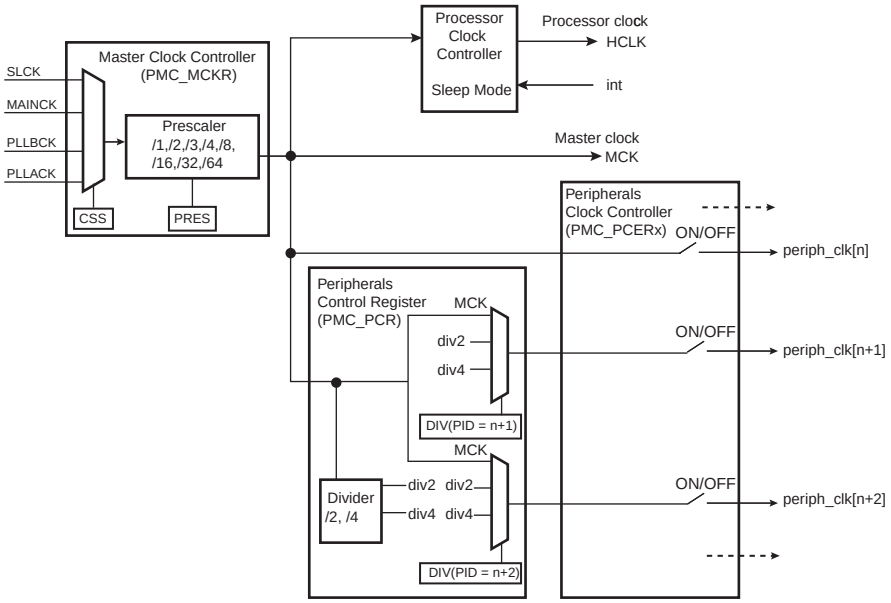


Figure 28-2. Peripheral Clock Divider Block Diagram



28.4 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.

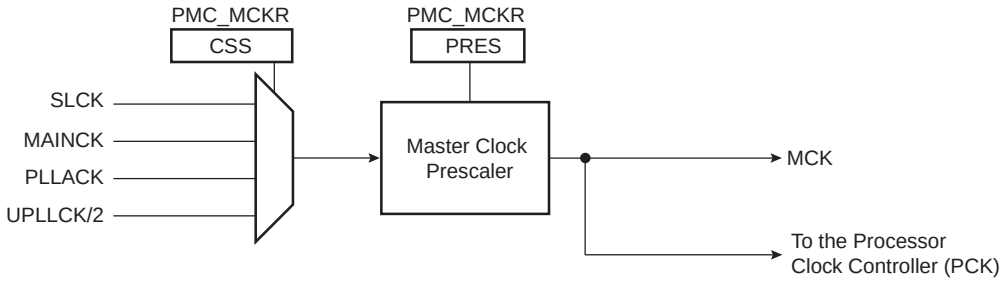
The Master Clock is selected from one of the clocks provided by the Clock Generator. Selecting the Slow Clock provides a Slow Clock signal to the whole device. Selecting the Main Clock saves power consumption of the PLLs.

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 high-speed clock to a lower one to inform the software when the change is actually done.

Figure 28-3. Master Clock Controller



28.5 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 Startup 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.

28.6 SysTick Clock

The SysTick calibration value is fixed to 10500 which allows the generation of a time base of 1 ms with SysTick clock to the maximum frequency on MCK divided by 8.

28.7 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.

In order to save power consumption, the clock of CAN0, CAN1 peripherals can be MCK divided by a division factor of 1, 2, 4.
This is done by setting the PMC_PCR register. It features a command and acts like a mailbox. To write the division factor, the user needs to write a WRITE command, the peripheral ID and the chosen division factor. To read the current division factor, the user just needs to write the READ command and the peripheral ID. Then a read access on PMC_PCR must be performed.
DIV must not be changed while peripheral is in use or when the peripheral clock is enabled.To change the clock division factor (DIV) of a peripheral, its clock must first be disabled by writing either EN to 0 for the corresponding PID (DIV must be kept the same if this method is used), or writing to PMC_PCDR register. Then a second write must be performed into PMC_PCR with the new value of DIV and a third write must be performed to enable the peripheral clock (either by using PMC_PCR or PMC_PCER register).

Code Example to select divider 4 for peripheral index 43 (0x2B) and enable its clock:

```
write_register(PMC_PCR,0x1002102B)
```

Code Example to read the divider of the same peripheral:

```
write_register(PMC_PCR,0x0000002B)
read_register(PMC_PCR)
```

28.8 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).

28.9 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), UTMI PLL Clock (UPLLCK/2) 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.

28.10 Fast Startup

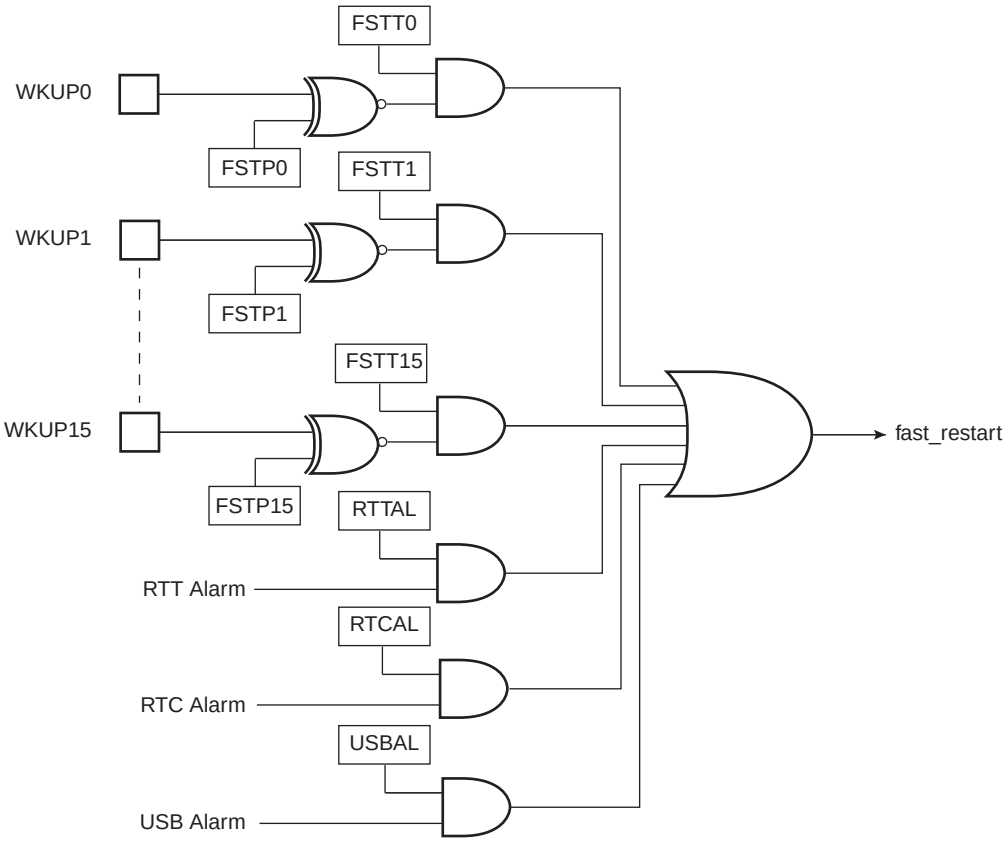
The device allows the processor to restart in less than 10 microseconds while the device is in Wait mode. The system enters Wait mode by executing the WaitForEvent (WFE) instruction of the processor while the LPM bit is at 1 in the PMC Fast Startup Mode Register (PMC_FSMR).

Important: Prior to asserting any WFE instruction to the processor, the internal sources of wakeup provided by RTT, RTC and USB must be cleared and verified too, that none of the enabled external wakeup inputs (WKUP) hold an active polarity.

A Fast Startup 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 Startup Polarity Register (PMC_FSPR).

The Fast Restart circuitry, as shown in Figure 28-4, is fully asynchronous and provides a fast startup signal to the Power Management Controller. As soon as the fast startup signal is asserted, the embedded 4/8/12 MHz Fast RC oscillator restarts automatically.

Figure 28-4. Fast Startup Circuitry



Each wake-up input pin and alarm can be enabled to generate a Fast Startup event by writing 1 to the corresponding bit in the Fast Startup Mode Register PMC_FSMR.

The user interface does not provide any status for Fast Startup, but the user can easily recover this information by reading the PIO Controller, and the status registers of the RTC, RTT and USB Controller.

28.11 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.

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 or UPLLCK (CSS = 2 or 3), 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 or UPLLCK.

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. 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).

The user can know the status of the fault output at any time by reading the FOS bit in the PMC_SR register.

28.12 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.

   Start Up Time = 8 * MOSCXTST / SLCK = 56 Slow Clock Cycles.
   The main oscillator will be enabled (MOSCXTS bit set) after 56 Slow Clock Cycles.

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. 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 2047. 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 28.13.0.2 “Clock Switching Waveforms” on page 535.

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, UPLLCK. 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.

28.13 Clock Switching Details

28.13.0.1Master Clock Switching Timings

Table 28-1 and Table 28-2 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 28-1. 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 either the PLLA or the UPLL clock.
2. PLLCOUNT designates either PLLACOUNT or UPLLCOUNT.

Table 28-2. Clock Switching Timings between Two PLLs (Worst Case)

From / To	PLLA Clock	UPLL Clock
PLLA Clock	2.5 x PLLA Clock + 4 x SLCK + PLLACOUNT x SLCK	3 x PLLA Clock + 4 x SLCK + 1.5 x PLLA Clock
UPLL Clock	3 x UPLL Clock + 4 x SLCK + 1.5 x UPLL Clock	2.5 x UPLL Clock + 4 x SLCK + x SLCK

28.13.0.2Clock Switching Waveforms

Figure 28-5. Switch Master Clock from Slow Clock to PLLx Clock

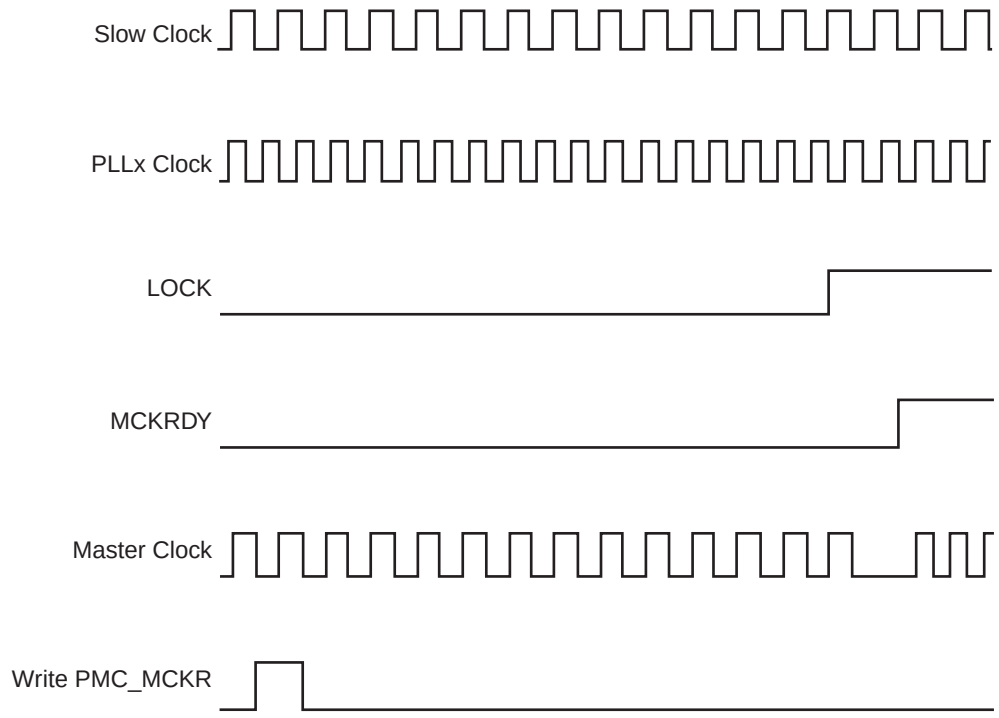


Figure 28-6. Switch Master Clock from Main Clock to Slow Clock

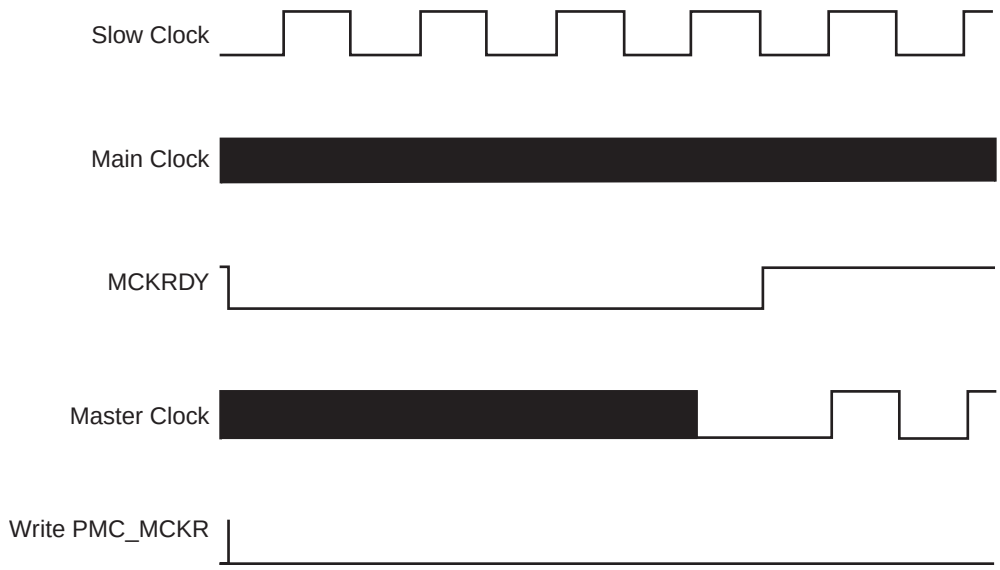


Figure 28-7. Change PLLx Programming

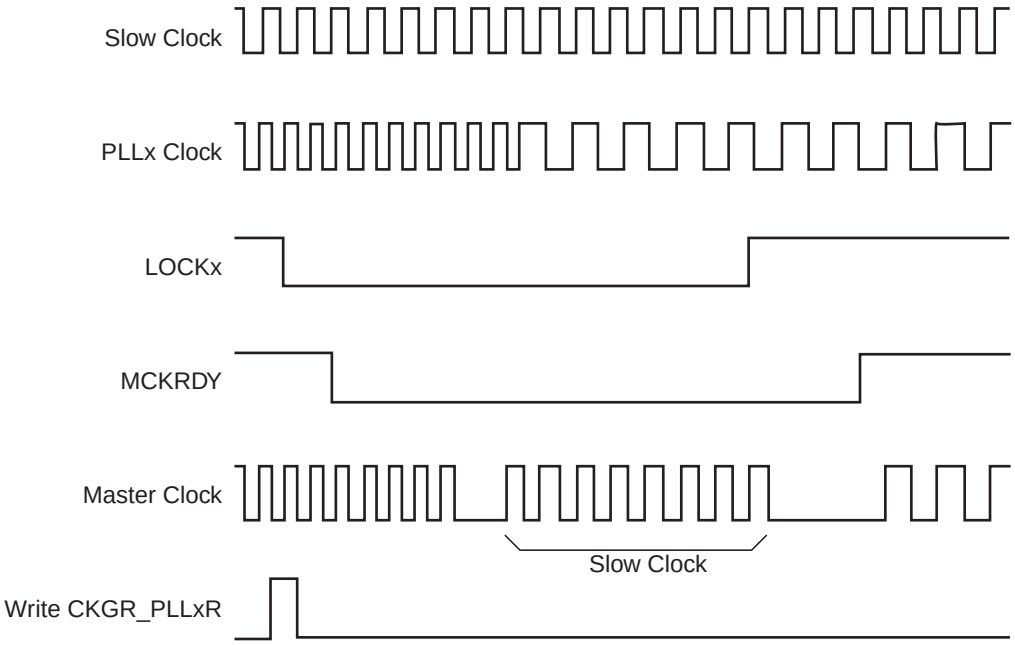
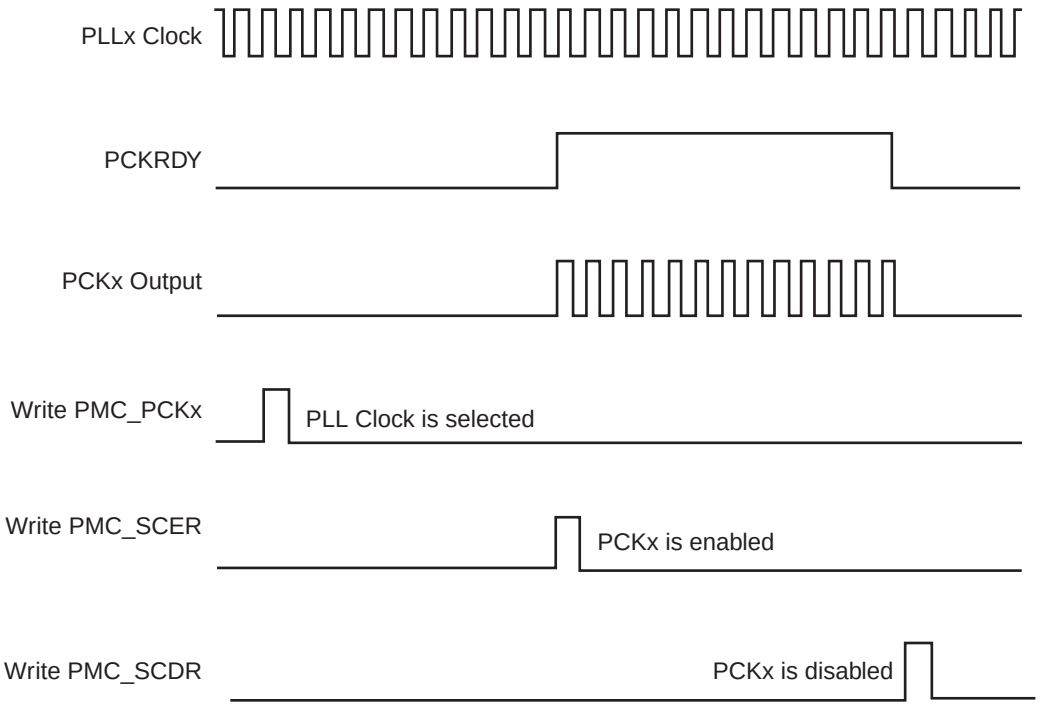


Figure 28-8. Programmable Clock Output Programming



28.14 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 WPVSRC 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 UTMI Clock Configuration Register"

"PMC Master Clock Register"

"PMC USB Clock Register"

"PMC Programmable Clock Register"

"PMC Fast Startup Mode Register"

"PMC Fast Startup Polarity Register"

"PMC Peripheral Clock Enable Register 1"

"PMC Peripheral Clock Disable Register 1"

28.15 Power Management Controller (PMC) User Interface

Table 28-3. 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	UTMI Clock Register	CKGR_UCKR	Read-write	0x1020_0800
0x0020	Main Oscillator Register	CKGR_MOR	Read-write	0x0000_0001
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read-only	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 Startup Mode Register	PMC_FSMR	Read-write	0x0000_0000
0x0074	Fast Startup 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
0x010C	Peripheral Control Register	PMC_PCR	Read-write	0x0000_0000

Note: If an offset is not listed in the table it must be considered as “reserved”.

28.15.1 PMC System Clock Enable Register

Name: PMC_SCER

Address: 0x400E0600

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
–	–	UOTGCLK	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **UOTGCLK: Enable USB OTG Clock (48 MHz, USB_48M) for UTMI**

Enable only when UOTGHS module is in low-power mode (SPDCONF =1).

0 = No effect.

1 = Enable USB_48M (to use if SPDCONF =1).

- **PCKx: Programmable Clock x Output Enable**

0 = No effect.

1 = Enables the corresponding Programmable Clock output.

28.15.2 PMC System Clock Disable Register

Name: PMC_SCDR

Address: 0x400E0604

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
–	–	UOTGCLK	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **UOTGCLK: Disable USB OTG Clock (48 MHz, USB_48M) for UTMI**

Enable only when UOTGHS module is in low-power mode (SPDCONF =1).

0: No effect.

1: Disable USB_48M (to use if SPDCONF =1).

- **PCKx: Programmable Clock x Output Disable**

0 = No effect.

1 = Disables the corresponding Programmable Clock output.

28.15.3 PMC System Clock Status Register

Name: PMC_SCSR

Address: 0x400E0608

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
–	–	UOTGCLK	–	–	–	–	–

- **UOTGCLK: USB OTG Clock (48 MHz, USB_48M) Clock Status**

0 = The 48 MHz clock (UOTGCK) of the USB OTG FS Port is disabled.

1 = The 48 MHz clock (UOTGCK) of the USB OTG FS 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.

28.15.4 PMC Peripheral Clock Enable Register 0

Name: PMC_PCER0

Address: 0x400E0610

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	PID6	PID5	PID4	PID3	PID2	–	–

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 28.15.23 ”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.

28.15.5 PMC Peripheral Clock Disable Register 0

Name: PMC_PCDR0

Address: 0x400E0614

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	PID6	PID5	PID4	PID3	PID2	-	-

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 28.15.24 ”PMC Peripheral Clock Disable Register 1”).

28.15.6 PMC Peripheral Clock Status Register 0

Name: PMC_PCSR0

Address: 0x400E0618

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	PID6	PID5	PID4	PID3	PID2	–	–

- **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 28.15.25 ”PMC Peripheral Clock Status Register 1”).

28.15.7 PMC UTMI Clock Configuration Register

Name: CKGR_UCKR

Address: 0x400E061C

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–		–	–	–	–

23	22	21	20	19	18	17	16
UPLLCOUNT				–	–	–	UPLLEN

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 “PMC Write Protect Mode Register” .

- **UPLLEN: UTMI PLL Enable**

0: The UTMI PLL is disabled.

1: The UTMI PLL is enabled.

When UPLLEN is set, the LOCKU flag is set once the UTMI PLL startup time is achieved.

- **UPLLCOUNT: UTMI PLL Start-up Time**

Specifies the number of Slow Clock cycles multiplied by 8 for the UTMI PLL start-up time.

28.15.8 PMC Clock Generator Main Oscillator Register

Name: CKGR_MOR

Address: 0x400E0620

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	–	MOSCXTBY	MOSCXTEN

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **KEY: Password**

Should be written at value 0x37. Writing any other value in this field aborts the write operation.

- **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 startup 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.

- **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 startup 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

- **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.

- **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.

28.15.9 PMC Clock Generator Main Clock Frequency Register

Name: CKGR_MCFR

Address: 0x400E0624

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	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 .

1 = The Main Oscillator has been enabled previously and MAINF value is available.

28.15.10 PMC Clock Generator PLLA Register

Name: CKGR_PLLAR

Address: 0x400E0628

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: Divider**

DIVA	Divider Selected
0	Divider output is 0
1	Divider is bypassed (DIVA=1)
2 - 255	Divider output is DIVA

- **PLLACOUNT: PLLA Counter**

Specifies the number of Slow Clock cycles x8 before the LOCKA bit is set in PMC_SR after CKGR_PLLAR is written.

- **MULA: PLLA Multiplier**

0 = The PLLA is deactivated.

1 up to 2047 = 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.

28.15.11 PMC Master Clock Register

Name: PMC_MCKR

Address: 0x400E0630

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
–	–	UPLLDIV2	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
3	UPLL_CLK	UPLL Clock is selected

- **PRES: Processor Clock Prescaler**

Value	Name	Description
0	CLK	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.

- **UPLLDIV2:UPLLDivisor by 2**

UPLLDIV2	UPLL Clock Division
0	UPLL clock frequency is divided by 1.
1	UPLL clock frequency is divided by 2.

28.15.12 PMC USB Clock Register

Name: PMC_USB

Address: 0x400E0638

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
–	–	–	–	–	–	–	USBS

This register can only be written if the WPEN bit is cleared in “PMC Write Protect Mode Register” .

- **USBS: USB Input Clock Selection**

0 = USB Clock Input is PLLA.

1 = USB Clock Input is PLLB.

- **USBDIV: Divider for USB Clock.**

USB Clock is Input clock divided by USBDIV+1.

28.15.13 PMC Programmable Clock Register

Name: PMC_PCKx

Address: 0x400E0640

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
3	UPLL_CLK	UPLL Clock is selected
4	MCK	Master Clock is selected

- **PRES: Programmable Clock Prescaler**

Value	Name	Description
0	CLK	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

28.15.14 PMC Interrupt Enable Register

Name: PMC_IER

Address: 0x400E0660

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
–	LOCKU–	–	–	MCKRDY	–	LOCKA	MOSCXTS

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Enable**
- **LOCKA: PLLA Lock Interrupt Enable**
- **MCKRDY: Master Clock Ready Interrupt Enable**
- **LOCKU: UTMI PLL Lock 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**

28.15.15 PMC Interrupt Disable Register

Name: PMC_IDR

Address: 0x400E0664

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
–	LOCKU–	–	–	MCKRDY	–	LOCKA	MOSCXTS

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Disable**
- **LOCKA: PLLA Lock Interrupt Disable**
- **MCKRDY: Master Clock Ready Interrupt Disable**
- **LOCKU: UTMI PLL Lock 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**

28.15.16 PMC Status Register

Name: PMC_SR

Address: 0x400E0668

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	LOCKU–	–	–	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.

- **LOCKU: UTMI PLL Lock Status**

0 = UTMI PLL is not locked

1 = UTMI PLL is locked.

- **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 done.

1 = Selection is in progress.

- **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 main on-chip RC oscillator clock has occurred since the last read of PMC_SR.

1 = At least one clock failure detection of the main on-chip RC oscillator clock has occurred since the last read of PMC_SR.

- **CFDS: Clock Failure Detector Status**

0 = A clock failure of the main on-chip RC oscillator clock is not detected.

1 = A clock failure of the main on-chip RC 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.

28.15.17 PMC Interrupt Mask Register

Name: PMC_IMR

Address: 0x400E066C

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
–	LOCKU–	–	–	MCKRDY	–	LOCKA	MOSCXTS

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Mask**
- **LOCKA: PLLA Lock Interrupt Mask**
- **MCKRDY: Master Clock Ready Interrupt Mask**
- **LOCKU: UTMI PLL Lock 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**

28.15.18 PMC Fast Startup Mode Register

Name: PMC_FSMR

Address: 0x400E0670

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	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 Startup 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 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.

28.15.19 PMC Fast Startup Polarity Register

Name: PMC_FSPR

Address: 0x400E0674

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 Startup Input Polarityx**

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.

28.15.20 PMC Fault Output Clear Register

Name: PMC_FOCR

Address: 0x400E0678

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.

28.15.21 PMC Write Protect Mode Register

Name: PMC_WPMR

Address: 0x400E06E4

Access: Read-write

Reset: See Table 28-3

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

- **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 Startup Mode Register”

“PMC Fast Startup Polarity Register”

“PMC Peripheral Clock Enable Register 1”

“PMC Peripheral Clock Disable Register 1”

- **WPKEY: Write Protect KEY**

Should be written at value 0x504D43 (“PMC” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

28.15.22 PMC Write Protect Status Register

Name: PMC_WPSR

Address: 0x400E06E8

Access: Read-only

Reset: See Table 28-3

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 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.

28.15.23 PMC Peripheral Clock Enable Register 1

Name: PMC_PCER1

Address: 0x400E0700

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
–	–	–	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.

28.15.24 PMC Peripheral Clock Disable Register 1

Name: PMC_PCDR1

Address: 0x400E0704

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
–	–	–	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” on page 561.

- **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.

28.15.25 PMC Peripheral Clock Status Register 1

Name: PMC_PCSR1

Address: 0x400E0708

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
–	–	–	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.

28.15.26 PMC Peripheral Control Register

Name: PMC_PCR

Address: 0x400E070C

Access: Read-write

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

- **PID: Peripheral ID**

Peripheral ID selection from PID2 to PID63

PID2 to PID63 refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

Not all PID can be configured with divided clock.

Only the following PID can be configured with divided clock: CAN0, CAN1.

- **CMD: Command**

0 = Read mode.

1 = Write mode.

- **DIV: Divisor Value**

Value	Name	Description
0	PERIPH_DIV_MCK	Peripheral clock is MCK
1	PERIPH_DIV2_MCK	Peripheral clock is MCK/2
2	PERIPH_DIV4_MCK	Peripheral clock is MCK/4

DIV must not be changed while peripheral is in use or when the peripheral clock is enabled.

To change the clock division factor (DIV) of a peripheral, its clock must first be disabled by writing either EN to 0 for the corresponding PID (DIV must be kept the same if this method is used), or writing to PMC_PCDR register. Then a second write must be performed into PMC_PCR with the new value of DIV and a third write must be performed to enable the peripheral clock (either by using PMC_PCR or PMC_PCER register).

- **EN: Enable**

0 = Selected Peripheral clock is disabled.

1 = Selected Peripheral clock is enabled.

29. Chip Identifier (CHIPID)

29.1 Description

Chip Identifier 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.

29.2 Embedded Characteristics

- Chip ID Registers
  - Identification of the Device Revision, Sizes of the Embedded Memories, Set of Peripherals, Embedded Processor

Table 29-1. ATSAM3A/X Chip IDs Register

Chip Name	CHIPID_CIDR	CHIPID_EXID
ATSAM3X8H(1) (Rev A)	0x286E0A60	0x0
ATSAM3X8E (Rev A)	0x285E0A60	0x0
ATSAM3X4E (Rev A)	0x285B0960	0x0
ATSAM3X8C (Rev A)	0x284E0A60	0x0
ATSAM3X4C (Rev A)	0x284B0960	0x0
ATSAM3A8C (Rev A)	0x283E0A60	0x0
ATSAM3A4C (Rev A)	0x283B0960	0x0

Note: 1. This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

29.3 Chip Identifier (CHIPID) User Interface

Table 29-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0	Chip ID Register	CHIPID_CIDR	Read-only	–
0x4	Chip ID Extension Register	CHIPID_EXID	Read-only	–

29.3.1 Chip ID Register

Name: CHIPID_CIDR

Address: 0x400E0940

Access: Read-only

31	30	29	28	27	26	25	24
EXT	NVPTYP			ARCH			

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

- **NVPSIZ: Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8K bytes
2	16K	16K bytes
3	32K	32K bytes
4		Reserved
5	64K	64K bytes
6		Reserved
7	128K	128K bytes
8		Reserved
9	256K	256K bytes
10	512K	512K bytes
11		Reserved
12	1024K	1024K bytes

Value	Name	Description
13		Reserved
14	2048K	2048K bytes
15		Reserved

- **NVPSIZ2 Second Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8K bytes
2	16K	16K bytes
3	32K	32K bytes
4		Reserved
5	64K	64K bytes
6		Reserved
7	128K	128K bytes
8		Reserved
9	256K	256K bytes
10	512K	512K bytes
11		Reserved
12	1024K	1024K bytes
13		Reserved
14	2048K	2048K bytes
15		Reserved

- **SRAMSIZ: Internal SRAM Size**

Value	Name	Description
0	48K	48K bytes
1	1K	1K bytes
2	2K	2K bytes
3	6K	6K bytes
4	24K	24K bytes
5	4K	4K bytes
6	80K	80K bytes
7	160K	160K bytes
8	8K	8K bytes
9	16K	16K bytes
10	32K	32K bytes
11	64K	64K bytes
12	128K	128K bytes

Value	Name	Description
13	256K	256K bytes
14	96K	96K bytes
15	512K	512K bytes

- **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
0x40	AT91x40	AT91x40 Series
0x42	AT91x42	AT91x42 Series
0x55	AT91x55	AT91x55 Series
0x60	AT91SAM7Axx	AT91SAM7Axx Series
0x61	AT91SAM7AQxx	AT91SAM7AQxx Series
0x63	AT91x63	AT91x63 Series
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 (217-pin version)
0x88	SAM3SxA	SAM3SxA Series (48-pin version)
0x89	SAM3SxB	SAM3SxB Series (64-pin version)
0x8A	SAM3SxC	SAM3SxC Series (100-pin version)
0x92	AT91x92	AT91x92 Series
0x93	SAM3NxA	SAM3NxA Series (48-pin version)
0x94	SAM3NxB	SAM3NxB Series (64-pin version)
0x95	SAM3NxC	SAM3NxC Series (100-pin version)
0x98	SAM3SDxA	SAM3SDxA Series (48-pin version)
0x99	SAM3SDxB	SAM3SDxB Series (64-pin version)

Value	Name	Description
0x9A	SAM3SDxC	SAM3SDxC Series (100-pin version)
0xA5	SAM5A	SAM5A
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 NVPSIZ is ROM size NVPSIZ2 is Flash size

- **EXT: Extension Flag**

0 = Chip ID has a single register definition without extension

1 = An extended Chip ID exists.

29.3.2 Chip ID Extension Register

Name: CHIPID_EXID

Address: 0x400E0944

Access: Read-only

31	30	29	28	27	26	25	24
EXID							

23	22	21	20	19	18	17	16
EXID							

15	14	13	12	11	10	9	8
EXID							

7	6	5	4	3	2	1	0
EXID							

- **EXID: Chip ID Extension**

Reads 0 if the bit EXT in CHIPID_CIDR is 0.

30. Synchronous Serial Controller (SSC)

30.1 Description

The Atmel Synchronous Serial Controller (SSC) provides a synchronous communication link with external devices. It supports many serial synchronous communication protocols generally used in audio and telecom applications such as I2S, Short Frame Sync, Long Frame Sync, etc.

The SSC contains an independent receiver and transmitter and a common clock divider. The receiver and the transmitter each interface with three signals: the TD/RD signal for data, the TK/RK signal for the clock and the TF/RF signal for the Frame Sync. The transfers can be programmed to start automatically or on different events detected on the Frame Sync signal.

The SSC's high-level of programmability and its use of DMA permit a continuous high bit rate data transfer without processor intervention.

Featuring connection to the DMA, the SSC permits interfacing with low processor overhead to the following:

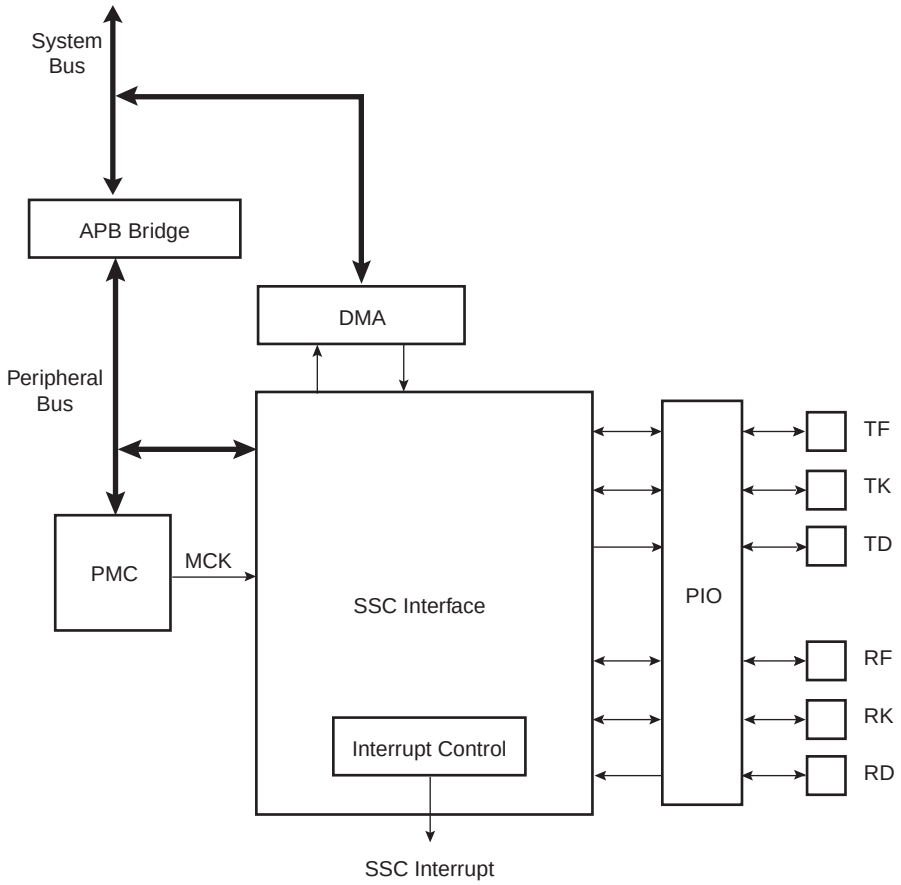
- CODEC's in master or slave mode
- DAC through dedicated serial interface, particularly I2S
- Magnetic card reader

30.2 Embedded Characteristics

- Provides Serial Synchronous Communication Links Used in Audio and Telecom Applications
- Contains an Independent Receiver and Transmitter and a Common Clock Divider
- Interfaced with the DMA Controller (DMAC) to Reduce Processor Overhead
- Offers a Configurable Frame Sync and Data Length
- Receiver and Transmitter Can be Programmed to Start Automatically or on Detection of Different Events on the Frame Sync Signal
- Receiver and Transmitter Include a Data Signal, a Clock Signal and a Frame Synchronization Signal

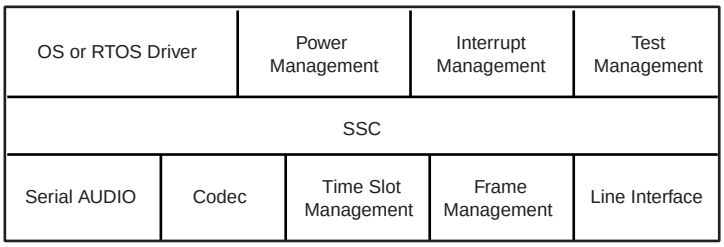
30.3 Block Diagram

Figure 30-1. Block Diagram



30.4 Application Block Diagram

Figure 30-2. Application Block Diagram



30.5 Pin Name List

Table 30-1. I/O Lines Description

Pin Name	Pin Description	Type
RF	Receiver Frame Synchro	Input/Output
RK	Receiver Clock	Input/Output
RD	Receiver Data	Input
TF	Transmitter Frame Synchro	Input/Output
TK	Transmitter Clock	Input/Output
TD	Transmitter Data	Output

30.6 Product Dependencies

30.6.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines.

Before using the SSC receiver, the PIO controller must be configured to dedicate the SSC receiver I/O lines to the SSC peripheral mode.

Before using the SSC transmitter, the PIO controller must be configured to dedicate the SSC transmitter I/O lines to the SSC peripheral mode.

Table 30-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
SSC	RD	PB18	A
SSC	RF	PB17	A
SSC	RK	PB19	A
SSC	TD	PA16	B
SSC	TF	PA15	B
SSC	TK	PA14	B

30.6.2 Power Management

The SSC is not continuously clocked. The SSC interface may be clocked through the Power Management Controller (PMC), therefore the programmer must first configure the PMC to enable the SSC clock.

30.6.3 Interrupt

The SSC interface has an interrupt line connected to the Nested Vector Interrupt Controller (NVIC). Handling interrupts requires programming the NVIC before configuring the SSC.

All SSC interrupts can be enabled/disabled configuring the SSC Interrupt mask register. Each pending and unmasked SSC interrupt will assert the SSC interrupt line. The SSC interrupt service routine can get the interrupt origin by reading the SSC interrupt status register.

Table 30-3. Peripheral IDs

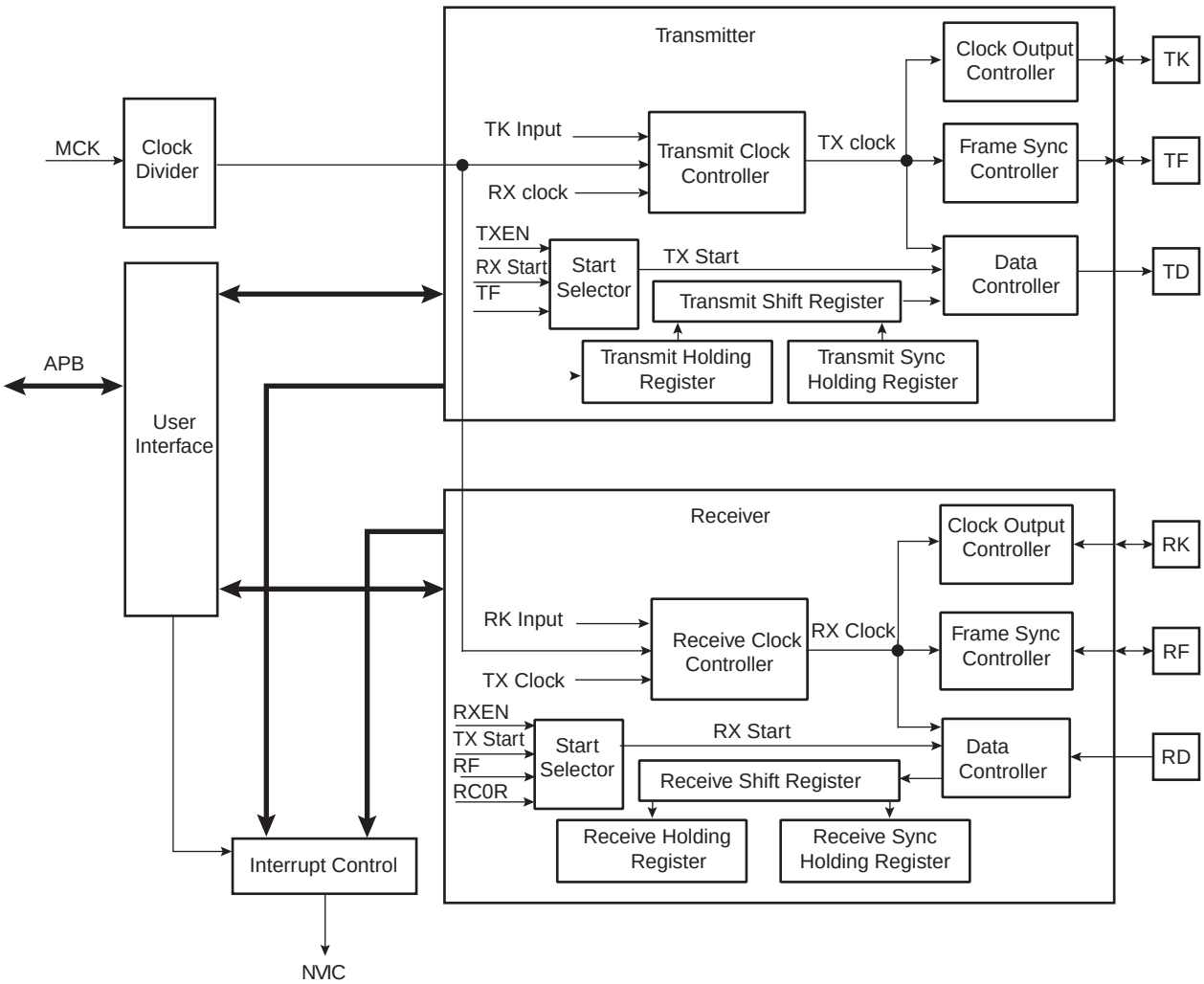
Instance	ID
SSC	26

30.7 Functional Description

This chapter contains the functional description of the following: SSC Functional Block, Clock Management, Data format, Start, Transmitter, Receiver and Frame Sync.

The receiver and transmitter operate separately. However, they can work synchronously by programming the receiver to use the transmit clock and/or to start a data transfer when transmission starts. Alternatively, this can be done by programming the transmitter to use the receive clock and/or to start a data transfer when reception starts. The transmitter and the receiver can be programmed to operate with the clock signals provided on either the TK or RK pins. This allows the SSC to support many slave-mode data transfers. The maximum clock speed allowed on the TK and RK pins is the master clock divided by 2.

Figure 30-3. SSC Functional Block Diagram



30.7.1 Clock Management

The transmitter clock can be generated by:

- an external clock received on the TK I/O pad
- the receiver clock
- the internal clock divider

The receiver clock can be generated by:

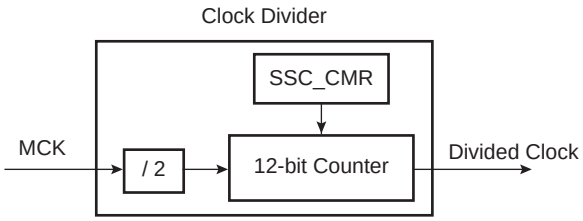
- an external clock received on the RK I/O pad
- the transmitter clock
- the internal clock divider

Furthermore, the transmitter block can generate an external clock on the TK I/O pad, and the receiver block can generate an external clock on the RK I/O pad.

This allows the SSC to support many Master and Slave Mode data transfers.

30.7.1.1 Clock Divider

Figure 30-4. Divided Clock Block Diagram



The Master Clock divider is determined by the 12-bit field DIV counter and comparator (so its maximal value is 4095) in the Clock Mode Register SSC_CMR, allowing a Master Clock division by up to 8190. The Divided Clock is provided to both the Receiver and Transmitter. When this field is programmed to 0, the Clock Divider is not used and remains inactive.

When DIV is set to a value equal to or greater than 1, the Divided Clock has a frequency of Master Clock divided by 2 times DIV. Each level of the Divided Clock has a duration of the Master Clock multiplied by DIV. This ensures a 50% duty cycle for the Divided Clock regardless of whether the DIV value is even or odd.

Figure 30-5. Divided Clock Generation

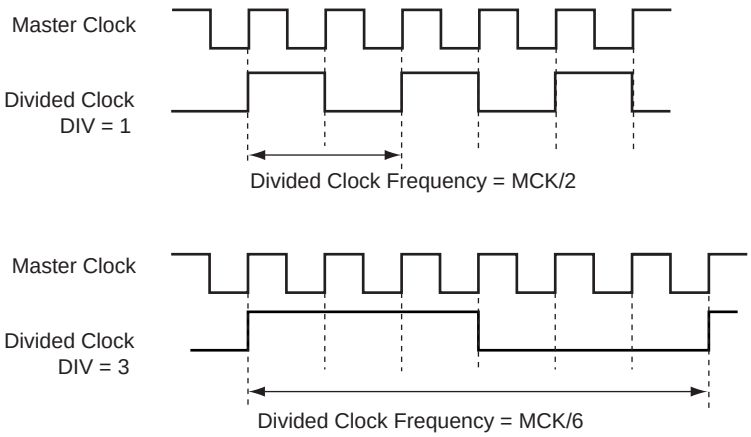


Table 30-4.

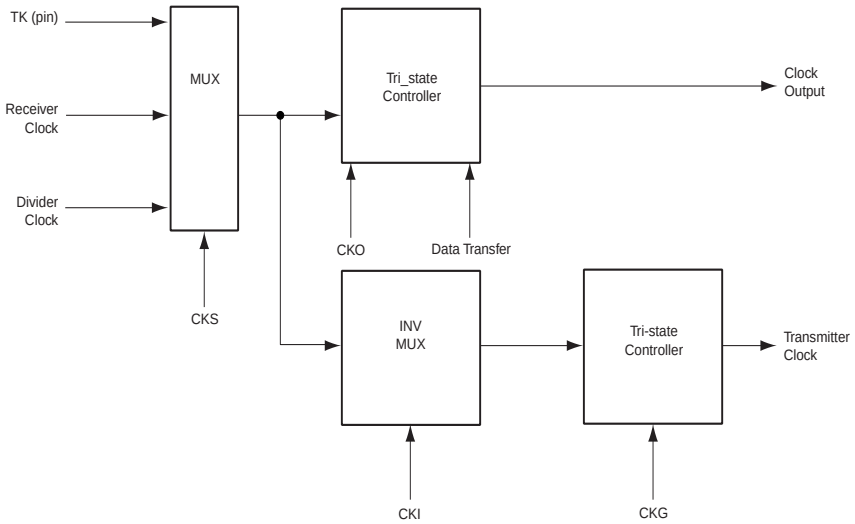
Maximum	Minimum
MCK / 2	MCK / 8190

30.7.1.2 Transmitter Clock Management

The transmitter clock is generated from the receiver clock or the divider clock or an external clock scanned on the TK I/O pad. The transmitter clock is selected by the CKS field in SSC_TCMR (Transmit Clock Mode Register). Transmit Clock can be inverted independently by the CKI bits in SSC_TCMR.

The transmitter can also drive the TK I/O pad continuously or be limited to the actual data transfer. The clock output is configured by the SSC_TCMR register. The Transmit Clock Inversion (CKI) bits have no effect on the clock outputs. Programming the TCMR register to select TK pin (CKS field) and at the same time Continuous Transmit Clock (CKO field) might lead to unpredictable results.

Figure 30-6. Transmitter Clock Management

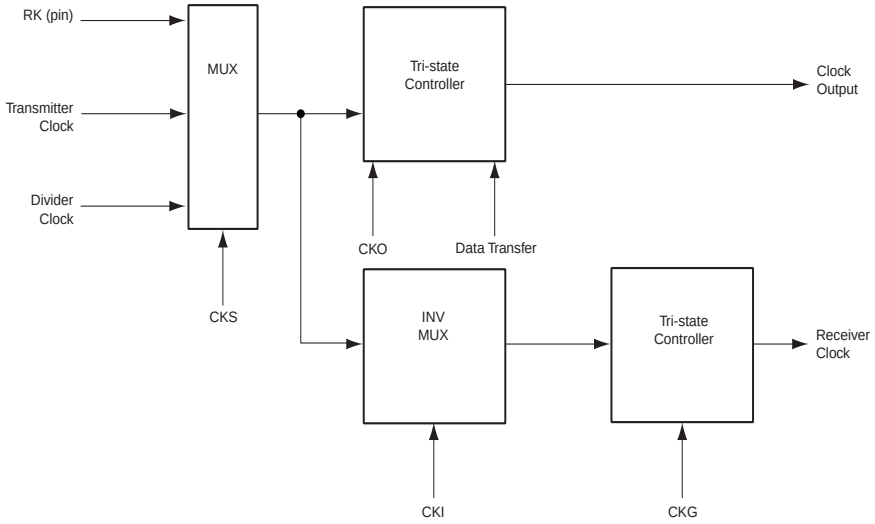


30.7.1.3 Receiver Clock Management

The receiver clock is generated from the transmitter clock or the divider clock or an external clock scanned on the RK I/O pad. The Receive Clock is selected by the CKS field in SSC_RCMR (Receive Clock Mode Register). Receive Clocks can be inverted independently by the CKI bits in SSC_RCMR.

The receiver can also drive the RK I/O pad continuously or be limited to the actual data transfer. The clock output is configured by the SSC_RCMR register. The Receive Clock Inversion (CKI) bits have no effect on the clock outputs. Programming the RCMR register to select RK pin (CKS field) and at the same time Continuous Receive Clock (CKO field) can lead to unpredictable results.

Figure 30-7. Receiver Clock Management



30.7.1.4 Serial Clock Ratio Considerations

The Transmitter and the Receiver can be programmed to operate with the clock signals provided on either the TK or RK pins. This allows the SSC to support many slave-mode data transfers. In this case, the maximum clock speed allowed on the RK pin is:

- Master Clock divided by 2 if Receiver Frame Synchro is input
- Master Clock divided by 3 if Receiver Frame Synchro is output

In addition, the maximum clock speed allowed on the TK pin is:

- Master Clock divided by 6 if Transmit Frame Synchro is input
- Master Clock divided by 2 if Transmit Frame Synchro is output

30.7.2 Transmitter Operations

A transmitted frame is triggered by a start event and can be followed by synchronization data before data transmission.

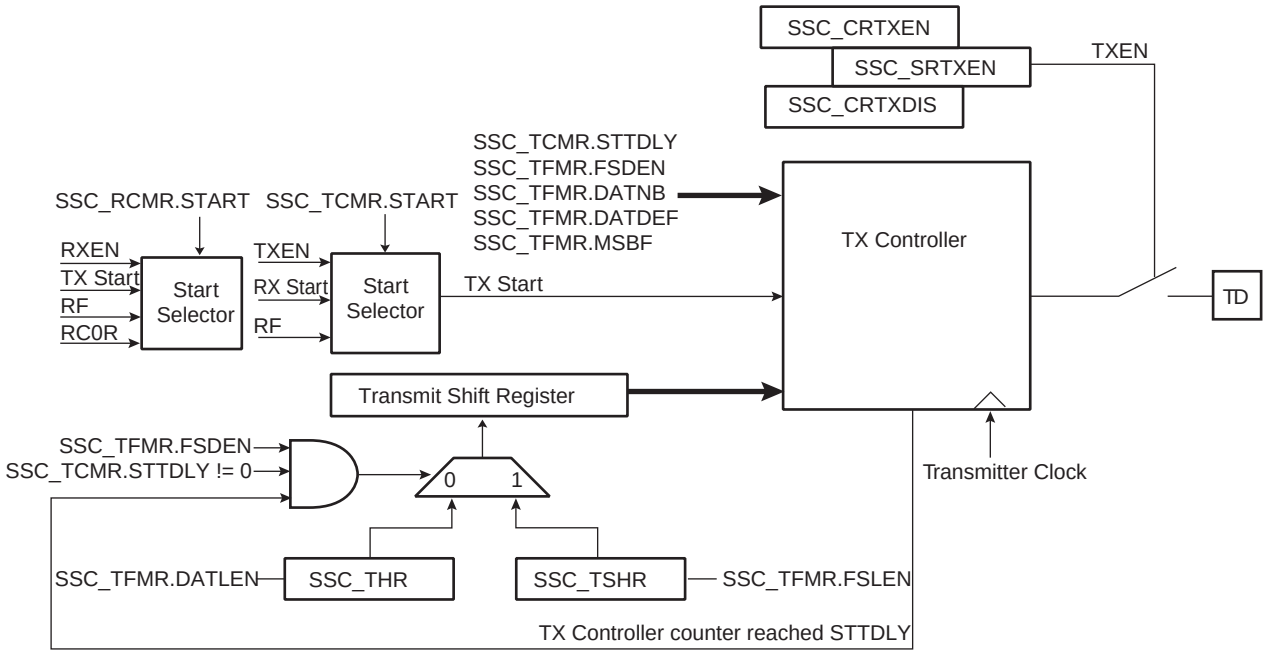
The start event is configured by setting the Transmit Clock Mode Register (SSC_TCMR). See “Start” on page 582.

The frame synchronization is configured setting the Transmit Frame Mode Register (SSC_TFMR). See “Frame Sync” on page 584.

To transmit data, the transmitter uses a shift register clocked by the transmitter clock signal and the start mode selected in the SSC_TCMR. Data is written by the application to the SSC_THR register then transferred to the shift register according to the data format selected.

When both the SSC_THR and the transmit shift register are empty, the status flag TXEMPTY is set in SSC_SR. When the Transmit Holding register is transferred in the Transmit shift register, the status flag TXRDY is set in SSC_SR and additional data can be loaded in the holding register.

Figure 30-8. Transmitter Block Diagram



30.7.3 Receiver Operations

A received frame is triggered by a start event and can be followed by synchronization data before data transmission.

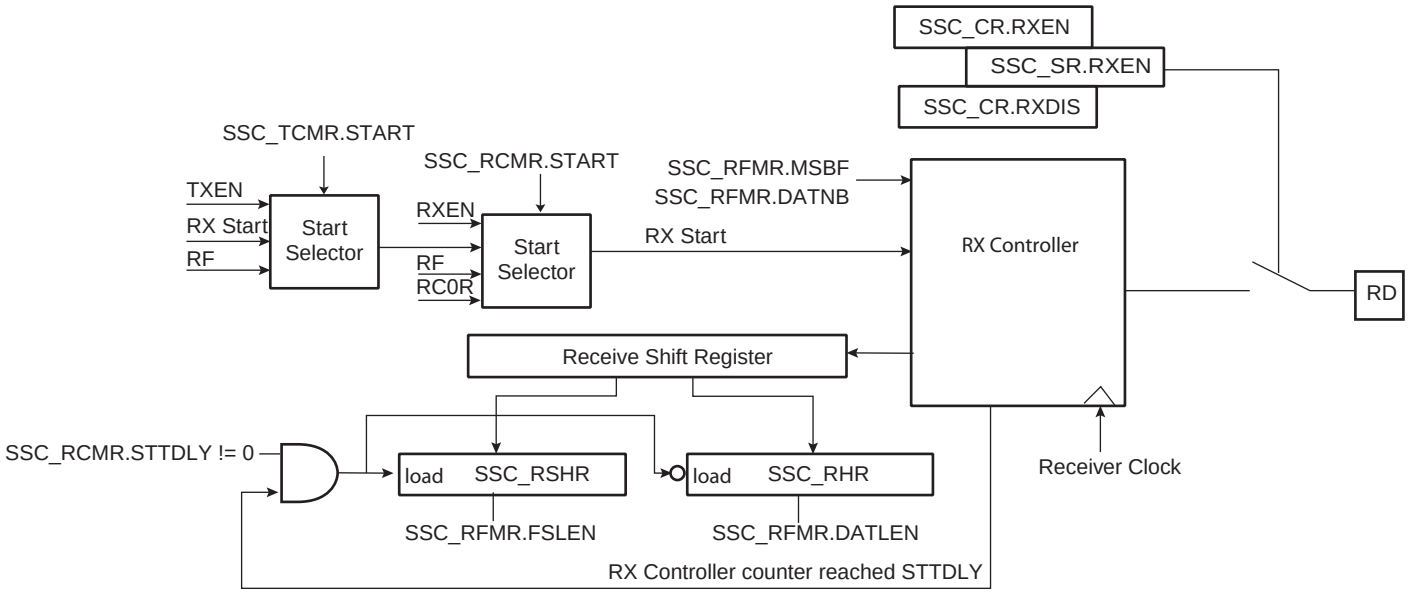
The start event is configured setting the Receive Clock Mode Register (SSC_RCMR). See “Start” on page 582.

The frame synchronization is configured setting the Receive Frame Mode Register (SSC_RFMR). See “Frame Sync” on page 584.

The receiver uses a shift register clocked by the receiver clock signal and the start mode selected in the SSC_RCMR. The data is transferred from the shift register depending on the data format selected.

When the receiver shift register is full, the SSC transfers this data in the holding register, the status flag RXRDY is set in SSC_SR and the data can be read in the receiver holding register. If another transfer occurs before read of the RHR register, the status flag OVERUN is set in SSC_SR and the receiver shift register is transferred in the RHR register.

Figure 30-9. Receiver Block Diagram



30.7.4 Start

The transmitter and receiver can both be programmed to start their operations when an event occurs, respectively in the Transmit Start Selection (START) field of SSC_TCMR and in the Receive Start Selection (START) field of SSC_RCMR.

Under the following conditions the start event is independently programmable:

- Continuous. In this case, the transmission starts as soon as a word is written in SSC_THR and the reception starts as soon as the Receiver is enabled.
- Synchronously with the transmitter/receiver
- On detection of a falling/rising edge on TF/RF
- On detection of a low level/high level on TF/RF
- On detection of a level change or an edge on TF/RF

A start can be programmed in the same manner on either side of the Transmit/Receive Clock Register (RCMR/TCMR). Thus, the start could be on TF (Transmit) or RF (Receive).

Moreover, the Receiver can start when data is detected in the bit stream with the Compare Functions.

Detection on TF/RF input/output is done by the field FSOS of the Transmit/Receive Frame Mode Register (TFMR/RFMR).

Figure 30-10. Transmit Start Mode

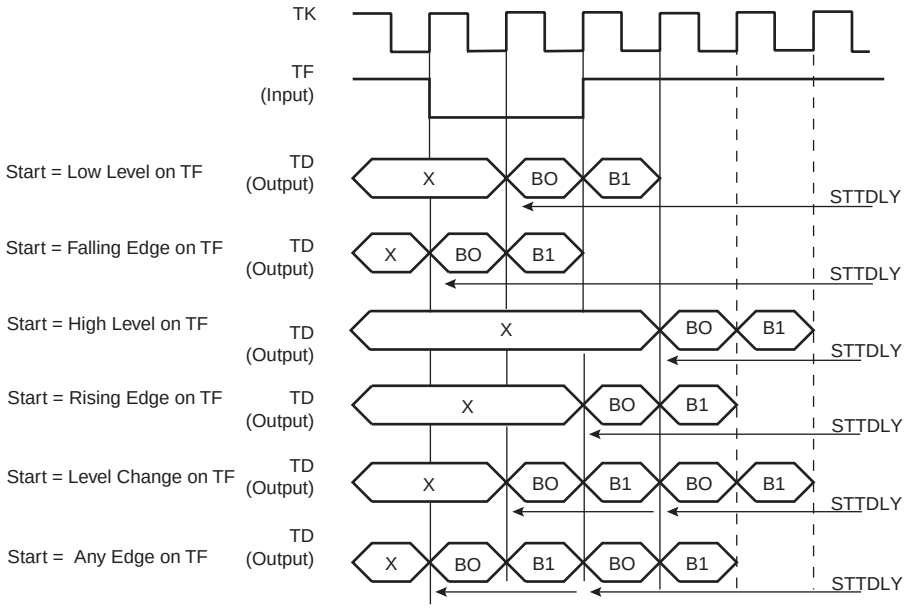
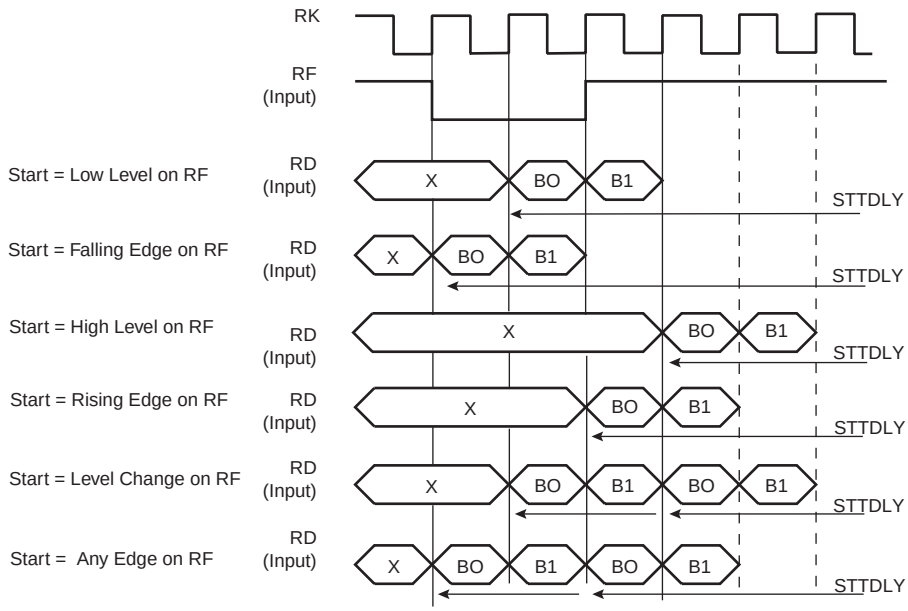


Figure 30-11. Receive Pulse/Edge Start Modes



30.7.5 Frame Sync

The Transmitter and Receiver Frame Sync pins, TF and RF, can be programmed to generate different kinds of frame synchronization signals. The Frame Sync Output Selection (FSOS) field in the Receive Frame Mode Register (SSC_RFMR) and in the Transmit Frame Mode Register (SSC_TFMR) are used to select the required waveform.

- Programmable low or high levels during data transfer are supported.
- Programmable high levels before the start of data transfers or toggling are also supported.

If a pulse waveform is selected, the Frame Sync Length (FSLEN) field in SSC_RFMR and SSC_TFMR programs the length of the pulse, from 1 bit time up to 256 bit time.

The periodicity of the Receive and Transmit Frame Sync pulse output can be programmed through the Period Divider Selection (PERIOD) field in SSC_RCMR and SSC_TCMR.

30.7.5.1 Frame Sync Data

Frame Sync Data transmits or receives a specific tag during the Frame Sync signal.

During the Frame Sync signal, the Receiver can sample the RD line and store the data in the Receive Sync Holding Register and the transmitter can transfer Transmit Sync Holding Register in the Shifter Register. The data length to be sampled/shifted out during the Frame Sync signal is programmed by the FSLEN field in SSC_RFMR/SSC_TFMR and has a maximum value of 16.

Concerning the Receive Frame Sync Data operation, if the Frame Sync Length is equal to or lower than the delay between the start event and the actual data reception, the data sampling operation is performed in the Receive Sync Holding Register through the Receive Shift Register.

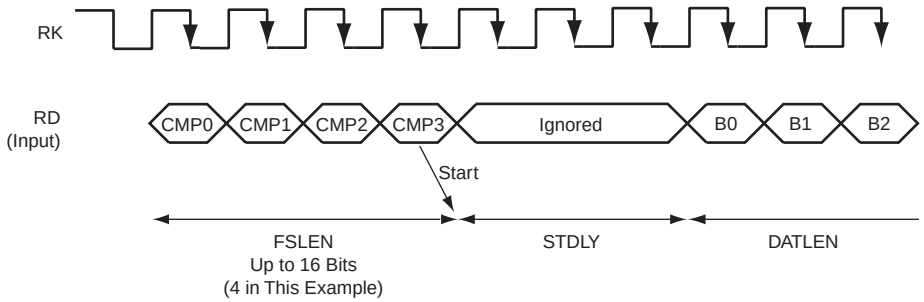
The Transmit Frame Sync Operation is performed by the transmitter only if the bit Frame Sync Data Enable (FSDEN) in SSC_TFMR is set. If the Frame Sync length is equal to or lower than the delay between the start event and the actual data transmission, the normal transmission has priority and the data contained in the Transmit Sync Holding Register is transferred in the Transmit Register, then shifted out.

30.7.5.2 Frame Sync Edge Detection

The Frame Sync Edge detection is programmed by the FSEDGE field in SSC_RFMR/SSC_TFMR. This sets the corresponding flags RXSYN/TXSYN in the SSC Status Register (SSC_SR) on frame synchro edge detection (signals RF/TF).

30.7.6 Receive Compare Modes

Figure 30-12. Receive Compare Modes



30.7.6.1 Compare Functions

Length of the comparison patterns (Compare 0, Compare 1) and thus the number of bits they are compared to is defined by FSLEN, but with a maximum value of 16 bits. Comparison is always done by comparing the last bits received with the comparison pattern. Compare 0 can be one start event of the Receiver. In this case, the receiver compares at each new sample the last bits received at the Compare 0 pattern contained in the Compare 0

Register (SSC_RC0R). When this start event is selected, the user can program the Receiver to start a new data transfer either by writing a new Compare 0, or by receiving continuously until Compare 1 occurs. This selection is done with the bit (STOP) in SSC_RCMR.

30.7.7 Data Format

The data framing format of both the transmitter and the receiver are programmable through the Transmitter Frame Mode Register (SSC_TFMR) and the Receiver Frame Mode Register (SSC_RFMR). In either case, the user can independently select:

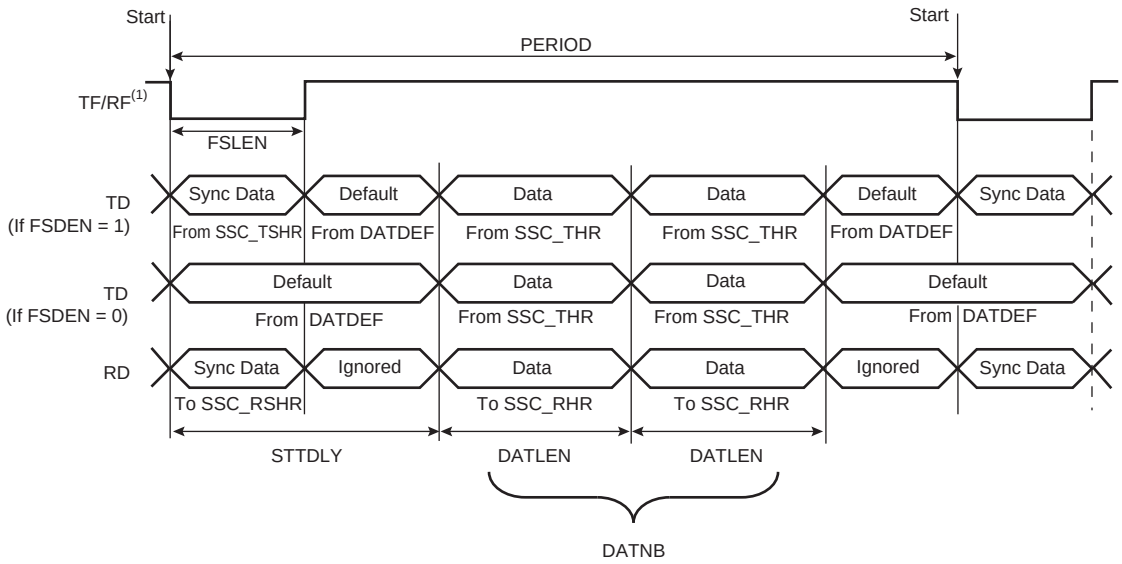
- the event that starts the data transfer (START)
- the delay in number of bit periods between the start event and the first data bit (STTDLY)
- the length of the data (DATLEN)
- the number of data to be transferred for each start event (DATNB).
- the length of synchronization transferred for each start event (FSLEN)
- the bit sense: most or lowest significant bit first (MSBF)

Additionally, the transmitter can be used to transfer synchronization and select the level driven on the TD pin while not in data transfer operation. This is done respectively by the Frame Sync Data Enable (FSDEN) and by the Data Default Value (DATDEF) bits in SSC_TFMR.

Table 30-5. Data Frame Registers

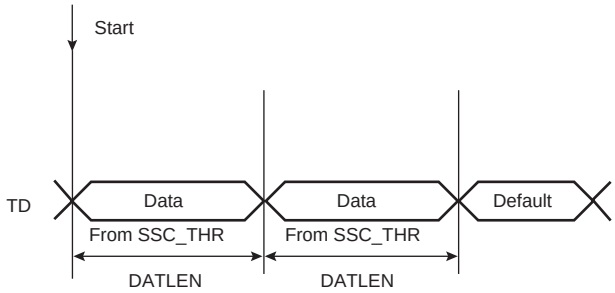
Transmitter	Receiver	Field	Length	Comment
SSC_TFMR	SSC_RFMR	DATLEN	Up to 32	Size of word
SSC_TFMR	SSC_RFMR	DATNB	Up to 16	Number of words transmitted in frame
SSC_TFMR	SSC_RFMR	MSBF		Most significant bit first
SSC_TFMR	SSC_RFMR	FSLEN	Up to 16	Size of Synchro data register
SSC_TFMR		DATDEF	0 or 1	Data default value ended
SSC_TFMR		FSDEN		Enable send SSC_TSHR
SSC_TCMR	SSC_RCMR	PERIOD	Up to 512	Frame size
SSC_TCMR	SSC_RCMR	STTDLY	Up to 255	Size of transmit start delay

Figure 30-13. Transmit and Receive Frame Format in Edge/Pulse Start Modes



Note: 1. Example of input on falling edge of TF/RF.

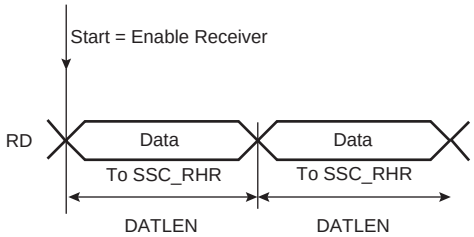
Figure 30-14. Transmit Frame Format in Continuous Mode



Start: 1. TXEMPTY set to 1
2. Write into the SSC_THR

Note: 1. STTDLY is set to 0. In this example, SSC_THR is loaded twice. FSDEN value has no effect on the transmission. SyncData cannot be output in continuous mode.

Figure 30-15. Receive Frame Format in Continuous Mode



Note: 1. STTDLY is set to 0.

30.7.8 Loop Mode

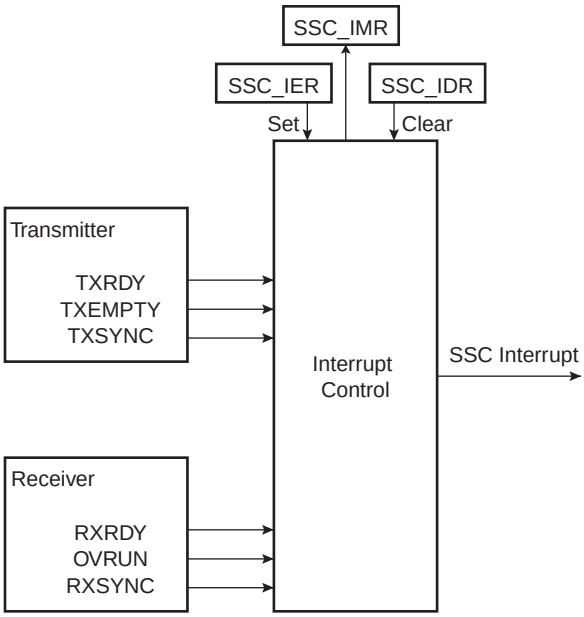
The receiver can be programmed to receive transmissions from the transmitter. This is done by setting the Loop Mode (LOOP) bit in SSC_RFMR. In this case, RD is connected to TD, RF is connected to TF and RK is connected to TK.

30.7.9 Interrupt

Most bits in SSC_SR have a corresponding bit in interrupt management registers.

The SSC can be programmed to generate an interrupt when it detects an event. The interrupt is controlled by writing SSC_IER (Interrupt Enable Register) and SSC_IDR (Interrupt Disable Register) These registers enable and disable, respectively, the corresponding interrupt by setting and clearing the corresponding bit in SSC_IMR (Interrupt Mask Register), which controls the generation of interrupts by asserting the SSC interrupt line connected to the NVIC.

Figure 30-16. Interrupt Block Diagram



30.8 SSC Application Examples

The SSC can support several serial communication modes used in audio or high speed serial links. Some standard applications are shown in the following figures. All serial link applications supported by the SSC are not listed here.

Figure 30-17. Audio Application Block Diagram

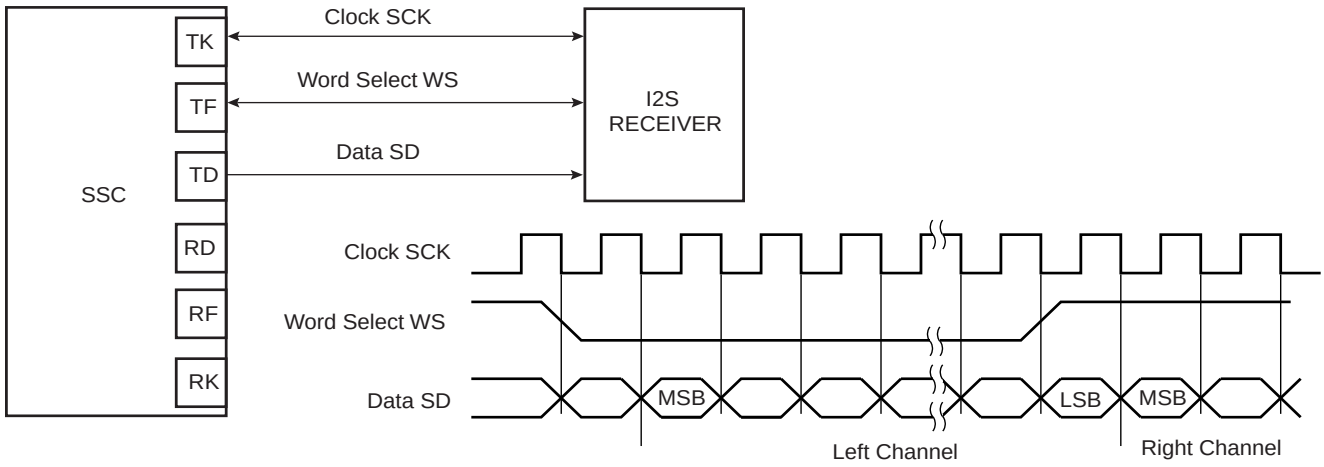


Figure 30-18. Codec Application Block Diagram

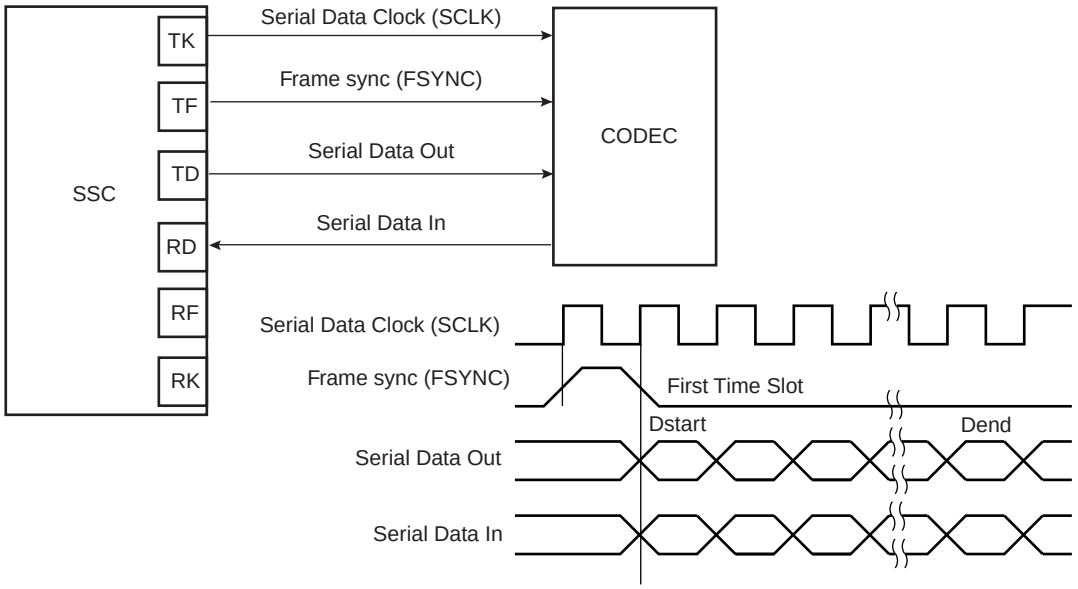
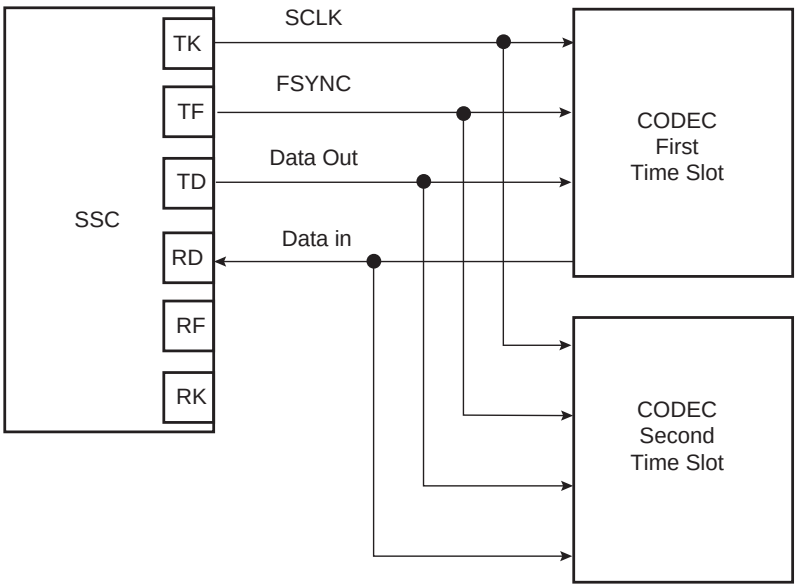
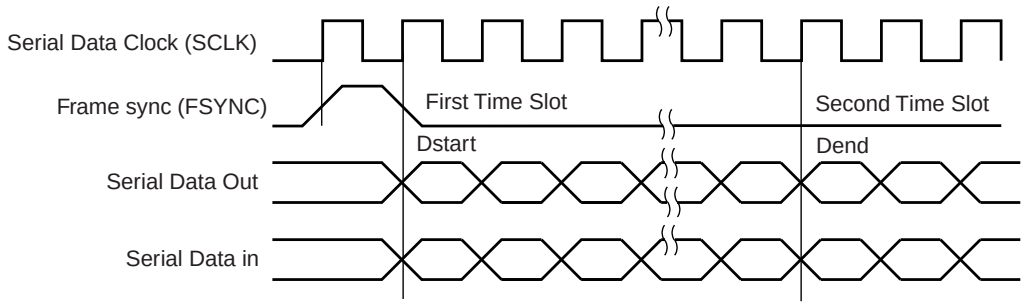


Figure 30-19. Time Slot Application Block Diagram





30.8.1 Write Protection Registers

To prevent any single software error that may corrupt SSC behavior, certain address spaces can be write-protected by setting the WPEN bit in the "SSC Write Protect Mode Register" (SSC_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the SSC Write Protect Status Register (US_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the SSC Write Protect Mode Register (SSC_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- "SSC Clock Mode Register" on page 593
- "SSC Receive Clock Mode Register" on page 594
- "SSC Receive Frame Mode Register" on page 596
- "SSC Transmit Clock Mode Register" on page 598
- "SSC Transmit Frame Mode Register" on page 600
- "SSC Receive Compare 0 Register" on page 606
- "SSC Receive Compare 1 Register" on page 607

30.9 Synchronous Serial Controller (SSC) User Interface

Table 30-6. Register Mapping

Offset	Register	Name	Access	Reset
0x0	Control Register	SSC_CR	Write-only	–
0x4	Clock Mode Register	SSC_CMR	Read-write	0x0
0x8	Reserved	–	–	–
0xC	Reserved	–	–	–
0x10	Receive Clock Mode Register	SSC_RCMR	Read-write	0x0
0x14	Receive Frame Mode Register	SSC_RFMR	Read-write	0x0
0x18	Transmit Clock Mode Register	SSC_TCMR	Read-write	0x0
0x1C	Transmit Frame Mode Register	SSC_TFMR	Read-write	0x0
0x20	Receive Holding Register	SSC_RHR	Read-only	0x0
0x24	Transmit Holding Register	SSC_THR	Write-only	–
0x28	Reserved	–	–	–
0x2C	Reserved	–	–	–
0x30	Receive Sync. Holding Register	SSC_RSHR	Read-only	0x0
0x34	Transmit Sync. Holding Register	SSC_TSHR	Read-write	0x0
0x38	Receive Compare 0 Register	SSC_RC0R	Read-write	0x0
0x3C	Receive Compare 1 Register	SSC_RC1R	Read-write	0x0
0x40	Status Register	SSC_SR	Read-only	0x000000CC
0x44	Interrupt Enable Register	SSC_IER	Write-only	–
0x48	Interrupt Disable Register	SSC_IDR	Write-only	–
0x4C	Interrupt Mask Register	SSC_IMR	Read-only	0x0
0xE4	Write Protect Mode Register	SSC_WPMR	Read-write	0x0
0xE8	Write Protect Status Register	SSC_WPSR	Read-only	0x0
0x50-0xFC	Reserved	–	–	–
0x100- 0x124	Reserved	–	–	–

30.9.1 SSC Control Register

Name: SSC_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	–	–	–	–	–	TXDIS	TXEN
7	6	5	4	3	2	1	0
–	–	–	–	–	–	RXDIS	RXEN

- **RXEN: Receive Enable**

0 = No effect.

1 = Enables Receive if RXDIS is not set.

- **RXDIS: Receive Disable**

0 = No effect.

1 = Disables Receive. If a character is currently being received, disables at end of current character reception.

- **TXEN: Transmit Enable**

0 = No effect.

1 = Enables Transmit if TXDIS is not set.

- **TXDIS: Transmit Disable**

0 = No effect.

1 = Disables Transmit. If a character is currently being transmitted, disables at end of current character transmission.

- **SWRST: Software Reset**

0 = No effect.

1 = Performs a software reset. Has priority on any other bit in SSC_CR.

30.9.2 SSC Clock Mode Register

Name: SSC_CMR

Address: 0x40004004

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							

This register can only be written if the WPEN bit is cleared in “SSC Write Protect Mode Register” .

- **DIV: Clock Divider**

0 = The Clock Divider is not active.

Any Other Value: The Divided Clock equals the Master Clock divided by 2 times DIV. The maximum bit rate is MCK/2. The minimum bit rate is MCK/2 x 4095 = MCK/8190.

30.9.3 SSC Receive Clock Mode Register

Name: SSC_RCMR

Address: 0x40004010

Access: Read-write

31	30	29	28	27	26	25	24
PERIOD							

23	22	21	20	19	18	17	16
STTDLY							

15	14	13	12	11	10	9	8
–	–	–	STOP	START			

7	6	5	4	3	2	1	0
CKG		CKI	CKO			CKS	

This register can only be written if the WPEN bit is cleared in “SSC Write Protect Mode Register” .

- **CKS: Receive Clock Selection**

Value	Name	Description
0	MCK	Divided Clock
1	TK	TK Clock signal
2	RK	RK pin
3		Reserved

- **CKO: Receive Clock Output Mode Selection**

Value	Name	Description	RK Pin
0	NONE	None	Input-only
1	CONTINUOUS	Continuous Receive Clock	Output
2	TRANSFER	Receive Clock only during data transfers	Output
3-7		Reserved	

- **CKI: Receive Clock Inversion**

0 = The data inputs (Data and Frame Sync signals) are sampled on Receive Clock falling edge. The Frame Sync signal output is shifted out on Receive Clock rising edge.

1 = The data inputs (Data and Frame Sync signals) are sampled on Receive Clock rising edge. The Frame Sync signal output is shifted out on Receive Clock falling edge.

CKI affects only the Receive Clock and not the output clock signal.

- **CKG: Receive Clock Gating Selection**

Value	Name	Description	RK Pin
0	NONE	None	Input-only
1	CONTINUOUS	Continuous Receive Clock	Output
2	TRANSFER	Receive Clock only during data transfers	Output
3-7		Reserved	

- **START: Receive Start Selection**

Value	Name	Description
0	CONTINUOUS	Continuous, as soon as the receiver is enabled, and immediately after the end of transfer of the previous data.
1	TRANSMIT	Transmit start
2	RF_LOW	Detection of a low level on RF signal
3	RF_HIGH	Detection of a high level on RF signal
4	RF_FALLING	Detection of a falling edge on RF signal
5	RF_RISING	Detection of a rising edge on RF signal
6	RF_LEVEL	Detection of any level change on RF signal
7	RF_EDGE	Detection of any edge on RF signal
8	CMP_0	Compare 0

- **STOP: Receive Stop Selection**

0 = After completion of a data transfer when starting with a Compare 0, the receiver stops the data transfer and waits for a new compare 0.

1 = After starting a receive with a Compare 0, the receiver operates in a continuous mode until a Compare 1 is detected.

- **STTDLY: Receive Start Delay**

If STTDLY is not 0, a delay of STTDLY clock cycles is inserted between the start event and the actual start of reception. When the Receiver is programmed to start synchronously with the Transmitter, the delay is also applied.

Note: It is very important that STTDLY be set carefully. If STTDLY must be set, it should be done in relation to TAG (Receive Sync Data) reception.

- **PERIOD: Receive Period Divider Selection**

This field selects the divider to apply to the selected Receive Clock in order to generate a new Frame Sync Signal. If 0, no PERIOD signal is generated. If not 0, a PERIOD signal is generated each 2 x (PERIOD+1) Receive Clock.

30.9.4 SSC Receive Frame Mode Register

Name: SSC_RFMR

Address: 0x40004014

Access: Read-write

31	30	29	28	27	26	25	24
FSLEN_EXT	FSLEN_EXT	FSLEN_EXT	FSLEN_EXT	–	–	–	FSEDGE
23	22	21	20	19	18	17	16
–	FSOS			FSLEN			
15	14	13	12	11	10	9	8
–	–	–	–	DATNB			
7	6	5	4	3	2	1	0
MSBF	–	LOOP	DATLEN				

This register can only be written if the WPEN bit is cleared in “SSC Write Protect Mode Register” .

- **DATLEN: Data Length**

0 = Forbidden value (1-bit data length not supported).
Any other value: The bit stream contains DATLEN + 1 data bits.

- **LOOP: Loop Mode**

0 = Normal operating mode.

1 = RD is driven by TD, RF is driven by TF and TK drives RK.

- **MSBF: Most Significant Bit First**

0 = The lowest significant bit of the data register is sampled first in the bit stream.

1 = The most significant bit of the data register is sampled first in the bit stream.

- **DATNB: Data Number per Frame**

This field defines the number of data words to be received after each transfer start, which is equal to (DATNB + 1).

- **FSLEN: Receive Frame Sync Length**

This field defines the number of bits sampled and stored in the Receive Sync Data Register. When this mode is selected by the START field in the Receive Clock Mode Register, it also determines the length of the sampled data to be compared to the Compare 0 or Compare 1 register.

This field is used with FSLEN_EXT to determine the pulse length of the Receive Frame Sync signal.

Pulse length is equal to FSLEN + (FSLEN_EXT * 16) + 1 Receive Clock periods.

- **FSOS: Receive Frame Sync Output Selection**

Value	Name	Description	RF Pin
0	NONE	None	Input-only
1	NEGATIVE	Negative Pulse	Output
2	POSITIVE	Positive Pulse	Output
3	LOW	Driven Low during data transfer	Output
4	HIGH	Driven High during data transfer	Output
5	TOGGLING	Toggling at each start of data transfer	Output
6-7		Reserved	Undefined

- **FSEDGE: Frame Sync Edge Detection**

Determines which edge on Frame Sync will generate the interrupt RXSYN in the SSC Status Register.

Value	Name	Description
0	POSITIVE	Positive Edge Detection
1	NEGATIVE	Negative Edge Detection

- **FSLEN_EXT: FSLEN Field Extension**

Extends FSLEN field. For details, refer to FSLEN bit description on page 596.

30.9.5 SSC Transmit Clock Mode Register

Name: SSC_TCMR

Address: 0x40004018

Access: Read-write

31	30	29	28	27	26	25	24
PERIOD							

23	22	21	20	19	18	17	16
STTDLY							

15	14	13	12	11	10	9	8
–	–	–	–	START			

7	6	5	4	3	2	1	0
CKG		CKI	CKO			CKS	

This register can only be written if the WPEN bit is cleared in “SSC Write Protect Mode Register” .

- **CKS: Transmit Clock Selection**

Value	Name	Description
0	MCK	Divided Clock
1	TK	TK Clock signal
2	RK	RK pin
3		Reserved

- **CKO: Transmit Clock Output Mode Selection**

Value	Name	Description	TK Pin
0	NONE	None	Input-only
1	CONTINUOUS	Continuous Receive Clock	Output
2	TRANSFER	Transmit Clock only during data transfers	Output
3-7		Reserved	

- **CKI: Transmit Clock Inversion**

0 = The data outputs (Data and Frame Sync signals) are shifted out on Transmit Clock falling edge. The Frame sync signal input is sampled on Transmit clock rising edge.

1 = The data outputs (Data and Frame Sync signals) are shifted out on Transmit Clock rising edge. The Frame sync signal input is sampled on Transmit clock falling edge.

CKI affects only the Transmit Clock and not the output clock signal.

- **CKG: Transmit Clock Gating Selection**

Value	Name	Description
0	NONE	None
1	CONTINUOUS	Transmit Clock enabled only if TF Low
2	TRANSFER	Transmit Clock enabled only if TF High

- **START: Transmit Start Selection**

Value	Name	Description
0	CONTINUOUS	Continuous, as soon as a word is written in the SSC_THR Register (if Transmit is enabled), and immediately after the end of transfer of the previous data.
1	RECEIVE	Receive start
2	RF_LOW	Detection of a low level on TF signal
3	RF_HIGH	Detection of a high level on TF signal
4	RF_FALLING	Detection of a falling edge on TF signal
5	RF_RISING	Detection of a rising edge on TF signal
6	RF_LEVEL	Detection of any level change on TF signal
7	RF_EDGE	Detection of any edge on TF signal
8	CMP_0	Compare 0

- **STTDLY: Transmit Start Delay**

If STTDLY is not 0, a delay of STTDLY clock cycles is inserted between the start event and the actual start of transmission of data. When the Transmitter is programmed to start synchronously with the Receiver, the delay is also applied.

Note: STTDLY must be set carefully. If STTDLY is too short in respect to TAG (Transmit Sync Data) emission, data is emitted instead of the end of TAG.

- **PERIOD: Transmit Period Divider Selection**

This field selects the divider to apply to the selected Transmit Clock to generate a new Frame Sync Signal. If 0, no period signal is generated. If not 0, a period signal is generated at each 2 x (PERIOD+1) Transmit Clock.

30.9.6 SSC Transmit Frame Mode Register

Name: SSC_TFMR

Address: 0x4000401C

Access: Read-write

31	30	29	28	27	26	25	24
FSLEN_EXT	FSLEN_EXT	FSLEN_EXT	FSLEN_EXT	–	–	–	FSEDGE

23	22	21	20	19	18	17	16
FSDEN	FSOS			FSLEN			

15	14	13	12	11	10	9	8
–	–	–	–	DATNB			

7	6	5	4	3	2	1	0
MSBF	–	DATDEF	DATLEN				

This register can only be written if the WPEN bit is cleared in “SSC Write Protect Mode Register” .

- **DATLEN: Data Length**

0 = Forbidden value (1-bit data length not supported).

Any other value: The bit stream contains DATLEN + 1 data bits. .

- **DATDEF: Data Default Value**

This bit defines the level driven on the TD pin while out of transmission. Note that if the pin is defined as multi-drive by the PIO Controller, the pin is enabled only if the SCC TD output is 1.

- **MSBF: Most Significant Bit First**

0 = The lowest significant bit of the data register is shifted out first in the bit stream.

1 = The most significant bit of the data register is shifted out first in the bit stream.

- **DATNB: Data Number per frame**

This field defines the number of data words to be transferred after each transfer start, which is equal to (DATNB +1).

- **FSLEN: Transmit Frame Sync Length**

This field defines the length of the Transmit Frame Sync signal and the number of bits shifted out from the Transmit Sync Data Register if FSDEN is 1.

This field is used with FSLEN_EXT to determine the pulse length of the Transmit Frame Sync signal.

Pulse length is equal to FSLEN + (FSLEN_EXT * 16) + 1 Transmit Clock period.

- **FSOS: Transmit Frame Sync Output Selection**

Value	Name	Description	RF Pin
0	NONE	None	Input-only
1	NEGATIVE	Negative Pulse	Output
2	POSITIVE	Positive Pulse	Output
3	LOW	Driven Low during data transfer	Output
4	HIGH	Driven High during data transfer	Output
5	TOGGLING	Toggling at each start of data transfer	Output
6-7		Reserved	Undefined

- **FSDEN: Frame Sync Data Enable**

0 = The TD line is driven with the default value during the Transmit Frame Sync signal.

1 = SSC_TSHR value is shifted out during the transmission of the Transmit Frame Sync signal.

- **FSEDGE: Frame Sync Edge Detection**

Determines which edge on frame sync will generate the interrupt TXSYN (Status Register).

Value	Name	Description
0	POSITIVE	Positive Edge Detection
1	NEGATIVE	Negative Edge Detection

- **FSLEN_EXT: FSLEN Field Extension**

Extends FSLEN field. For details, refer to FSLEN bit description on page 600.

30.9.7 SSC Receive Holding Register

Name: SSC_RHR

Address: 0x40004020

Access: Read-only

31	30	29	28	27	26	25	24
RDAT							

23	22	21	20	19	18	17	16
RDAT							

15	14	13	12	11	10	9	8
RDAT							

7	6	5	4	3	2	1	0
RDAT							

- **RDAT: Receive Data**

Right aligned regardless of the number of data bits defined by DATLEN in SSC_RFMR.

30.9.8 SSC Transmit Holding Register

Name: SSC_THR

Address: 0x40004024

Access: Write-only

31	30	29	28	27	26	25	24
TDAT							

23	22	21	20	19	18	17	16
TDAT							

15	14	13	12	11	10	9	8
TDAT							

7	6	5	4	3	2	1	0
TDAT							

- **TDAT: Transmit Data**

Right aligned regardless of the number of data bits defined by DATLEN in SSC_TFMR.

30.9.9 SSC Receive Synchronization Holding Register

Name: SSC_RSHR

Address: 0x40004030

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
RSDAT							

7	6	5	4	3	2	1	0
RSDAT							

- **RSDAT: Receive Synchronization Data**

30.9.10 SSC Transmit Synchronization Holding Register

Name: SSC_TSHR

Address: 0x40004034

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
TSDAT							

7	6	5	4	3	2	1	0
TSDAT							

- **TSDAT: Transmit Synchronization Data**

30.9.11 SSC Receive Compare 0 Register

Name: SSC_RC0R

Address: 0x40004038

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
CP0							

7	6	5	4	3	2	1	0
CP0							

This register can only be written if the WPEN bit is cleared in “SSC Write Protect Mode Register” .

- **CP0: Receive Compare Data 0**

30.9.12 SSC Receive Compare 1 Register

Name: SSC_RC1R

Address: 0x4000403C

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
CP1							
7	6	5	4	3	2	1	0
CP1							

This register can only be written if the WPEN bit is cleared in “SSC Write Protect Mode Register” .

- **CP1: Receive Compare Data 1**

30.9.13 SSC Status Register

Name: SSC_SR

Address: 0x40004040

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	RXEN	TXEN
15	14	13	12	11	10	9	8
–	–	–	–	RXSYN	TXSYN	CP1	CP0
7	6	5	4	3	2	1	0
–	–	OVRUN	RXRDY	–	–	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready**

0 = Data has been loaded in SSC_THR and is waiting to be loaded in the Transmit Shift Register (TSR).

1 = SSC_THR is empty.

- **TXEMPTY: Transmit Empty**

0 = Data remains in SSC_THR or is currently transmitted from TSR.

1 = Last data written in SSC_THR has been loaded in TSR and last data loaded in TSR has been transmitted.

- **RXRDY: Receive Ready**

0 = SSC_RHR is empty.

1 = Data has been received and loaded in SSC_RHR.

- **OVRUN: Receive Overrun**

0 = No data has been loaded in SSC_RHR while previous data has not been read since the last read of the Status Register.

1 = Data has been loaded in SSC_RHR while previous data has not yet been read since the last read of the Status Register.

- **CP0: Compare 0**

0 = A compare 0 has not occurred since the last read of the Status Register.

1 = A compare 0 has occurred since the last read of the Status Register.

- **CP1: Compare 1**

0 = A compare 1 has not occurred since the last read of the Status Register.

1 = A compare 1 has occurred since the last read of the Status Register.

- **TXSYN: Transmit Sync**

0 = A Tx Sync has not occurred since the last read of the Status Register.

1 = A Tx Sync has occurred since the last read of the Status Register.

- **RXSYN: Receive Sync**

0 = An Rx Sync has not occurred since the last read of the Status Register.

1 = An Rx Sync has occurred since the last read of the Status Register.

- **TXEN: Transmit Enable**

0 = Transmit is disabled.

1 = Transmit is enabled.

- **RXEN: Receive Enable**

0 = Receive is disabled.

1 = Receive is enabled.

30.9.14 SSC Interrupt Enable Register

Name: SSC_IER

Address: 0x40004044

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
–	–	–	–	RXSYN	TXSYN	CP1	CP0
7	6	5	4	3	2	1	0
–	–	OVRUN	RXRDY	–	–	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready Interrupt Enable**

0 = 0 = No effect.

1 = Enables the Transmit Ready Interrupt.

- **TXEMPTY: Transmit Empty Interrupt Enable**

0 = No effect.

1 = Enables the Transmit Empty Interrupt.

- **RXRDY: Receive Ready Interrupt Enable**

0 = No effect.

1 = Enables the Receive Ready Interrupt.

- **OVRUN: Receive Overrun Interrupt Enable**

0 = No effect.

1 = Enables the Receive Overrun Interrupt.

- **CP0: Compare 0 Interrupt Enable**

0 = No effect.

1 = Enables the Compare 0 Interrupt.

- **CP1: Compare 1 Interrupt Enable**

0 = No effect.

1 = Enables the Compare 1 Interrupt.

- **TXSYN: Tx Sync Interrupt Enable**

0 = No effect.

1 = Enables the Tx Sync Interrupt.

- **RXSYN: Rx Sync Interrupt Enable**

0 = No effect.

1 = Enables the Rx Sync Interrupt.

30.9.15 SSC Interrupt Disable Register

Name: SSC_IDR

Address: 0x40004048

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
–	–	–	–	RXSYN	TXSYN	CP1	CP0
7	6	5	4	3	2	1	0
–	–	OVRUN	RXRDY	–	–	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready Interrupt Disable**

0 = No effect.

1 = Disables the Transmit Ready Interrupt.

- **TXEMPTY: Transmit Empty Interrupt Disable**

0 = No effect.

1 = Disables the Transmit Empty Interrupt.

- **RXRDY: Receive Ready Interrupt Disable**

0 = No effect.

1 = Disables the Receive Ready Interrupt.

- **OVRUN: Receive Overrun Interrupt Disable**

0 = No effect.

1 = Disables the Receive Overrun Interrupt.

- **CP0: Compare 0 Interrupt Disable**

0 = No effect.

1 = Disables the Compare 0 Interrupt.

- **CP1: Compare 1 Interrupt Disable**

0 = No effect.

1 = Disables the Compare 1 Interrupt.

- **TXSYN: Tx Sync Interrupt Enable**

0 = No effect.

1 = Disables the Tx Sync Interrupt.

- **RXSYN: Rx Sync Interrupt Enable**

0 = No effect.

1 = Disables the Rx Sync Interrupt.

30.9.16 SSC Interrupt Mask Register

Name: SSC_IMR

Address: 0x4000404C

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
–	–	–	–	RXSYN	TXSYN	CP1	CP0
7	6	5	4	3	2	1	0
–	–	OVRUN	RXRDY	–	–	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready Interrupt Mask**

0 = The Transmit Ready Interrupt is disabled.

1 = The Transmit Ready Interrupt is enabled.

- **TXEMPTY: Transmit Empty Interrupt Mask**

0 = The Transmit Empty Interrupt is disabled.

1 = The Transmit Empty Interrupt is enabled.

- **RXRDY: Receive Ready Interrupt Mask**

0 = The Receive Ready Interrupt is disabled.

1 = The Receive Ready Interrupt is enabled.

- **OVRUN: Receive Overrun Interrupt Mask**

0 = The Receive Overrun Interrupt is disabled.

1 = The Receive Overrun Interrupt is enabled.

- **CP0: Compare 0 Interrupt Mask**

0 = The Compare 0 Interrupt is disabled.

1 = The Compare 0 Interrupt is enabled.

- **CP1: Compare 1 Interrupt Mask**

0 = The Compare 1 Interrupt is disabled.

1 = The Compare 1 Interrupt is enabled.

- **TXSYN: Tx Sync Interrupt Mask**

0 = The Tx Sync Interrupt is disabled.

1 = The Tx Sync Interrupt is enabled.

- **RXSYN: Rx Sync Interrupt Mask**

0 = The Rx Sync Interrupt is disabled.

1 = The Rx Sync Interrupt is enabled.

30.9.17 SSC Write Protect Mode Register

Name: SSC_WPMR

Address: 0x400040E4

Access: Read-write

Reset: See Table 30-6

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 0x535343 (“SSC” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x535343 (“SSC” in ASCII).

Protects the registers:

- “SSC Clock Mode Register” on page 593
- “SSC Receive Clock Mode Register” on page 594
- “SSC Receive Frame Mode Register” on page 596
- “SSC Transmit Clock Mode Register” on page 598
- “SSC Transmit Frame Mode Register” on page 600
- “SSC Receive Compare 0 Register” on page 606
- “SSC Receive Compare 1 Register” on page 607

- **WPKEY: Write Protect KEY**

Should be written at value 0x535343 (“SSC” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

30.9.18 SSC Write Protect Status Register

Name: SSC_WPSR

Address: 0x400040E8

Access: Read-only

Reset: See Table 30-6

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 SSC_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the SSC_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 SSC_WPSR automatically clears all fields.

31. Parallel Input/Output Controller (PIO)

31.1 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 system 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 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.

31.2 Embedded Characteristics

- Up to 4 PIO Controllers, PIOA, PIOB, PIOC, PIOD, PIOE and PIOF controlling a maximum of 103 I/O Lines
- Each PIO Controller controls up to 32 programmable I/O Lines

Table 31-1. PIO Lines per PIO according to Version

Version	100 pin SAM3X/A	144 pin SAM3X	217 pin SAM3X8H[1]
PIOA	30		32
PIOB	32		
PIOC	1	31	
PIOD	-	10	31
PIOE	-	-	32
PIOF	-	-	6
Total	63	103	164

Note: 1.This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

- 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, rising edge, falling edge, low level and level interrupt
  - Debouncing and Glitch 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
- Synchronous output, provides Set and Clear of several I/O lines in a single write

31.3 Block Diagram

Figure 31-1. Block Diagram

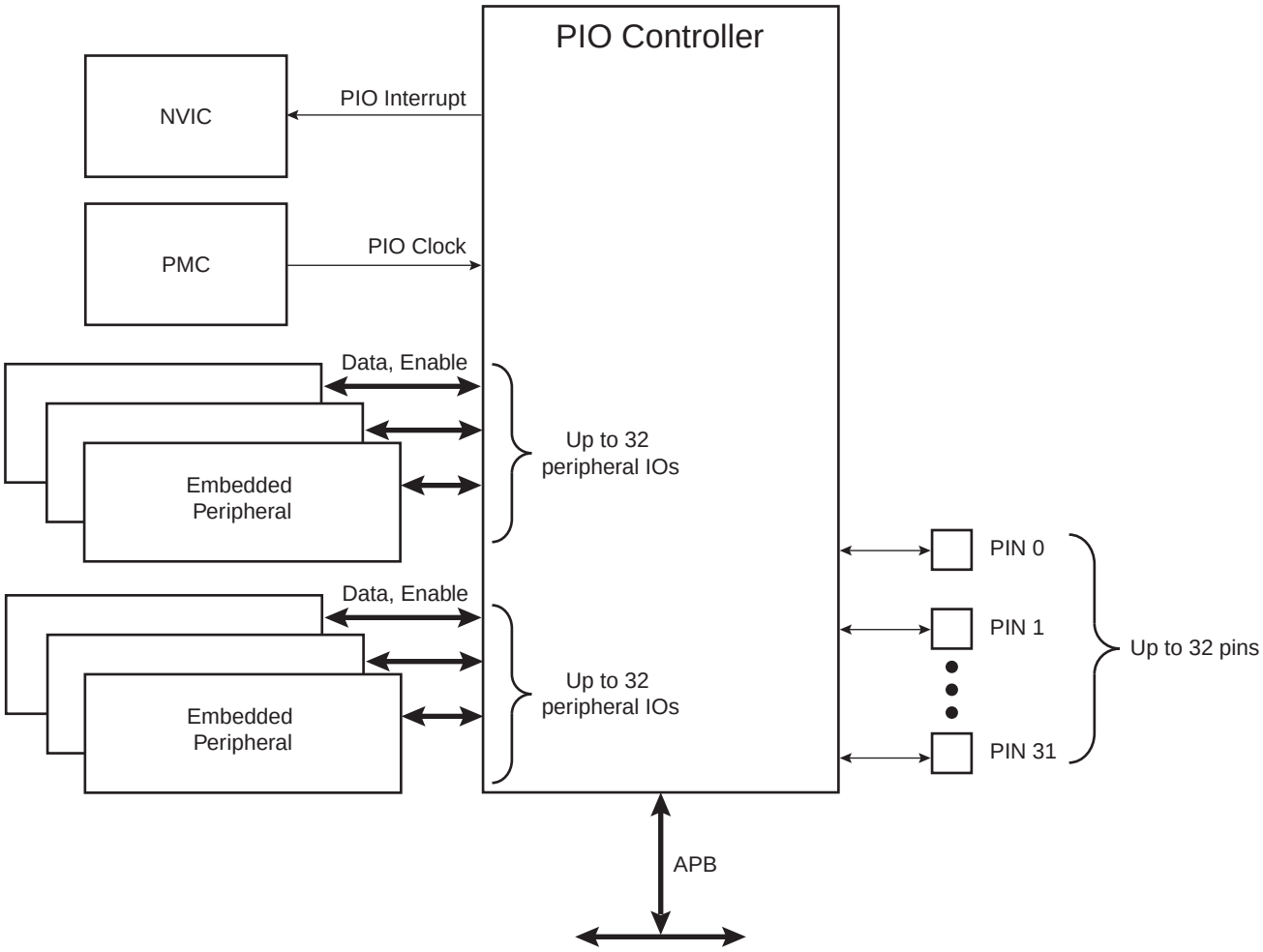
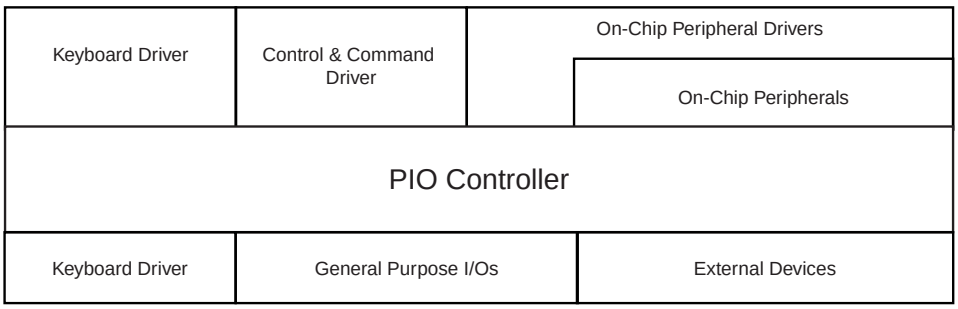


Figure 31-2. Application Block Diagram



31.4 Product Dependencies

31.4.1 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.

31.4.2 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.

31.4.3 Interrupt Generation

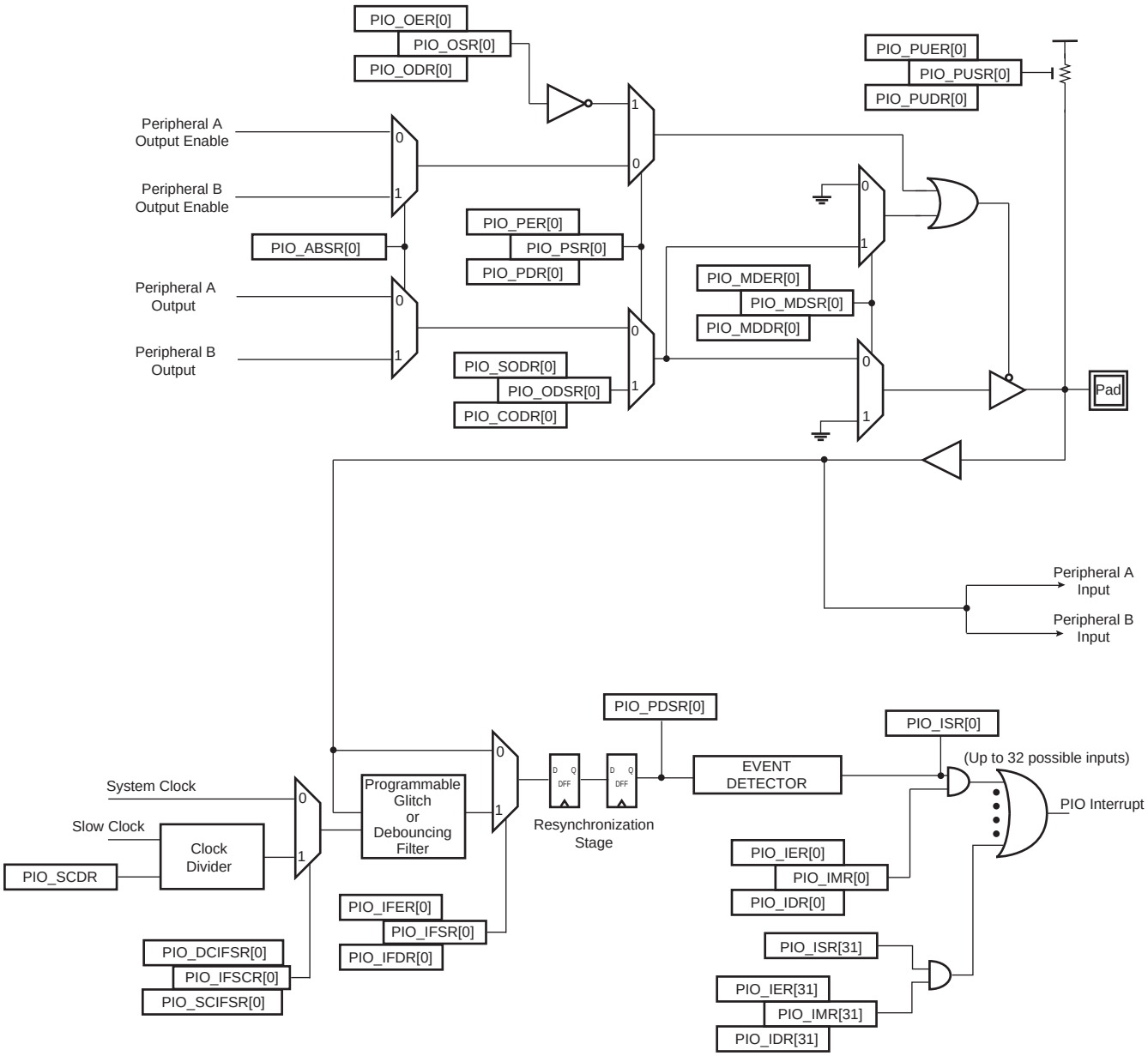
The PIO COntroller is connected on one of the sources of the Nested Vectored Interrupt Controller (NVIC). Using the PIO Controller requires the NVIC to be programmed first.

The PIO Controller interrupt can be generated only if the PIO Controller clock is enabled.

31.5 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 31-3. In this description each signal shown represents but one of up to 32 possible indexes.

Figure 31-3. I/O Line Control Logic



31.5.1 Pull-up Resistor Control

Each I/O line is designed with an embedded pull-up 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 Resistor). 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.

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.

31.5.2 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_ABSR (AB Select Register). 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.

31.5.3 Peripheral A or B Selection

The PIO Controller provides multiplexing of up to two peripheral functions on a single pin. The selection is performed by writing PIO_ABSR (AB Select Register). For each pin, the corresponding bit at level 0 means peripheral A is selected whereas the corresponding bit at level 1 indicates that peripheral B is selected.

Note that multiplexing of peripheral lines A and B only affects the output line. The peripheral input lines are always connected to the pin input.

After reset, PIO_ABSR is 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.

Writing in PIO_ABSR 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 register (PIO_ABSR) in addition to a write in PIO_PDR.

31.5.4 Output Control

When the I/0 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 depending on the value in PIO_ABSR (AB Select Register) 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.

31.5.5 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.

31.5.6 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.

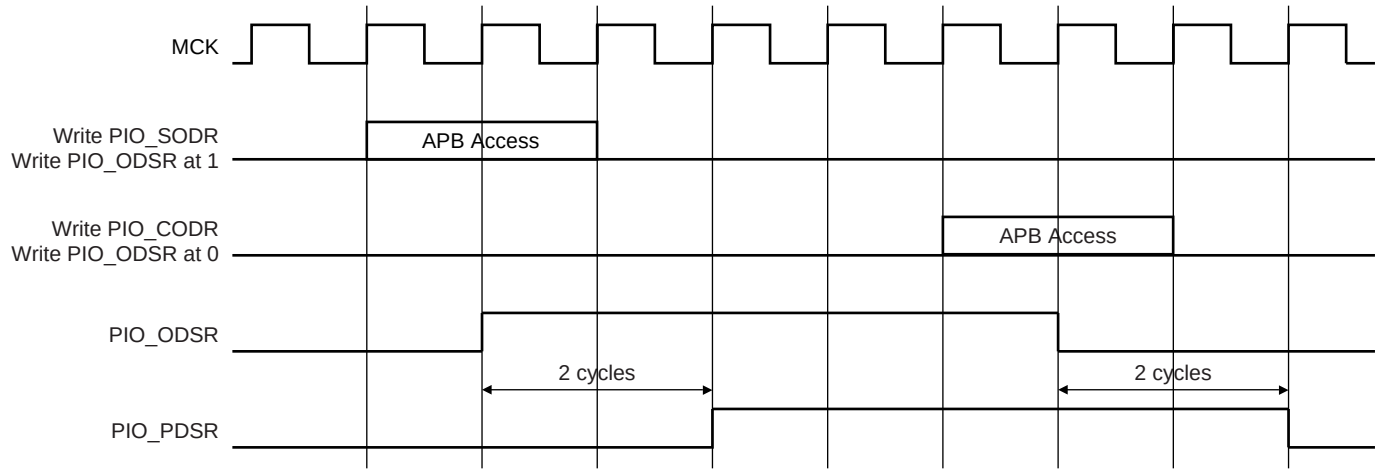
The Multi Drive feature is controlled by PIO_MDER (Multi-driver Enable Register) and PIO_MDDR (Multi-driver Disable Register). The Multi Drive can be selected whether the I/O line is controlled by the PIO controller or assigned to a peripheral function. PIO_MDSR (Multi-driver Status Register) indicates 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.

31.5.7 Output Line Timings

Figure 31-4 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 31-4 also shows when the feedback in PIO_PDSR is available.

Figure 31-4. Output Line Timings



31.5.8 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.

31.5.9 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_SCIFSR (System Clock Glitch Input Filter Select Register) and PIO_DIFSR (Debouncing Input Filter Select Register). Writing PIO_SCIFSR and PIO_DIFSR respectively, sets and clears bits in PIO_IFDGSR.

The current selection status can be checked by reading the register PIO_IFDGSR (Glitch or Debouncing Input Filter Selection Status Register).

- If PIO_IFDGSR[i] = 0: The glitch filter can filter a glitch with a duration of less than 1/2 Period of Master Clock.
- If PIO_IFDGSR[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)

Tdiv_slclk = ((DIV+1)*2).Tslow_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_SCIFSR and PIO_DIFSR 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 31-5 and Figure 31-6.

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 31-5. Input Glitch Filter Timing

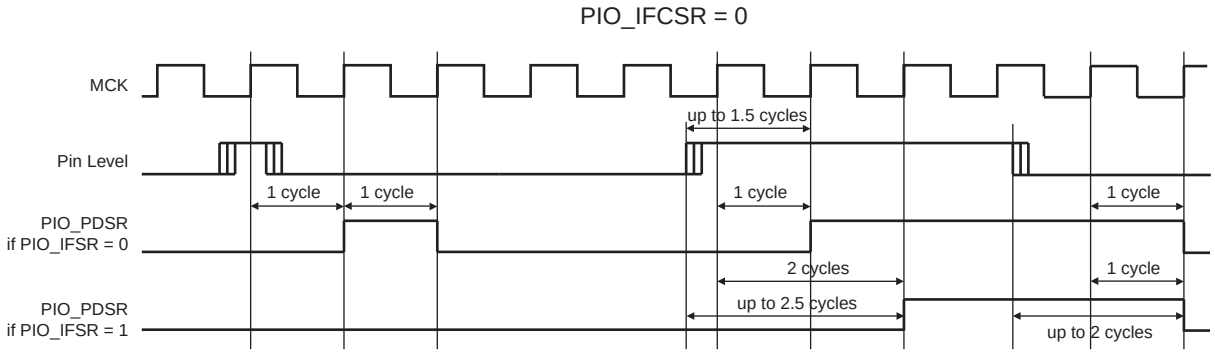
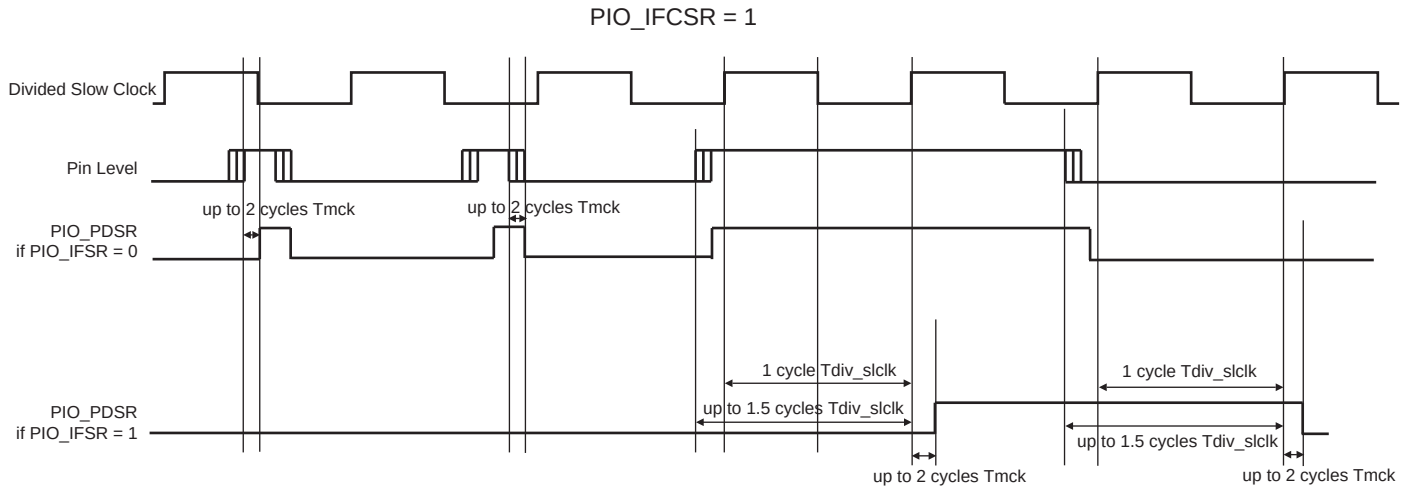


Figure 31-6. Input Debouncing Filter Timing



31.5.10 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_AIMDR (Additional Interrupt Modes Disable Register). The current state of this selection can be read through the PIO_AIMMR (Additional Interrupt Modes Mask Register)

These Additional Modes are:

- Rising Edge Detection
- Falling Edge Detection
- Low Level Detection
- High Level Detection

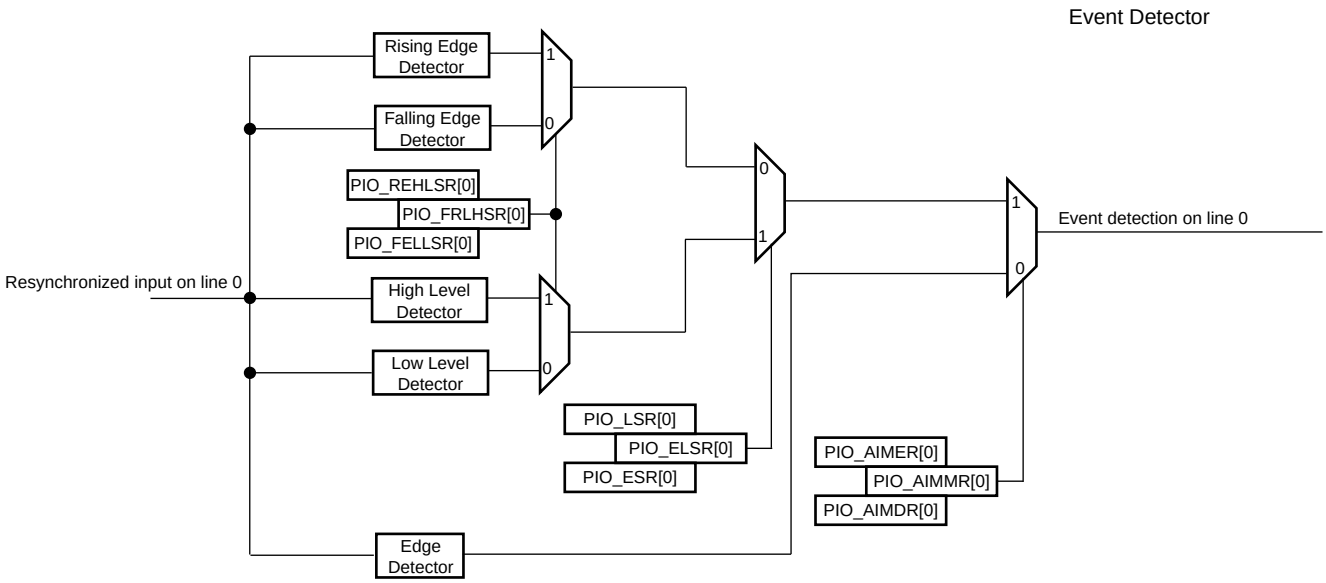
In order to select an Additional Interrupt Mode:

- 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 Nested Vector Interrupt Controller (NVIC).

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 31-7. Event Detector on Input Lines (Figure represents line 0)



31.5.10.1 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.

31.5.10.2 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.

31.5.10.3 Edge or Level Detection Configuration

Lines 3, 4 and 5 are configured in Level detection by writing 32'h0000_0038 in PIO_LSR.

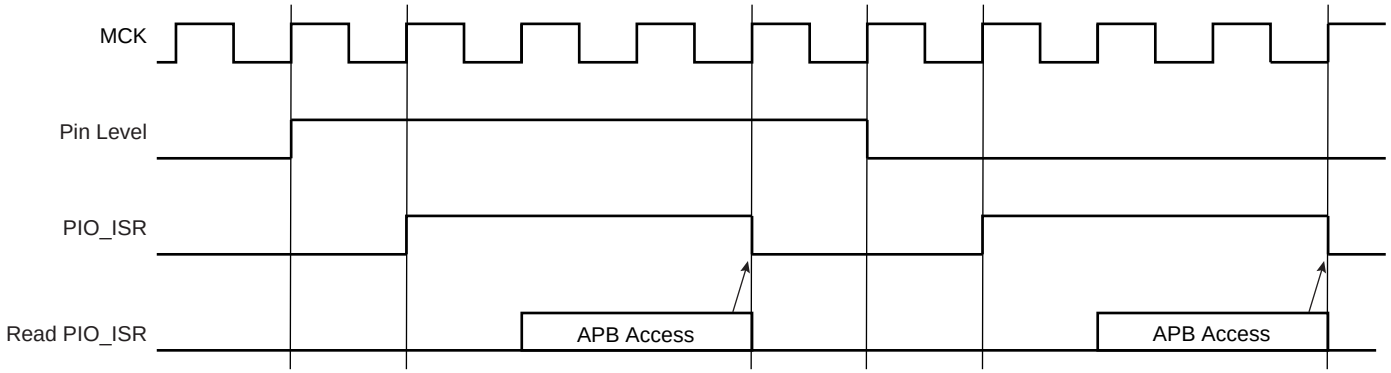
The other lines are configured in Edge detection by default, if they have not been previously configured. Otherwise, lines 0, 1, 2, 6 and 7 must be configured in Edge detection by writing 32'h0000_00C7 in PIO_ESR.

31.5.10.4 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 31-8. Input Change Interrupt Timings if there are no Additional Interrupt Modes



31.5.11 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 and PIO_ABSR 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 an hardware reset to the PIO Controller.

31.6 I/O Lines Programming Example

The programing example as shown in Table 31-2 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
- 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, no pull-up resistor
- I/O line 24 to 27 assigned to peripheral A with Input Change Interrupt and pull-up resistor

Table 31-2. Programming Example

Register	Value to be Written
PIO_PER	0x0000 FFFF
PIO_PDR	0xFFFF 0000
PIO_OER	0x0000 00FF
PIO_ODR	0xFFFF FF00
PIO_IFER	0x0000 0F00
PIO_IFDR	0xFFFF F0FF
PIO_SODR	0x0000 0000
PIO_CODR	0x0FFF FFFF
PIO_IER	0x0F00 0F00
PIO_IDR	0xF0FF F0FF
PIO_MDER	0x0000 000F
PIO_MDDR	0xFFFF FFF0
PIO_PUDR	0xF0F0 00F0
PIO_PUER	0x0F0F FF0F
PIO_ABSR	0x00F0 0000
PIO_OWER	0x0000 000F
PIO_OWDR	0x0FFF FFF0

31.6.1 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 WPVSRC 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 Controller PIO Enable Register" on page 633
- "PIO Controller PIO Disable Register" on page 634
- "PIO Controller Output Enable Register" on page 636
- "PIO Controller Output Disable Register" on page 637
- "PIO Controller Input Filter Enable Register" on page 639
- "PIO Controller Input Filter Disable Register" on page 640
- "PIO Multi-driver Enable Register" on page 650
- "PIO Multi-driver Disable Register" on page 651
- "PIO Pull Up Disable Register" on page 653
- "PIO Pull Up Enable Register" on page 654
- "PIO Peripheral AB Select Register" on page 656
- "PIO Output Write Enable Register" on page 661
- "PIO Output Write Disable Register" on page 662

31.7 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 31-3. 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(4)	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	0x00000000
0x006C	Reserved			

Table 31-3. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0070	Peripheral AB Select Register[5]	PIO_ABSR	Read-Write	0x00000000
0x0074 to 0x007C	Reserved			
0x0080	System Clock Glitch Input Filter Select Register	PIO_SCIFSR	Write-Only	–
0x0084	Debouncing Input Filter Select Register	PIO_DIFSR	Write-Only	–
0x0088	Glitch or Debouncing Input Filter Clock Selection Status Register	PIO_IFDGSR	Read-Only	0x00000000
0x008C	Slow Clock Divider Debouncing Register	PIO_SCDR	Read-Write	0x00000000
0x0090 to 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 to 0x0144	Reserved			

Notes:
1. Reset value of PIO_PSR 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.
5. Only this set of registers clears the status by writing 1 in the first register and sets the status by writing 1 in the second register.

31.7.1 PIO Controller PIO Enable Register

Name: PIO_PER

Address: 0x400E0E00 (PIOA), 0x400E1000 (PIOB), 0x400E1200 (PIOC), 0x400E1400 (PIOD), 0x400E1600 (PIOE), 0x400E1800 (PIOF)

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).

31.7.2 PIO Controller PIO Disable Register

Name: PIO_PDR

Address: 0x400E0E04 (PIOA), 0x400E1004 (PIOB), 0x400E1204 (PIOC), 0x400E1404 (PIOD), 0x400E1604 (PIOE), 0x400E1804 (PIOF)

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).

31.7.3 PIO Controller PIO Status Register

Name: PIO_PSR

Address: 0x400E0E08 (PIOA), 0x400E1008 (PIOB), 0x400E1208 (PIOC), 0x400E1408 (PIOD), 0x400E1608 (PIOE), 0x400E1808 (PIOF)

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).

31.7.4 PIO Controller Output Enable Register

Name: PIO_OER

Address: 0x400E0E10 (PIOA), 0x400E1010 (PIOB), 0x400E1210 (PIOC), 0x400E1410 (PIOD), 0x400E1610 (PIOE), 0x400E1810 (PIOF)

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.

31.7.5 PIO Controller Output Disable Register

Name: PIO_ODR

Address: 0x400E0E14 (PIOA), 0x400E1014 (PIOB), 0x400E1214 (PIOC), 0x400E1414 (PIOD), 0x400E1614 (PIOE), 0x400E1814 (PIOF)

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.

31.7.6 PIO Controller Output Status Register

Name: PIO_OSR

Address: 0x400E0E18 (PIOA), 0x400E1018 (PIOB), 0x400E1218 (PIOC), 0x400E1418 (PIOD), 0x400E1618 (PIOE), 0x400E1818 (PIOF)

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.

31.7.7 PIO Controller Input Filter Enable Register

Name: PIO_IFER

Address: 0x400E0E20 (PIOA), 0x400E1020 (PIOB), 0x400E1220 (PIOC), 0x400E1420 (PIOD), 0x400E1620 (PIOE), 0x400E1820 (PIOF)

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.

31.7.8 PIO Controller Input Filter Disable Register

Name: PIO_IFDR

Address: 0x400E0E24 (PIOA), 0x400E1024 (PIOB), 0x400E1224 (PIOC), 0x400E1424 (PIOD), 0x400E1624 (PIOE), 0x400E1824 (PIOF)

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.

31.7.9 PIO Controller Input Filter Status Register

Name: PIO_IFSR

Address: 0x400E0E28 (PIOA), 0x400E1028 (PIOB), 0x400E1228 (PIOC), 0x400E1428 (PIOD), 0x400E1628 (PIOE), 0x400E1828 (PIOF)

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.

31.7.10 PIO Controller Set Output Data Register

Name: PIO_SODR

Address: 0x400E0E30 (PIOA), 0x400E1030 (PIOB), 0x400E1230 (PIOC), 0x400E1430 (PIOD), 0x400E1630 (PIOE), 0x400E1830 (PIOF)

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.

31.7.11 PIO Controller Clear Output Data Register

Name: PIO_CODR

Address: 0x400E0E34 (PIOA), 0x400E1034 (PIOB), 0x400E1234 (PIOC), 0x400E1434 (PIOD), 0x400E1634 (PIOE), 0x400E1834 (PIOF)

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.

31.7.12 PIO Controller Output Data Status Register

Name: PIO_ODSR

Address: 0x400E0E38 (PIOA), 0x400E1038 (PIOB), 0x400E1238 (PIOC), 0x400E1438 (PIOD), 0x400E1638 (PIOE), 0x400E1838 (PIOF)

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.

31.7.13 PIO Controller Pin Data Status Register

Name: PIO_PDSR

Address: 0x400E0E3C (PIOA), 0x400E103C (PIOB), 0x400E123C (PIOC), 0x400E143C (PIOD), 0x400E163C (PIOE), 0x400E183C (PIOF)

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.

31.7.14 PIO Controller Interrupt Enable Register

Name: PIO_IER

Address: 0x400E0E40 (PIOA), 0x400E1040 (PIOB), 0x400E1240 (PIOC), 0x400E1440 (PIOD), 0x400E1640 (PIOE), 0x400E1840 (PIOF)

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.

31.7.15 PIO Controller Interrupt Disable Register

Name: PIO_IDR

Address: 0x400E0E44 (PIOA), 0x400E1044 (PIOB), 0x400E1244 (PIOC), 0x400E1444 (PIOD), 0x400E1644 (PIOE), 0x400E1844 (PIOF)

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.

31.7.16 PIO Controller Interrupt Mask Register

Name: PIO_IMR

Address: 0x400E0E48 (PIOA), 0x400E1048 (PIOB), 0x400E1248 (PIOC), 0x400E1448 (PIOD), 0x400E1648 (PIOE), 0x400E1848 (PIOF)

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.

31.7.17 PIO Controller Interrupt Status Register

Name: PIO_ISR

Address: 0x400E0E4C (PIOA), 0x400E104C (PIOB), 0x400E124C (PIOC), 0x400E144C (PIOD), 0x400E164C (PIOE), 0x400E184C (PIOF)

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.

31.7.18 PIO Multi-driver Enable Register

Name: PIO_MDER

Address: 0x400E0E50 (PIOA), 0x400E1050 (PIOB), 0x400E1250 (PIOC), 0x400E1450 (PIOD), 0x400E1650 (PIOE), 0x400E1850 (PIOF)

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.

31.7.19 PIO Multi-driver Disable Register

Name: PIO_MDDR

Address: 0x400E0E54 (PIOA), 0x400E1054 (PIOB), 0x400E1254 (PIOC), 0x400E1454 (PIOD), 0x400E1654 (PIOE), 0x400E1854 (PIOF)

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.

31.7.20 PIO Multi-driver Status Register

Name: PIO_MDSR

Address: 0x400E0E58 (PIOA), 0x400E1058 (PIOB), 0x400E1258 (PIOC), 0x400E1458 (PIOD), 0x400E1658 (PIOE), 0x400E1858 (PIOF)

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.

31.7.21 PIO Pull Up Disable Register

Name: PIO_PUDR

Address: 0x400E0E60 (PIOA), 0x400E1060 (PIOB), 0x400E1260 (PIOC), 0x400E1460 (PIOD), 0x400E1660 (PIOE), 0x400E1860 (PIOF)

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.

31.7.22 PIO Pull Up Enable Register

Name: PIO_PUER

Address: 0x400E0E64 (PIOA), 0x400E1064 (PIOB), 0x400E1264 (PIOC), 0x400E1464 (PIOD), 0x400E1664 (PIOE), 0x400E1864 (PIOF)

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.

31.7.23 PIO Pull Up Status Register

Name: PIO_PUSR

Address: 0x400E0E68 (PIOA), 0x400E1068 (PIOB), 0x400E1268 (PIOC), 0x400E1468 (PIOD), 0x400E1668 (PIOE), 0x400E1868 (PIOF)

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.

31.7.24 PIO Peripheral AB Select Register

Name: PIO_ABSR

Address: 0x400E0E70 (PIOA), 0x400E1070 (PIOB), 0x400E1270 (PIOC), 0x400E1470 (PIOD), 0x400E1670 (PIOE), 0x400E1870 (PIOF)

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 A Select.**

0: Assigns the I/O line to the Peripheral A function.

1: Assigns the I/O line to the Peripheral B function.

31.7.25 PIO System Clock Glitch Input Filtering Select Register

Name: PIO_SCIFSR

Address: 0x400E0E80 (PIOA), 0x400E1080 (PIOB), 0x400E1280 (PIOC), 0x400E1480 (PIOD), 0x400E1680 (PIOE), 0x400E1880 (PIOF)

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: System Clock Glitch Filtering Select.**

0: No Effect.

1: The Glitch Filter is able to filter glitches with a duration < Tmck/2.

31.7.26 PIO Debouncing Input Filtering Select Register

Name: PIO_DIFSR

Address: 0x400E0E84 (PIOA), 0x400E1084 (PIOB), 0x400E1284 (PIOC), 0x400E1484 (PIOD), 0x400E1684 (PIOE), 0x400E1884 (PIOF)

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.

31.7.27 PIO Glitch or Debouncing Input Filter Selection Status Register

Name: PIO_IFDGSR

Address: 0x400E0E88 (PIOA), 0x400E1088 (PIOB), 0x400E1288 (PIOC), 0x400E1488 (PIOD), 0x400E1688 (PIOE), 0x400E1888 (PIOF)

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.

31.7.28 PIO Slow Clock Divider Debouncing Register

Name: PIO_SCDR

Address: 0x400E0E8C (PIOA), 0x400E108C (PIOB), 0x400E128C (PIOC), 0x400E148C (PIOD), 0x400E168C (PIOE), 0x400E188C (PIOF)

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
-	-	DIV13	DIV12	DIV11	DIV10	DIV9	DIV8
7	6	5	4	3	2	1	0
DIV7	DIV6	DIV5	DIV4	DIV3	DIV2	DIV1	DIV0

- **DIV: Slow Clock Divider Selection for Debouncing**

Tdiv_slclk = 2*(DIV+1)*Tslow_clock.

31.7.29 PIO Output Write Enable Register

Name: PIO_OWER

Address: 0x400E0EA0 (PIOA), 0x400E10A0 (PIOB), 0x400E12A0 (PIOC), 0x400E14A0 (PIOD), 0x400E16A0 (PIOE), 0x400E18A0 (PIOF)

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.

31.7.30 PIO Output Write Disable Register

Name: PIO_OWDR

Address: 0x400E0EA4 (PIOA), 0x400E10A4 (PIOB), 0x400E12A4 (PIOC), 0x400E14A4 (PIOD), 0x400E16A4 (PIOE), 0x400E18A4 (PIOF)

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.

31.7.31 PIO Output Write Status Register

Name: PIO_OWSR

Address: 0x400E0EA8 (PIOA), 0x400E10A8 (PIOB), 0x400E12A8 (PIOC), 0x400E14A8 (PIOD), 0x400E16A8 (PIOE), 0x400E18A8 (PIOF)

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.

31.7.32 Additional Interrupt Modes Enable Register

Name: PIO_AIMER

Address: 0x400E0EB0 (PIOA), 0x400E10B0 (PIOB), 0x400E12B0 (PIOC), 0x400E14B0 (PIOD), 0x400E16B0 (PIOE), 0x400E18B0 (PIOF)

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.

31.7.33 Additional Interrupt Modes Disable Register

Name: PIO_AIMDR

Address: 0x400E0EB4 (PIOA), 0x400E10B4 (PIOB), 0x400E12B4 (PIOC), 0x400E14B4 (PIOD), 0x400E16B4 (PIOE), 0x400E18B4 (PIOF)

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).

31.7.34 Additional Interrupt Modes Mask Register

Name: PIO_AIMMR

Address: 0x400E0EB8 (PIOA), 0x400E10B8 (PIOB), 0x400E12B8 (PIOC), 0x400E14B8 (PIOD), 0x400E16B8 (PIOE), 0x400E18B8 (PIOF)

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

31.7.35 Edge Select Register

Name: PIO_ESR

Address: 0x400E0EC0 (PIOA), 0x400E10C0 (PIOB), 0x400E12C0 (PIOC), 0x400E14C0 (PIOD), 0x400E16C0 (PIOE), 0x400E18C0 (PIOF)

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.

31.7.36 Level Select Register

Name: PIO_LSR

Address: 0x400E0EC4 (PIOA), 0x400E10C4 (PIOB), 0x400E12C4 (PIOC), 0x400E14C4 (PIOD), 0x400E16C4 (PIOE), 0x400E18C4 (PIOF)

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.

31.7.37 Edge/Level Status Register

Name: PIO_ELSR

Address: 0x400E0EC8 (PIOA), 0x400E10C8 (PIOB), 0x400E12C8 (PIOC), 0x400E14C8 (PIOD), 0x400E16C8 (PIOE), 0x400E18C8 (PIOF)

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.

31.7.38 Falling Edge/Low Level Select Register

Name: PIO_FELLSR

Address: 0x400E0ED0 (PIOA), 0x400E10D0 (PIOB), 0x400E12D0 (PIOC), 0x400E14D0 (PIOD), 0x400E16D0 (PIOE), 0x400E18D0 (PIOF)

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.

31.7.39 Rising Edge/High Level Select Register

Name: PIO_REHLSR

Address: 0x400E0ED4 (PIOA), 0x400E10D4 (PIOB), 0x400E12D4 (PIOC), 0x400E14D4 (PIOD), 0x400E16D4 (PIOE), 0x400E18D4 (PIOF)

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.

31.7.40 Fall/Rise - Low/High Status Register

Name: PIO_FRLHSR

Address: 0x400E0ED8 (PIOA), 0x400E10D8 (PIOB), 0x400E12D8 (PIOC), 0x400E14D8 (PIOD), 0x400E16D8 (PIOE), 0x400E18D8 (PIOF)

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).

31.7.41 Lock Status Register

Name: PIO_LOCKSR

Address: 0x400E0EE0 (PIOA), 0x400E10E0 (PIOB), 0x400E12E0 (PIOC), 0x400E14E0 (PIOD), 0x400E16E0 (PIOE), 0x400E18E0 (PIOF)

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.

31.7.42 PIO Write Protect Mode Register

Name: PIO_WPMR

Address: 0x400E0EE4 (PIOA), 0x400E10E4 (PIOB), 0x400E12E4 (PIOC), 0x400E14E4 (PIOD), 0x400E16E4 (PIOE), 0x400E18E4 (PIOF)

Access: Read-write

Reset: See Table 31-3

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 31.6.1 ”Write Protection Registers”.

- **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 Controller PIO Enable Register” on page 633
- “PIO Controller PIO Disable Register” on page 634
- “PIO Controller Output Enable Register” on page 636
- “PIO Controller Output Disable Register” on page 637
- “PIO Controller Input Filter Enable Register” on page 639
- “PIO Controller Input Filter Disable Register” on page 640
- “PIO Multi-driver Enable Register” on page 650
- “PIO Multi-driver Disable Register” on page 651
- “PIO Pull Up Disable Register” on page 653
- “PIO Pull Up Enable Register” on page 654
- “PIO Peripheral AB Select Register” on page 656
- “PIO Output Write Enable Register” on page 661
- “PIO Output Write Disable Register” on page 662

- **WPKEY: Write Protect KEY**

Should be written at value 0x50494F (“PIO” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

31.7.43 PIO Write Protect Status Register

Name: PIO_WPSR

Address: 0x400E0EE8 (PIOA), 0x400E10E8 (PIOB), 0x400E12E8 (PIOC), 0x400E14E8 (PIOD), 0x400E16E8 (PIOE), 0x400E18E8 (PIOF)

Access: Read-only

Reset: See Table 31-3

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.

32. Serial Peripheral Interface (SPI)

32.1 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:

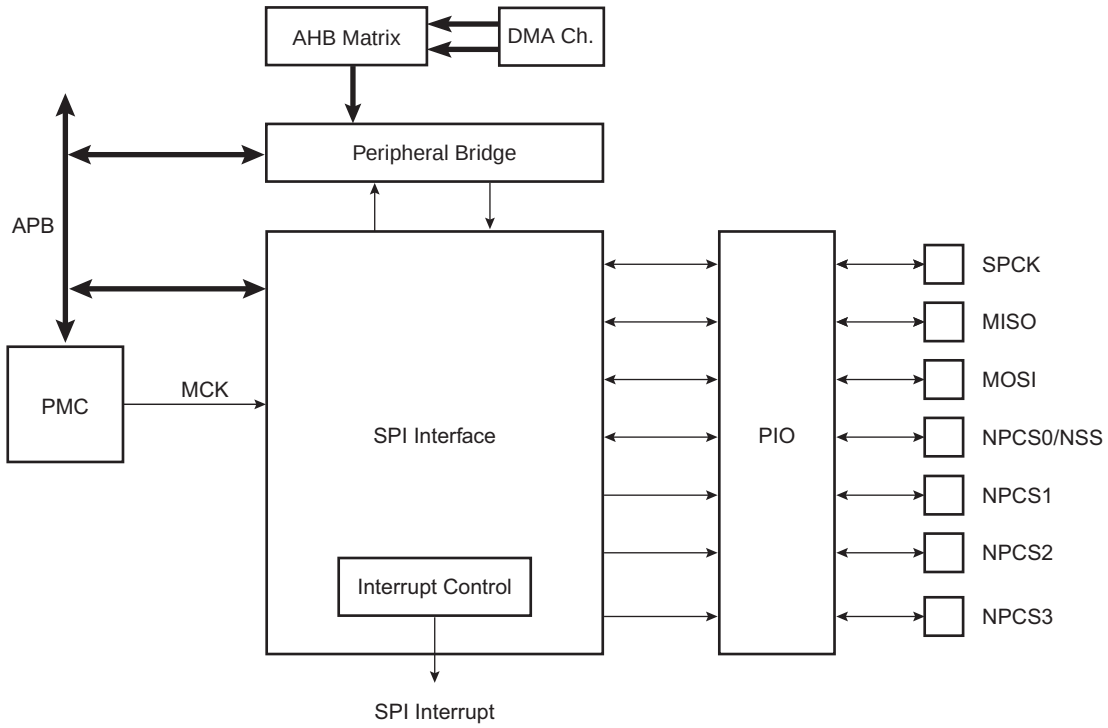
- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during 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.

32.2 Embedded Characteristics

- Supports Communication with Serial External Devices
  - 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 Co-processors
- Master or Slave Serial Peripheral Bus Interface
  - 8- 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 DMA Channel Capabilities Optimizes Data Transfers
  - One channel for the Receiver, One Channel for the Transmitter

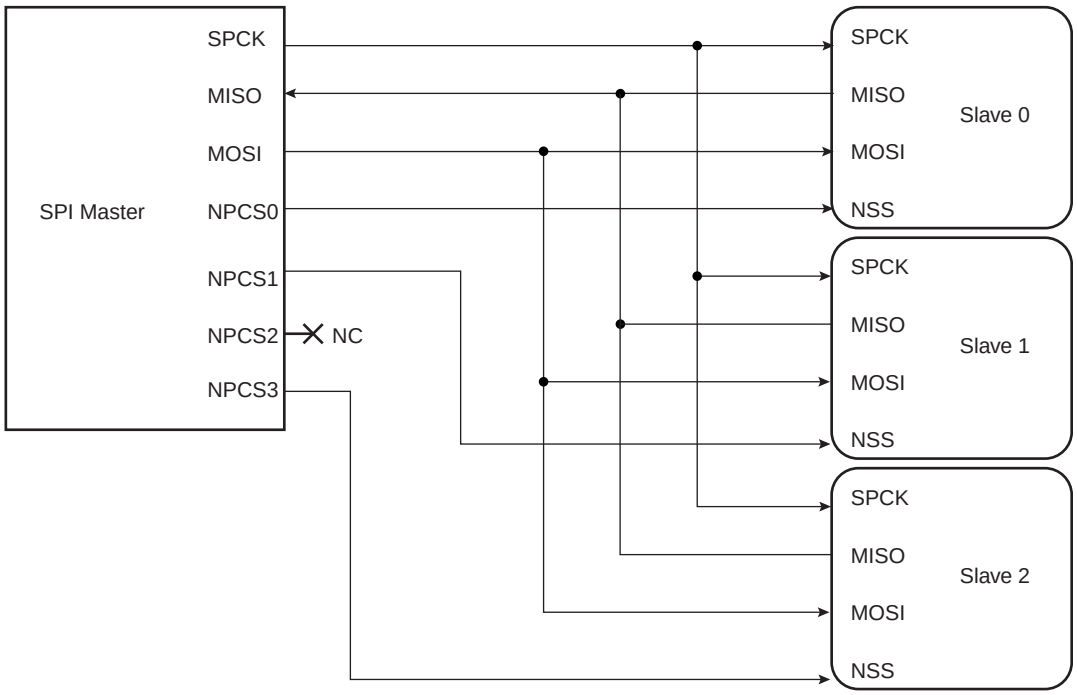
32.3 Block Diagram

Figure 32-1. Block Diagram



32.4 Application Block Diagram

Figure 32-2. Application Block Diagram: Single Master/Multiple Slave Implementation



32.5 Signal Description

Table 32-1. 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

32.6 Product Dependencies

32.6.1 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 32-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
SPI0	SPI0_MISO	PA25	A
SPI0	SPI0_MOSI	PA26	A
SPI0	SPI0_NPCS0	PA28	A
SPI0	SPI0_NPCS1	PA29	A
SPI0	SPI0_NPCS1	PB20	B
SPI0	SPI0_NPCS2	PA30	A
SPI0	SPI0_NPCS2	PB21	B
SPI0	SPI0_NPCS3	PA31	A
SPI0	SPI0_NPCS3	PB23	B
SPI0	SPI0_SPCK	PA27	A
SPI1	SPI1_MISO	PE28	A
SPI1	SPI1_MOSI	PE29	A
SPI1	SPI1_NPCS0	PE31	A
SPI1	SPI1_NPCS1	PF0	A
SPI1	SPI1_NPCS2	PF1	A
SPI1	SPI1_NPCS3	PF2	A
SPI1	SPI1_SPCK	PE30	A

32.6.2 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.

32.6.3 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 32-3. Peripheral IDs

Instance	ID
SPI0	24
SPI1	25

32.7 Functional Description

32.7.1 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.

32.7.2 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.

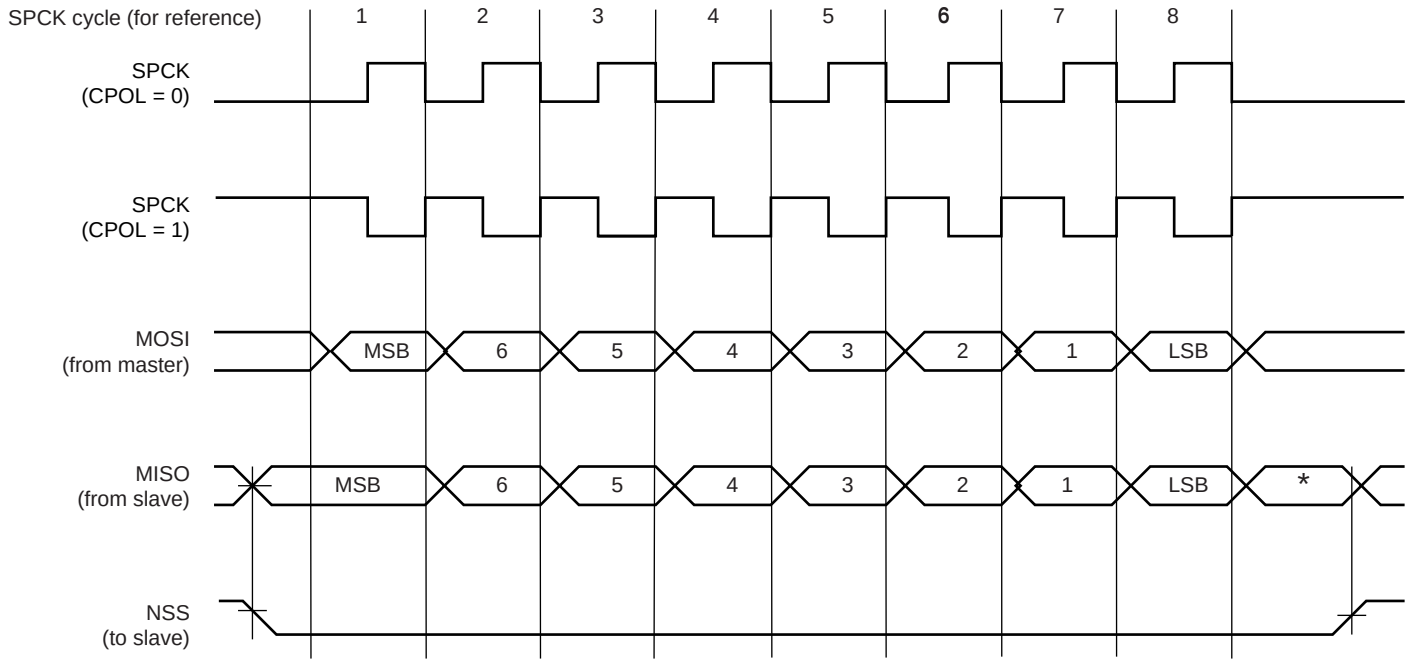
Table 32-4 shows the four modes and corresponding parameter settings.

Table 32-4. 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

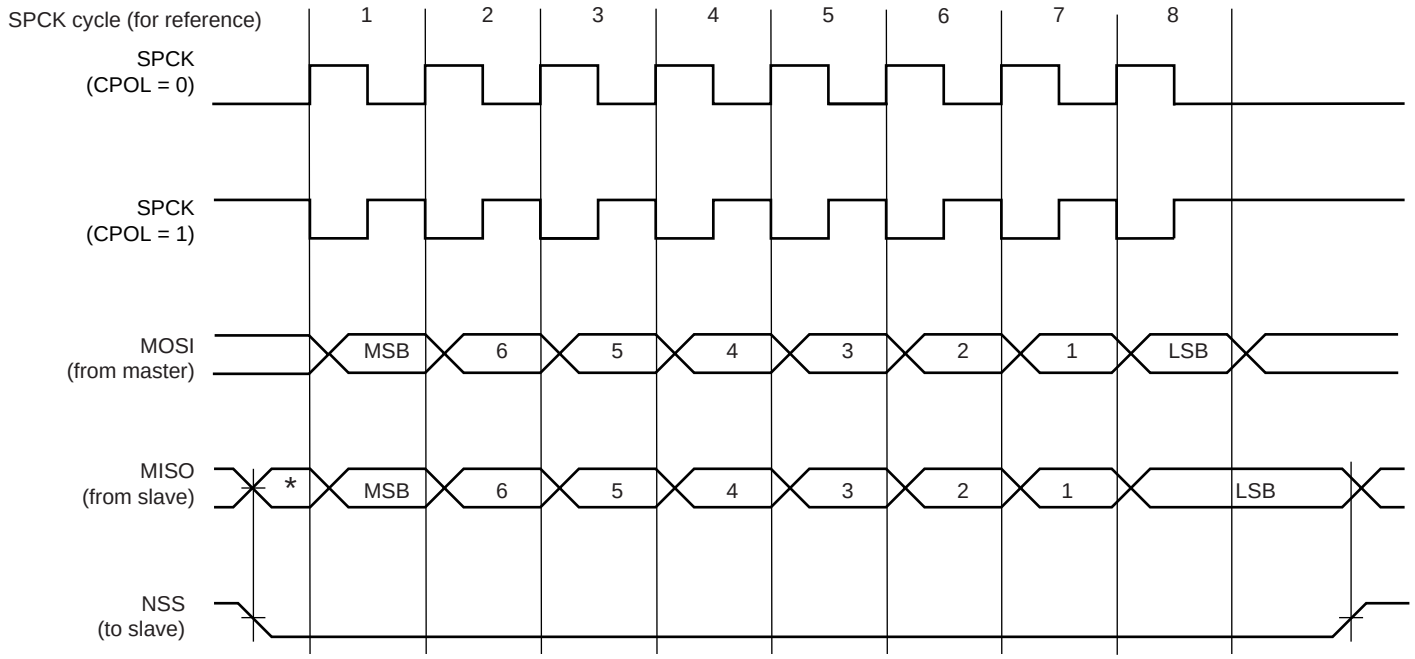
Figure 32-3 and Figure 32-4 show examples of data transfers.

Figure 32-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)



* Not defined, but normally MSB of previous character received.

Figure 32-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)



* Not defined but normally LSB of previous character transmitted.

32.7.3 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 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.

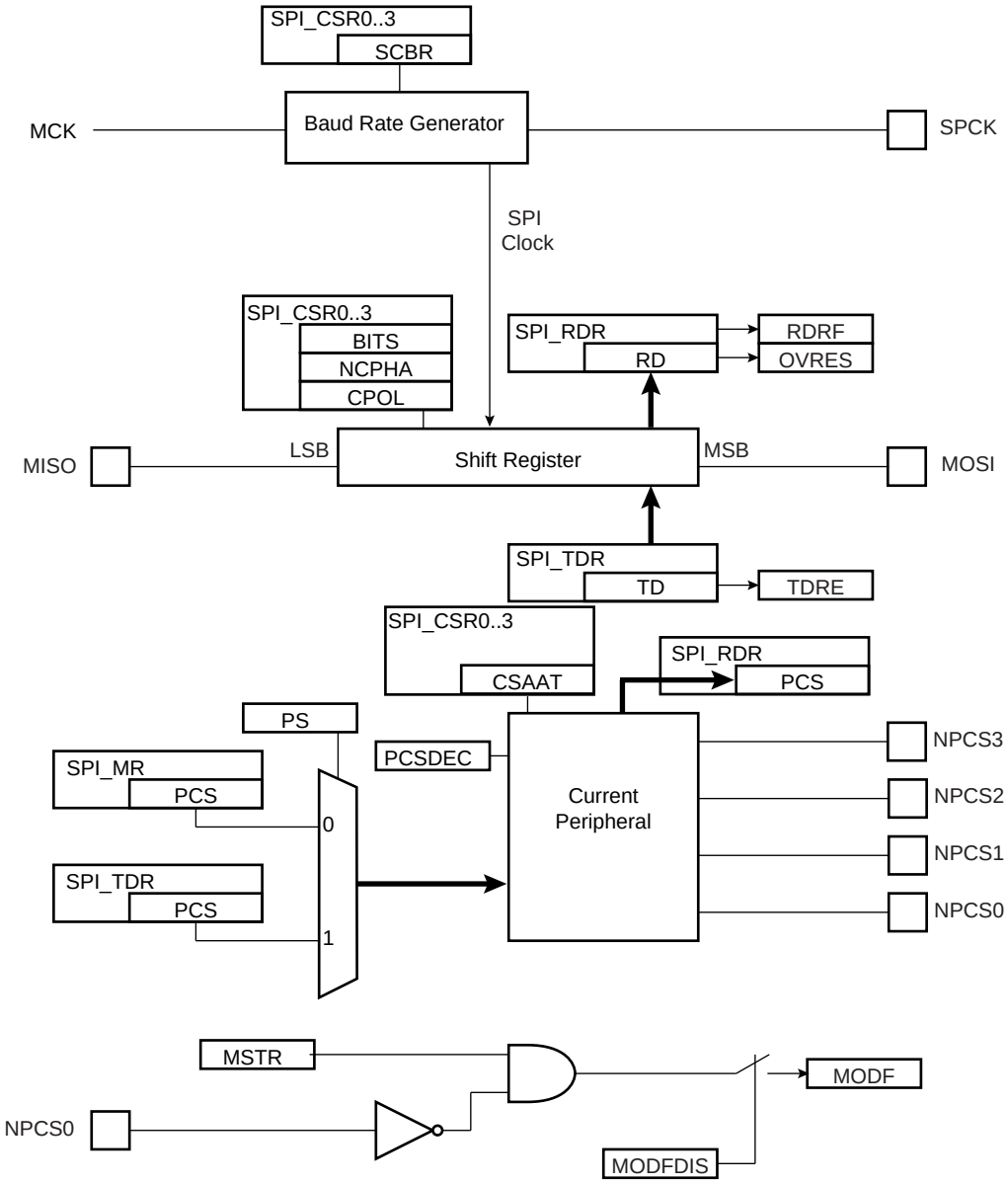
The transfer of received data from the Shift Register in SPI_RDR is indicated by the RDRF bit (Receive Data Register Full) in the Status Register (SPI_SR). When the received data is read, the RDRF bit is cleared.

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 32-5, shows a block diagram of the SPI when operating in Master Mode. Figure 32-6 on page 684 shows a flow chart describing how transfers are handled.

32.7.3.1 Master Mode Block Diagram

Figure 32-5. Master Mode Block Diagram



32.7.3.2 Master Mode Flow Diagram

Figure 32-6. Master Mode Flow Diagram

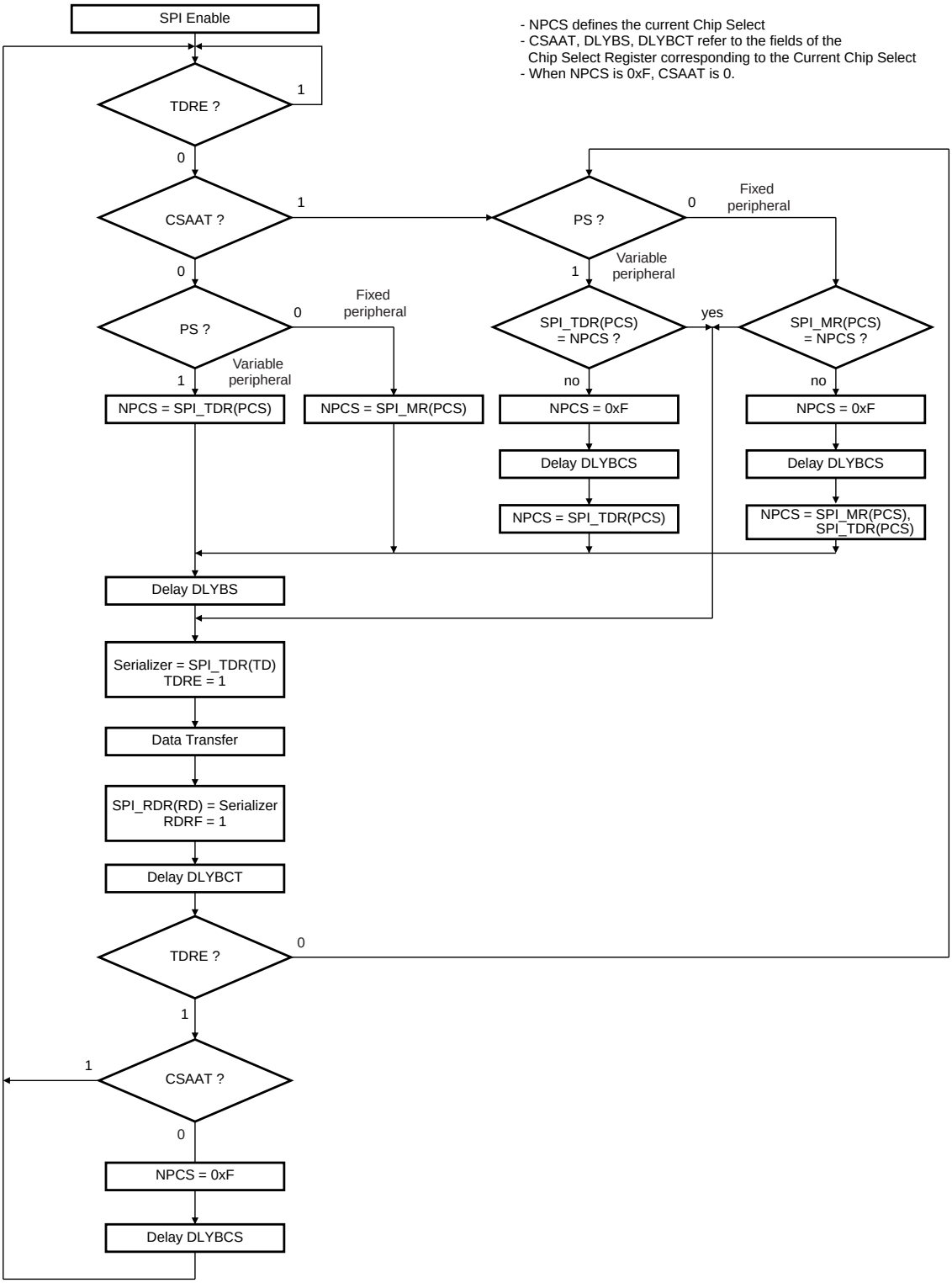
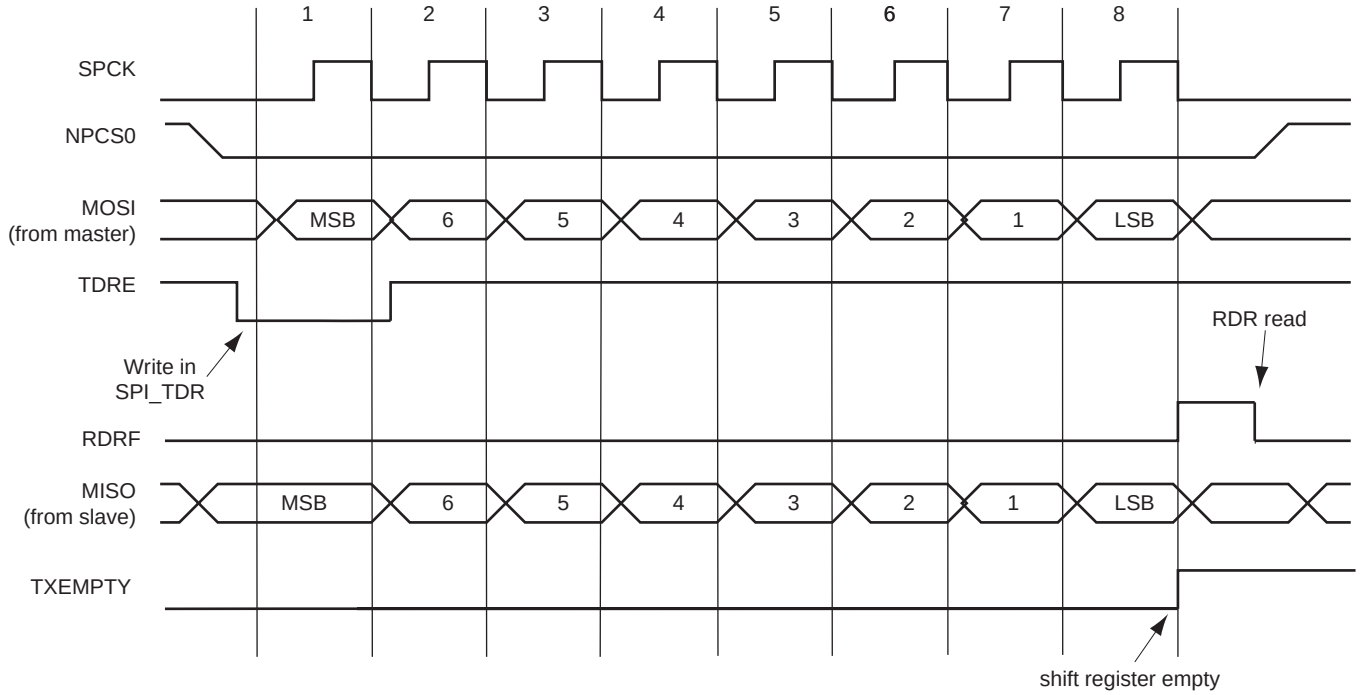


Figure 32-7 shows Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags behavior within the SPI_SR (Status Register) during an 8-bit data transfer in fixed mode and no Peripheral Data Controller involved.

Figure 32-7. Status Register Flags Behavior



32.7.3.3 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.

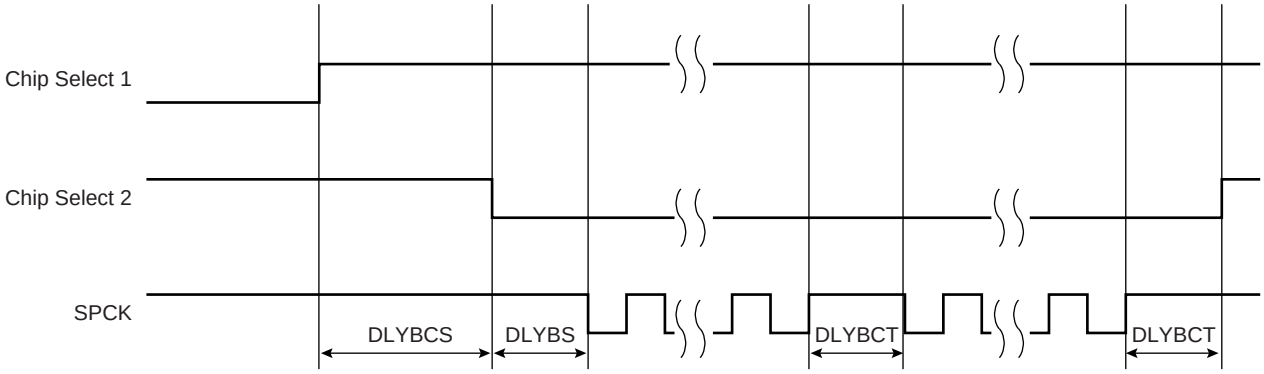
32.7.3.4 Transfer Delays

Figure 32-8 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 32-8. Programmable Delays



32.7.3.5 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.

[xxxxxxx(7-bit) + LASTXFER(1-bit)(1)+ xxxx(4-bit) + PCS (4-bit) + DATA (8 to 16-bit)] with PCS equals to the chip select to assert as defined in Section 32.8.4 (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 32.7.3.9 ”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 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 transfer can be started if the SPIEN was previously written in the SPI_CR register.

32.7.3.6 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.

32.7.3.7 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.

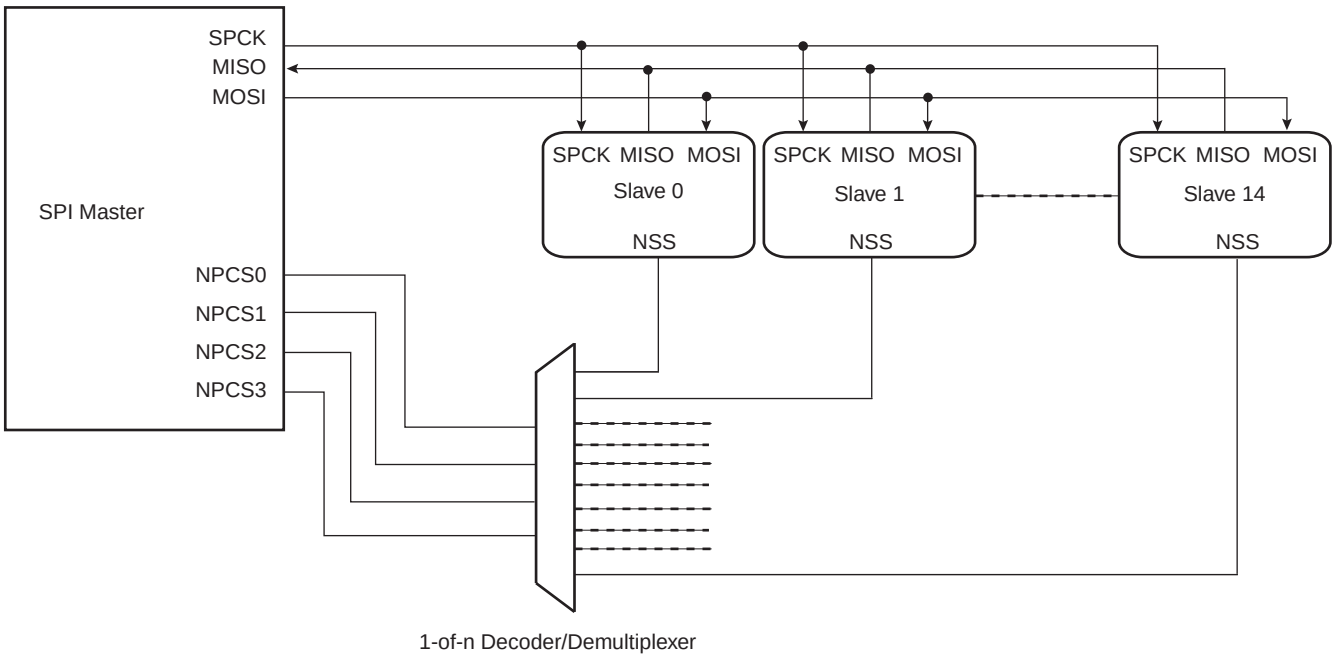
When operating with decoding, the SPI directly outputs the value defined by the PCS field on NPCS lines of either the Mode Register or the Transmit Data Register (depending on PS).

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 32-9 below shows such an implementation.

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 32-9. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation



32.7.3.8 Peripheral Deselection without DMAC

During a transfer of more than one data on a Chip Select without the DMAC, 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.

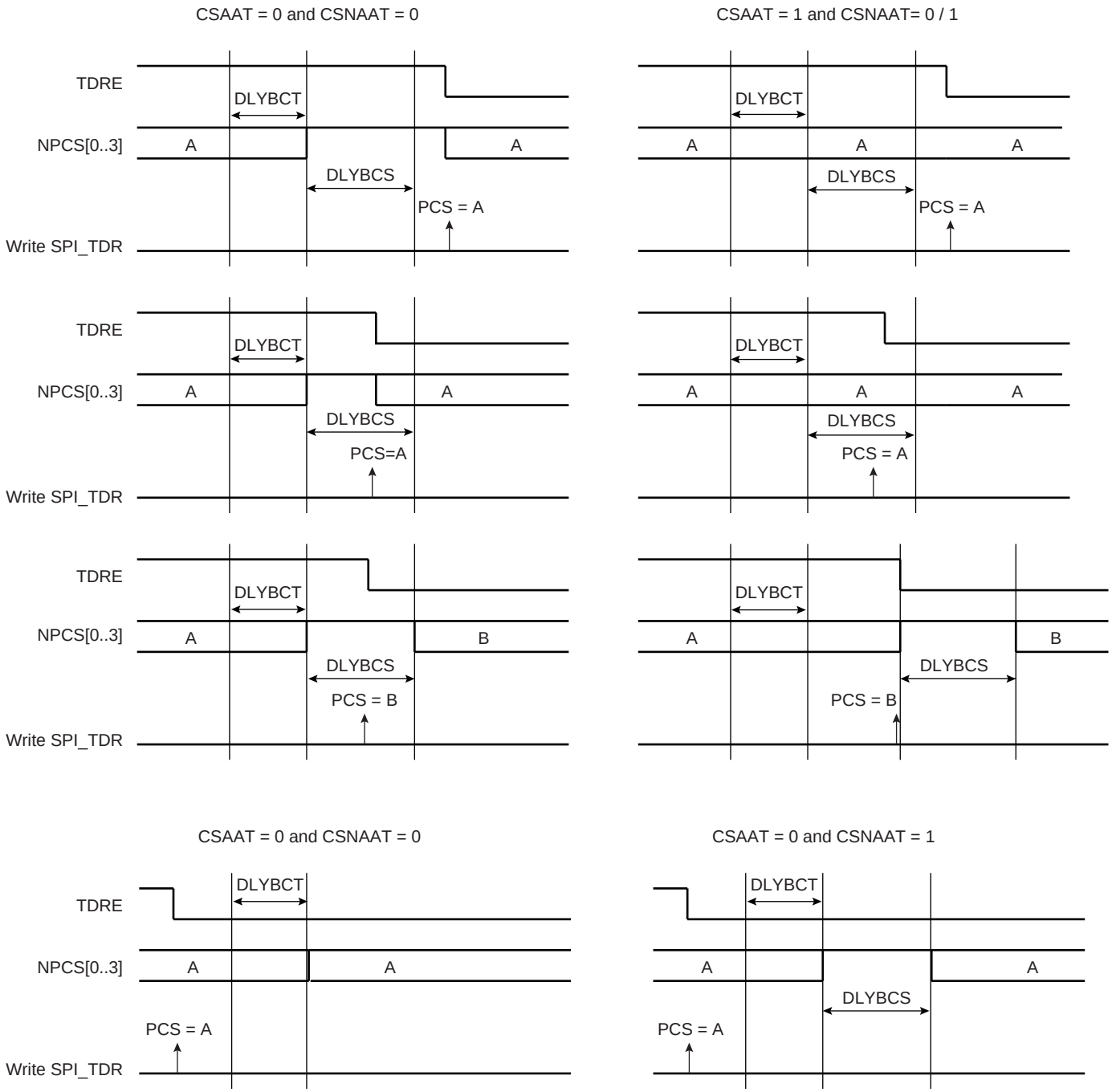
32.7.3.9 Peripheral Deselection with DMAC

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 32-10 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

Figure 32-10. Peripheral Deselection



32.7.3.10Mode 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).

32.7.4 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 32.8.9 “SPI Chip Select Register” on page 703.)

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.

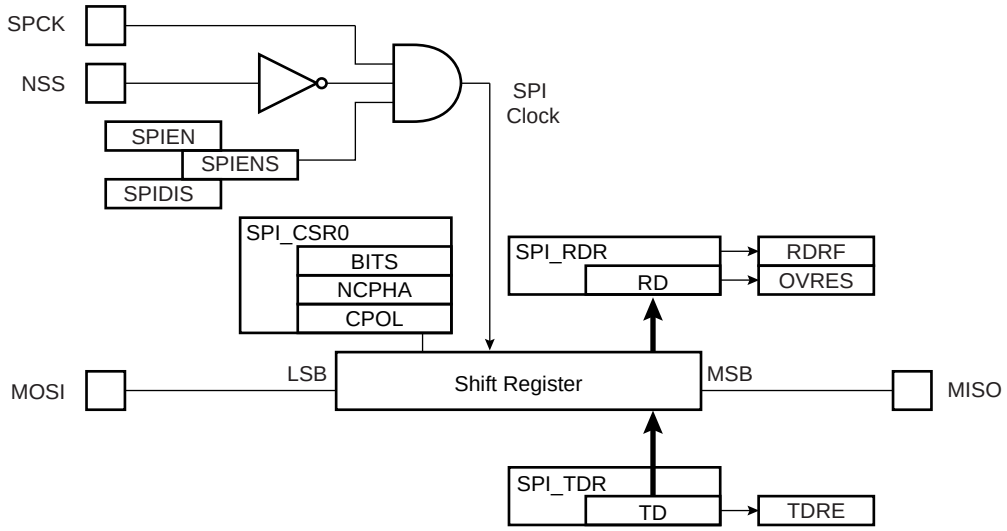
When a transfer starts, the data shifted out is the data present in the Shift Register. If no data has been written in the Transmit Data Register (SPI_TDR), the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift Register resets 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 Shift Register is not modified and the last received character is retransmitted. In this case the Underrun Error Status Flag (UNDES) is set in the SPI_SR.

Figure 32-11 shows a block diagram of the SPI when operating in Slave Mode.

Figure 32-11. Slave Mode Functional Bloc Diagram



32.7.5 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 WPVSRC 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 32.8.2 ”SPI Mode Register”

Section 32.8.9 ”SPI Chip Select Register”

32.8 Serial Peripheral Interface (SPI) User Interface

Table 32-5. 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
0x4C - 0xE0	Reserved	–	–	–
0xE4	Write Protection Control Register	SPI_WPMR	Read-write	0x0
0xE8	Write Protection Status Register	SPI_WPSR	Read-only	0x0
0x00E8 - 0x00F8	Reserved	–	–	–
0x00FC	Reserved	–	–	–

32.8.1 SPI Control Register

Name: SPI_CR

Address: 0x40008000 (0), 0x4000C000 (1)

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.

DMAC 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 32.7.3.5 ”Peripheral Selection” for more details.

32.8.2 SPI Mode Register

Name: SPI_MR

Address: 0x40008004 (0), 0x4000C004 (1)

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:

$$\text{Delay Between Chip Selects} = \frac{DLYBCS}{MCK}$$

32.8.3 SPI Receive Data Register

Name: SPI_RDR

Address: 0x40008008 (0), 0x4000C008 (1)

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.

32.8.4 SPI Transmit Data Register

Name: SPI_TDR

Address: 0x4000800C (0), 0x4000C00C (1)

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).

32.8.5 SPI Status Register

Name: SPI_SR

Address: 0x40008010 (0), 0x4000C010 (1)

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
–	–	–	–	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.

- **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.

32.8.6 SPI Interrupt Enable Register

Name: SPI_IER

Address: 0x40008014 (0), 0x4000C014 (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
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
–	–	–	–	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**
- **NSSR: NSS Rising Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **UNDES: Underrun Error Interrupt Enable**

32.8.7 SPI Interrupt Disable Register

Name: SPI_IDR

Address: 0x40008018 (0), 0x4000C018 (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
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
–	–	–	–	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**
- **NSSR: NSS Rising Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **UNDES: Underrun Error Interrupt Disable**

32.8.8 SPI Interrupt Mask Register

Name: SPI_IMR

Address: 0x4000801C (0), 0x4000C01C (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
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
–	–	–	–	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**
- **NSSR: NSS Rising Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **UNDES: Underrun Error Interrupt Mask**

32.8.9 SPI Chip Select Register

Name: SPI_CSRx[x=0..3]

Address: 0x40008030 (0), 0x4000C030 (1)

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.

- **CSNAAT: 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:

  - $\frac{DLYBCT}{MCK}$ (if DLYBCT field is different from 0)
  - $\frac{DLYBCT + 1}{MCK}$ (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 32.8.9 “SPI Chip Select Register” on page 703.)

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 inSPI_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:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times DLYBCT}{MCK}$$

32.8.10 SPI Write Protection Mode Register

Name: SPI_WPMR

Address: 0x400080E4 (0), 0x4000C0E4 (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: Write Protection Key Password**

If a value is written in WPEN, the value is taken into account only if WPKEY is written with “SPI” (SPI written in ASCII Code, ie 0x535049 in hexadecimal).

List of the write-protected registers:

Section 32.8.2 ”SPI Mode Register”

Section 32.8.9 ”SPI Chip Select Register”

32.8.11 SPI Write Protection Status Register

Name: SPI_WPSR

Address: 0x400080E8 (0), 0x4000C0E8 (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 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)

33. Two-wire Interface (TWI)

33.1 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 I²C 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. 20

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 33-1 lists the compatibility level of the Atmel Two-wire Interface in Master Mode and a full I2C compatible device.

Table 33-1. Atmel TWI compatibility with i2C Standard

I2C Standard	Atmel TWI
Standard Mode Speed (100 KHz)	Supported
Fast Mode Speed (400 KHz)	Supported
7 or 10 bits Slave Addressing	Supported
START BYTE(1)	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

33.2 Embedded Characteristics

- 2 x TWI
- Compatible with Atmel Two-wire Interface Serial Memory and I²C Compatible Devices(Note:)
- 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 in Master Mode Only
  - One Channel for the Receiver, One Channel for the Transmitter
  - Next Buffer Support

Note: See Table 33-1 for details on compatibility with I²C Standard.

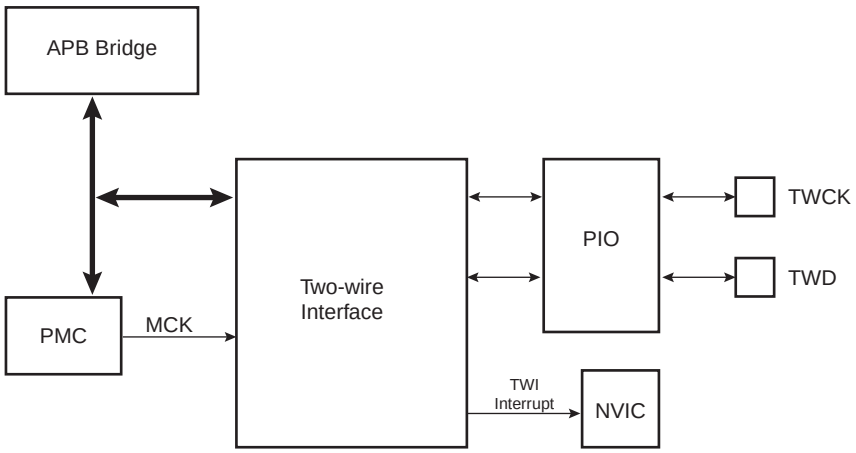
33.3 List of Abbreviations

Table 33-2. 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

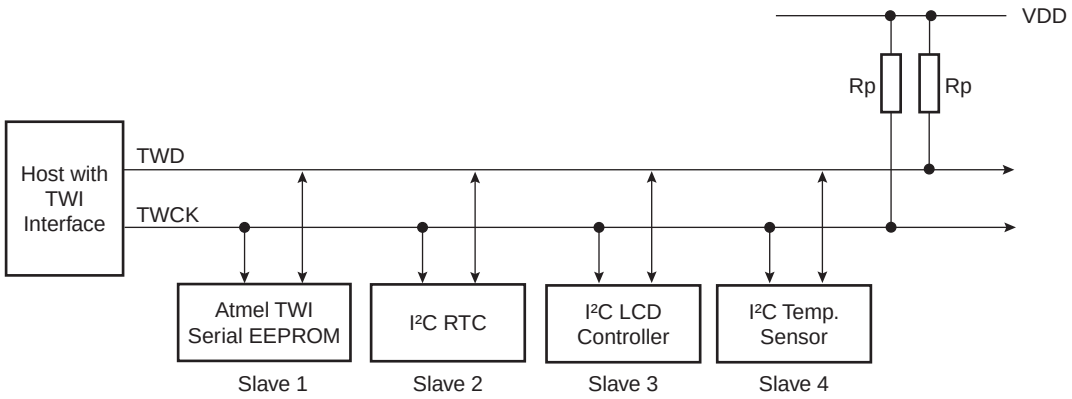
33.4 Block Diagram

Figure 33-1. Block Diagram



33.5 Application Block Diagram

Figure 33-2. Application Block Diagram



Rp: Pull up value as given by the I²C Standard

33.5.1 I/O Lines Description

Table 33-3. I/O Lines Description

Pin Name	Pin Description	Type
TWD	Two-wire Serial Data	Input/Output
TWCK	Two-wire Serial Clock	Input/Output

33.6 Product Dependencies

33.6.1 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 33-2 on page 710). 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 33-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TWI0	TWCK0	PA18	A
TWI0	TWD0	PA17	A
TWI1	TWCK1	PB13	A
TWI1	TWD1	PB12	A

33.6.2 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.

33.6.3 Interrupt

The TWI interface has an interrupt line connected to the Nested Vector Interrupt Controller (NVIC). In order to handle interrupts, the NVIC must be programmed before configuring the TWI.

Table 33-5. Peripheral IDs

Instance	ID
TWI0	22
TWI1	23

33.7 Functional Description

33.7.1 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 33-4).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 33-3).

- 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 33-3. START and STOP Conditions

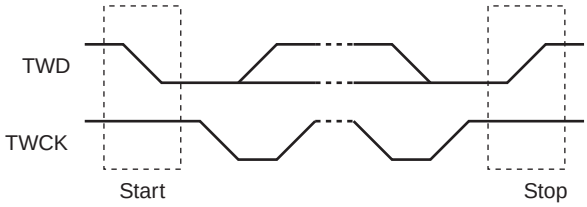
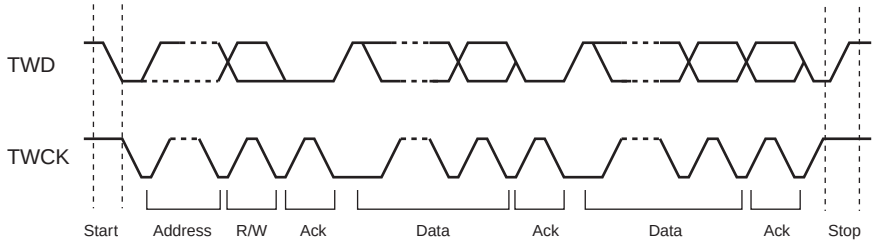


Figure 33-4. Transfer Format



33.7.2 Modes of Operation

The TWI has six 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.

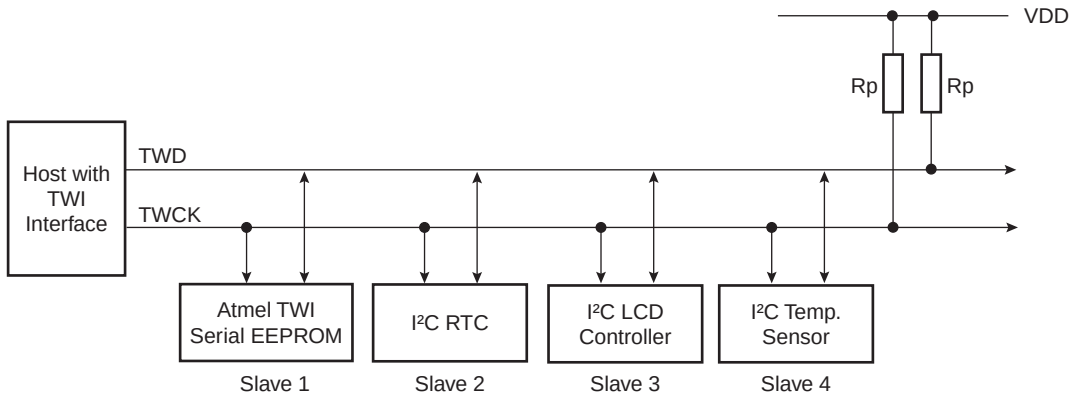
33.8 Master Mode

33.8.1 Definition

The Master is the device that starts a transfer, generates a clock and stops it.

33.8.2 Application Block Diagram

Figure 33-5. Master Mode Typical Application Block Diagram



Rp: Pull up value as given by the I²C Standard

33.8.3 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.

33.8.4 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 33-6, Figure 33-7, and Figure 33-8.

Figure 33-6. Master Write with One Data Byte

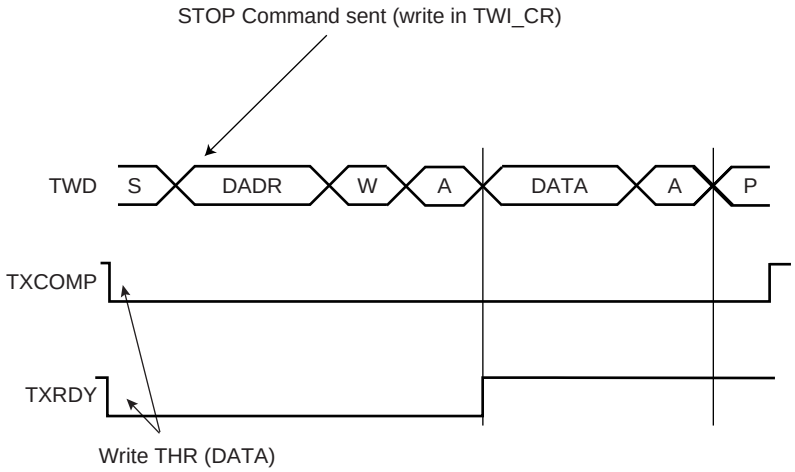


Figure 33-7. Master Write with Multiple Data Bytes

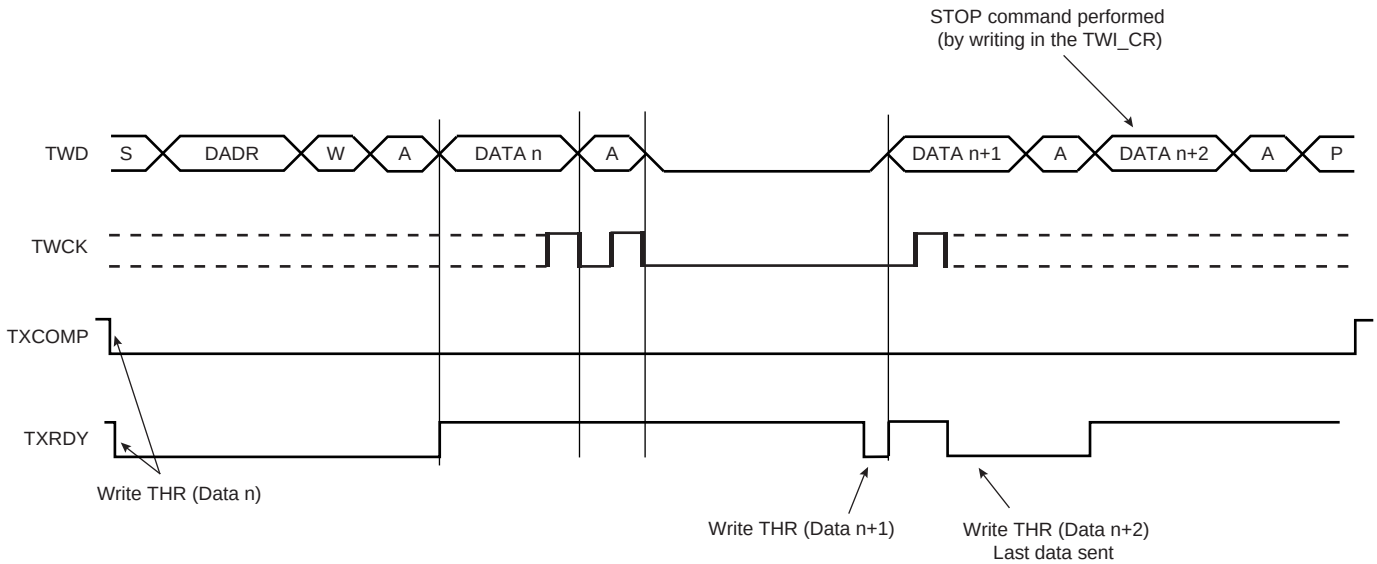
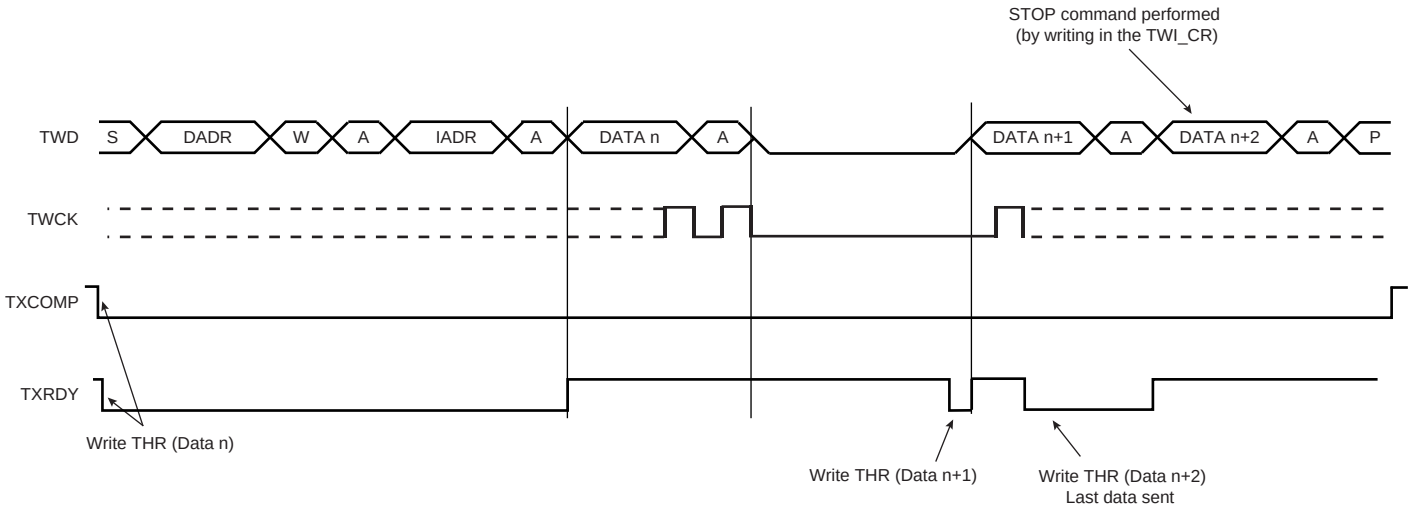


Figure 33-8. Master Write with One Byte Internal Address and Multiple Data Bytes



33.8.5 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 33-9. 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 33-9. 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 33-10. For Internal Address usage see Section 33.8.6.

Figure 33-9. Master Read with One Data Byte

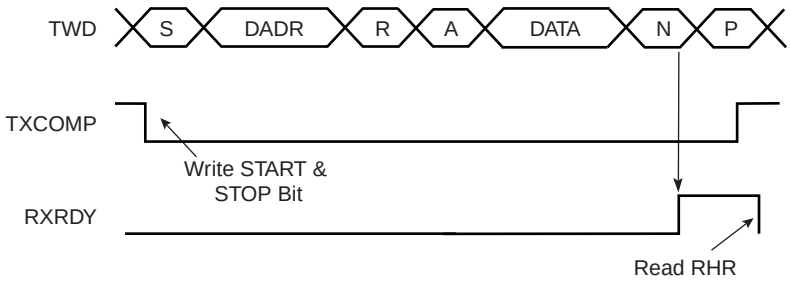
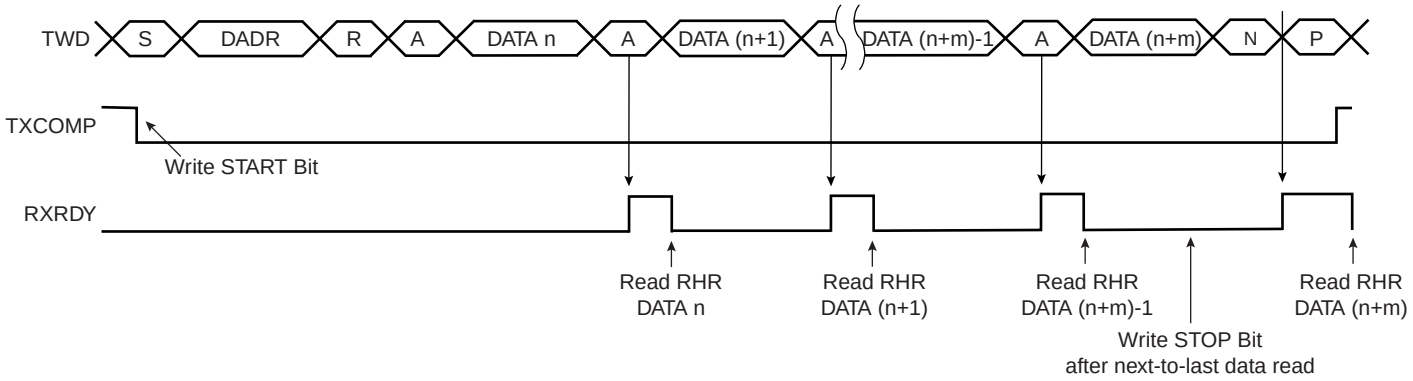


Figure 33-10. Master Read with Multiple Data Bytes



RXRDY is used as Receive Ready for the PDC receive channel.

33.8.6 Internal Address

The TWI interface can perform various transfer formats: Transfers with 7-bit slave address devices and 10-bit slave address devices.

33.8.6.1 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 I2C fully-compatible devices. See Figure 33-12. See Figure 33-11 and Figure 33-13 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
- N Not Acknowledge
- DADR Device Address
- IADR Internal Address

Figure 33-11. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

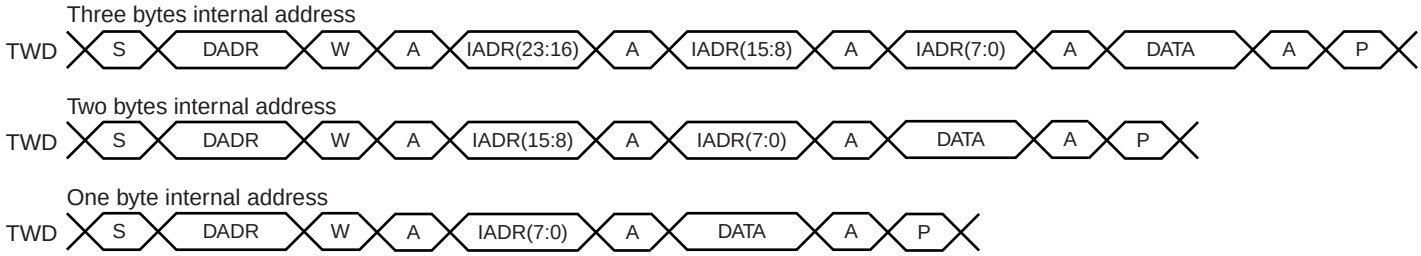
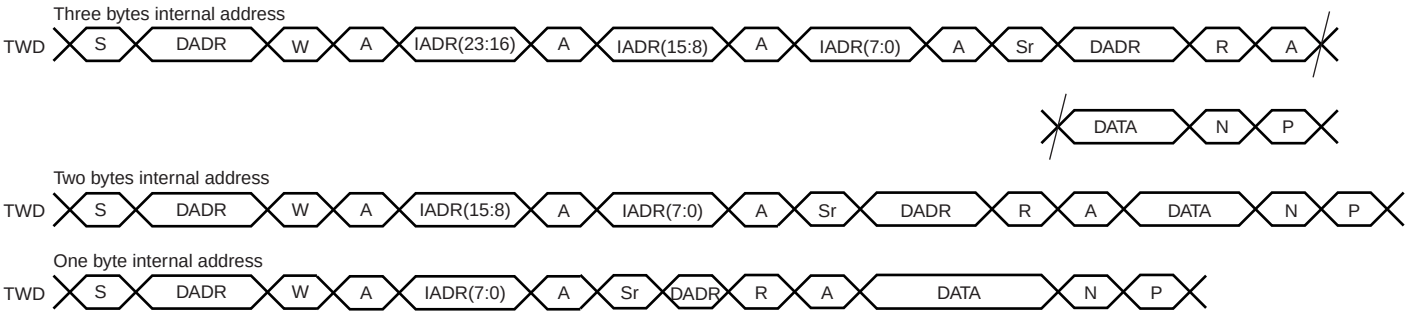


Figure 33-12. Master Read with One, Two or Three Bytes Internal Address and One Data Byte



33.8.6.2 10-bit Slave Addressing

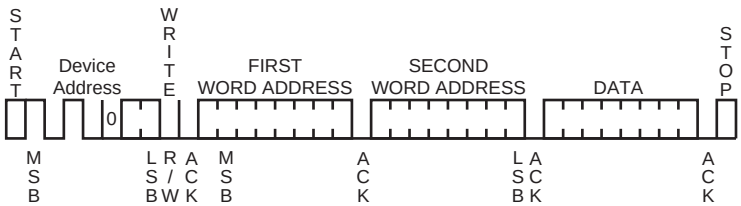
For a slave address higher than 7 bits, the user must configure the address size **(IADRSZ**) and set the other slave address bits in the internal address register (TWI_IADR). The two remaining Internal address bytes, IADR[15:8] and IADR[23:16] can be used the same as in 7-bit Slave Addressing.

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 33-13 below shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

Figure 33-13. Internal Address Usage



33.8.7 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:

33.8.7.1 Data Transmit with the PDC

1. Initialize the transmit PDC (memory pointers, size, etc.).
2. Configure the master mode (DADR, CKDIV, etc.).
3. Start the transfer by setting the PDC TXTEN bit.
4. Wait for the PDC end TX flag.
5. Disable the PDC by setting the PDC TXDIS bit.

33.8.7.2 Data Receive with the PDC

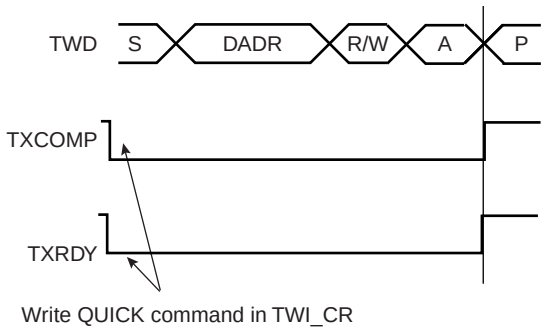
1. Initialize the receive PDC (memory pointers, size - 1, etc.).
2. Configure the master mode (DADR, CKDIV, etc.).
3. Start the transfer by setting the PDC RXTEN bit.
4. Wait for the PDC end RX flag.
5. Disable the PDC by setting the PDC RXDIS bit.

33.8.8 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 33-14. SMBUS Quick Command



33.8.9 Read-write Flowcharts

The following flowcharts shown in Figure 33-16 on page 720, Figure 33-17 on page 721, Figure 33-18 on page 722, Figure 33-19 on page 723 and Figure 33-20 on page 724 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 33-15. TWI Write Operation with Single Data Byte without Internal Address

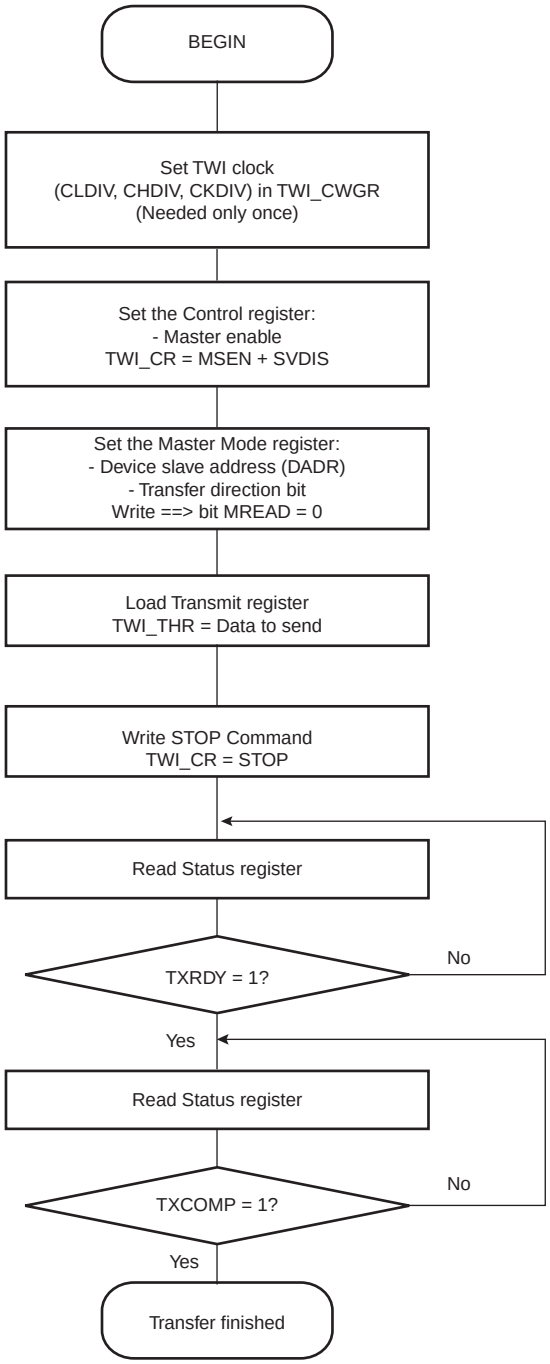


Figure 33-16. TWI Write Operation with Single Data Byte and Internal Address

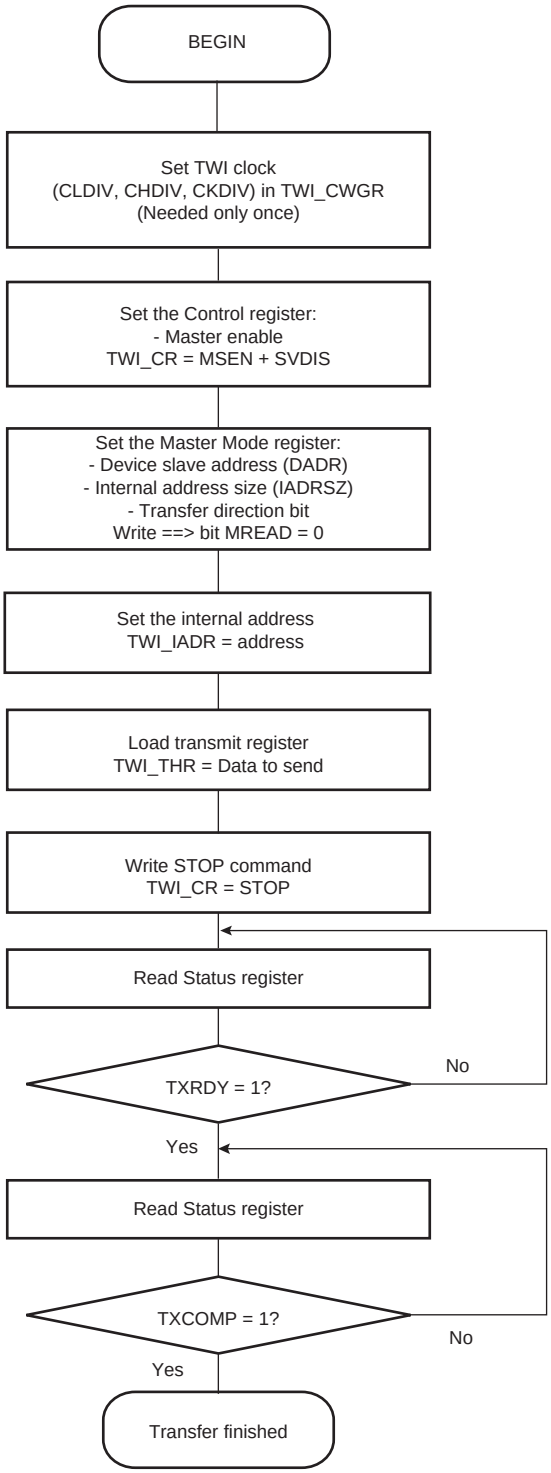


Figure 33-17. TWI Write Operation with Multiple Data Bytes with or without Internal Address

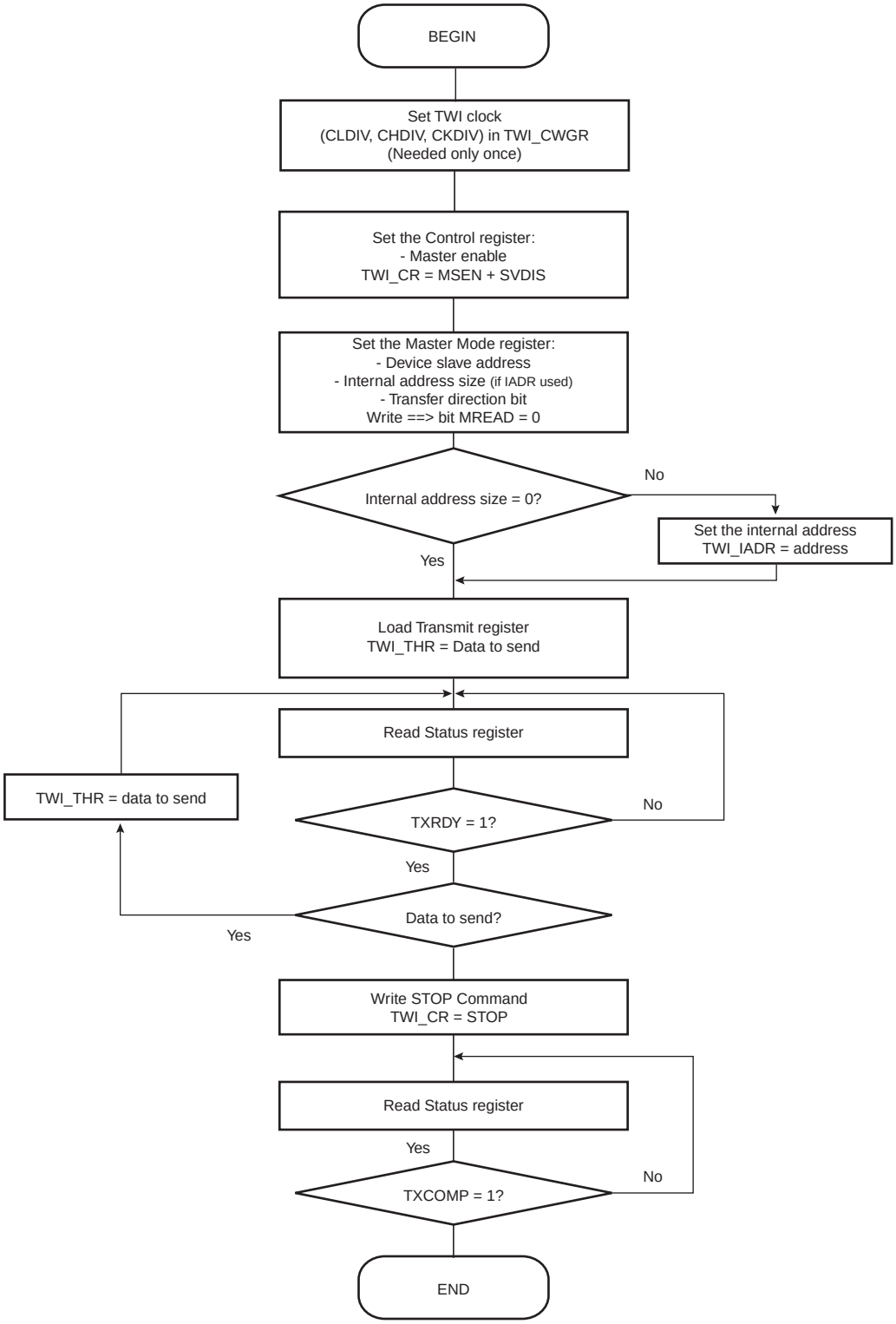


Figure 33-18. TWI Read Operation with Single Data Byte without Internal Address

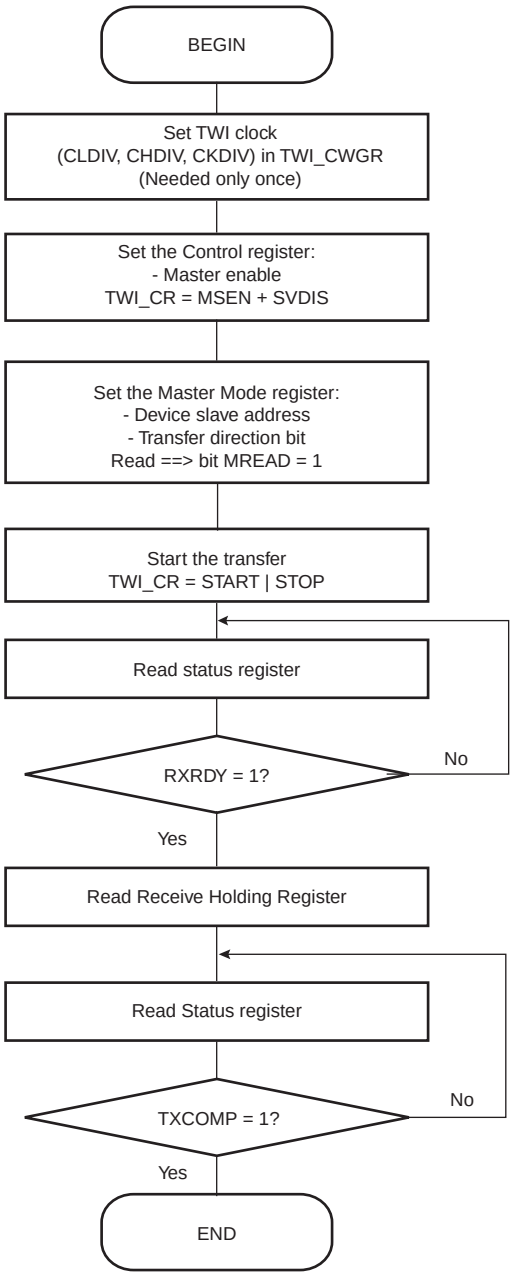


Figure 33-19. TWI Read Operation with Single Data Byte and Internal Address

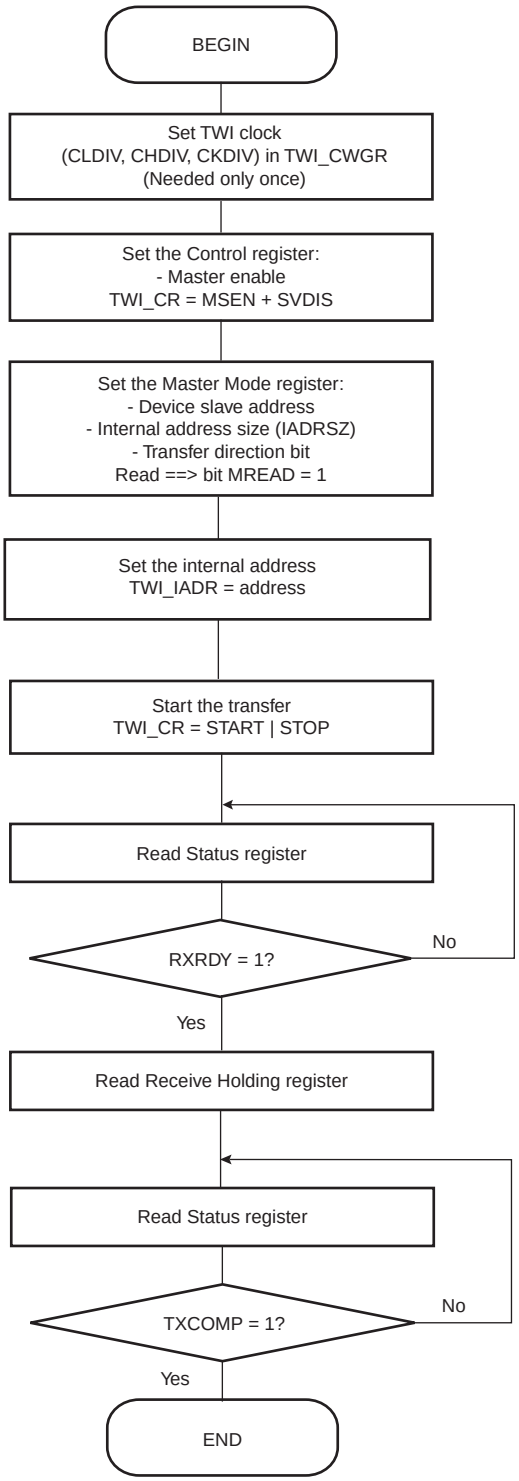
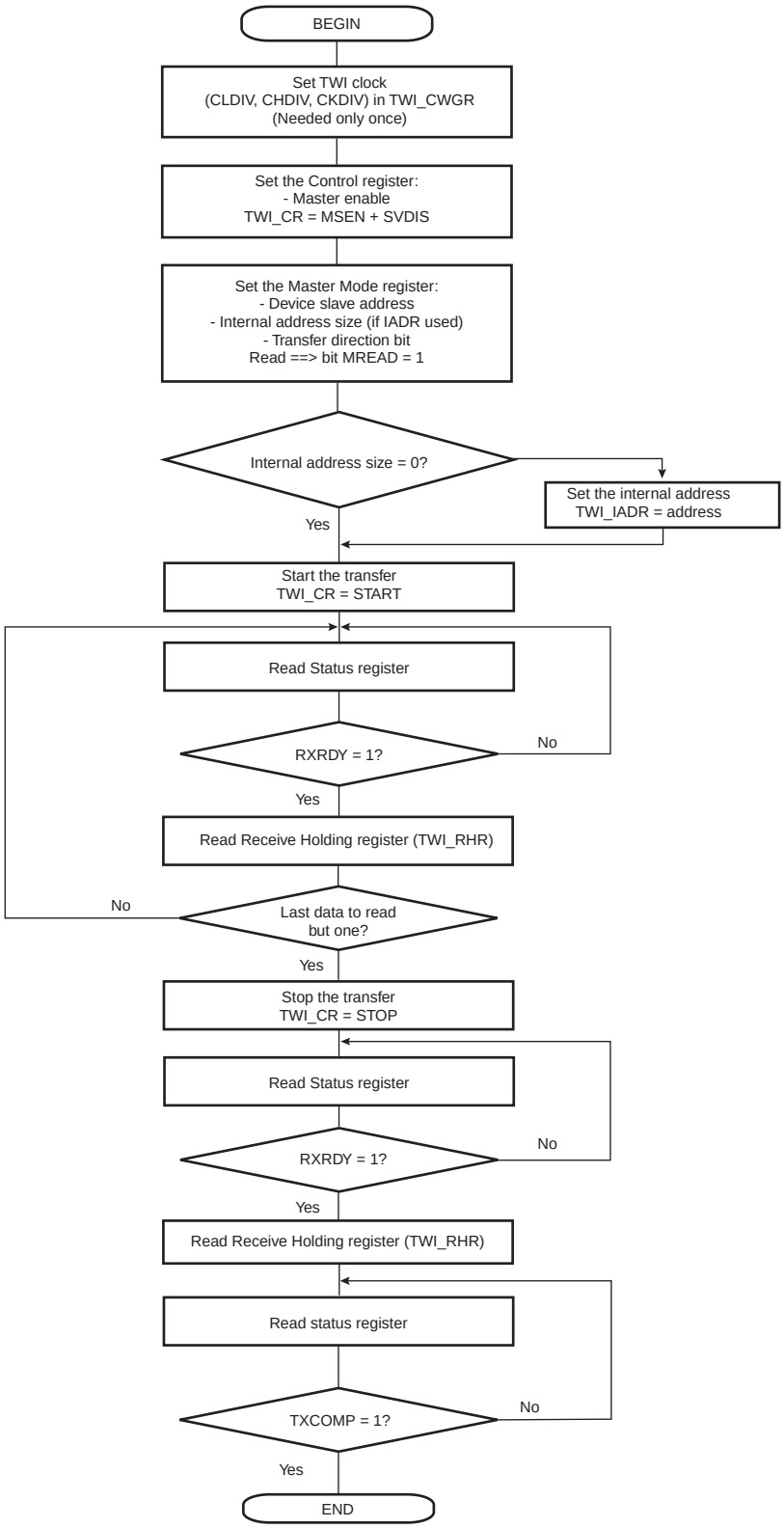


Figure 33-20. TWI Read Operation with Multiple Data Bytes with or without Internal Address



33.9 Multi-master Mode

33.9.1 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 33-22 on page 726.

33.9.2 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.

33.9.2.1 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 33-21 on page 726).

Note: The state of the bus (busy or free) is not indicated in the user interface.

33.9.2.2 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 33-21. Programmer Sends Data While the Bus is Busy

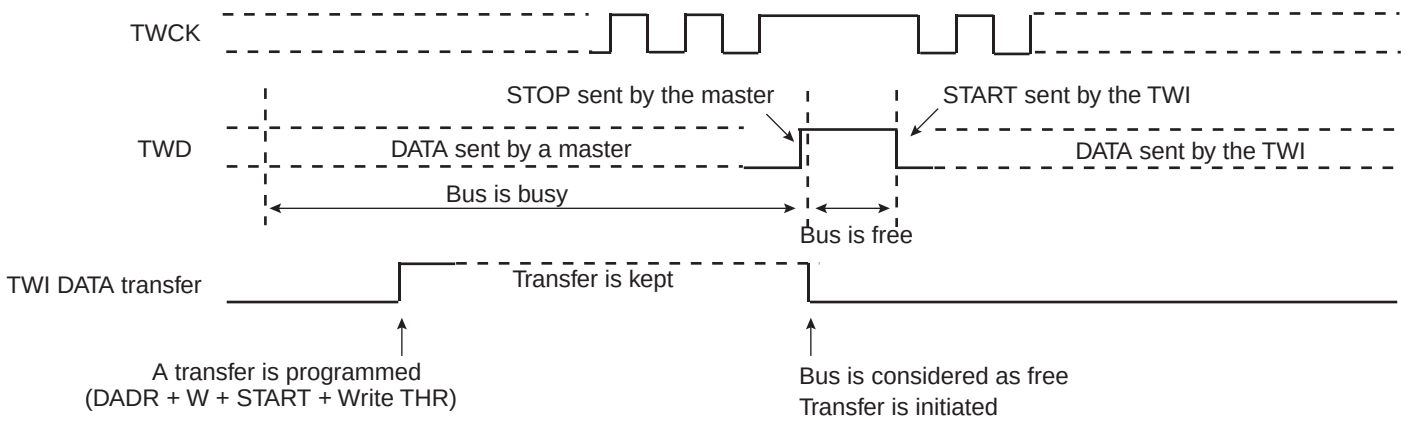
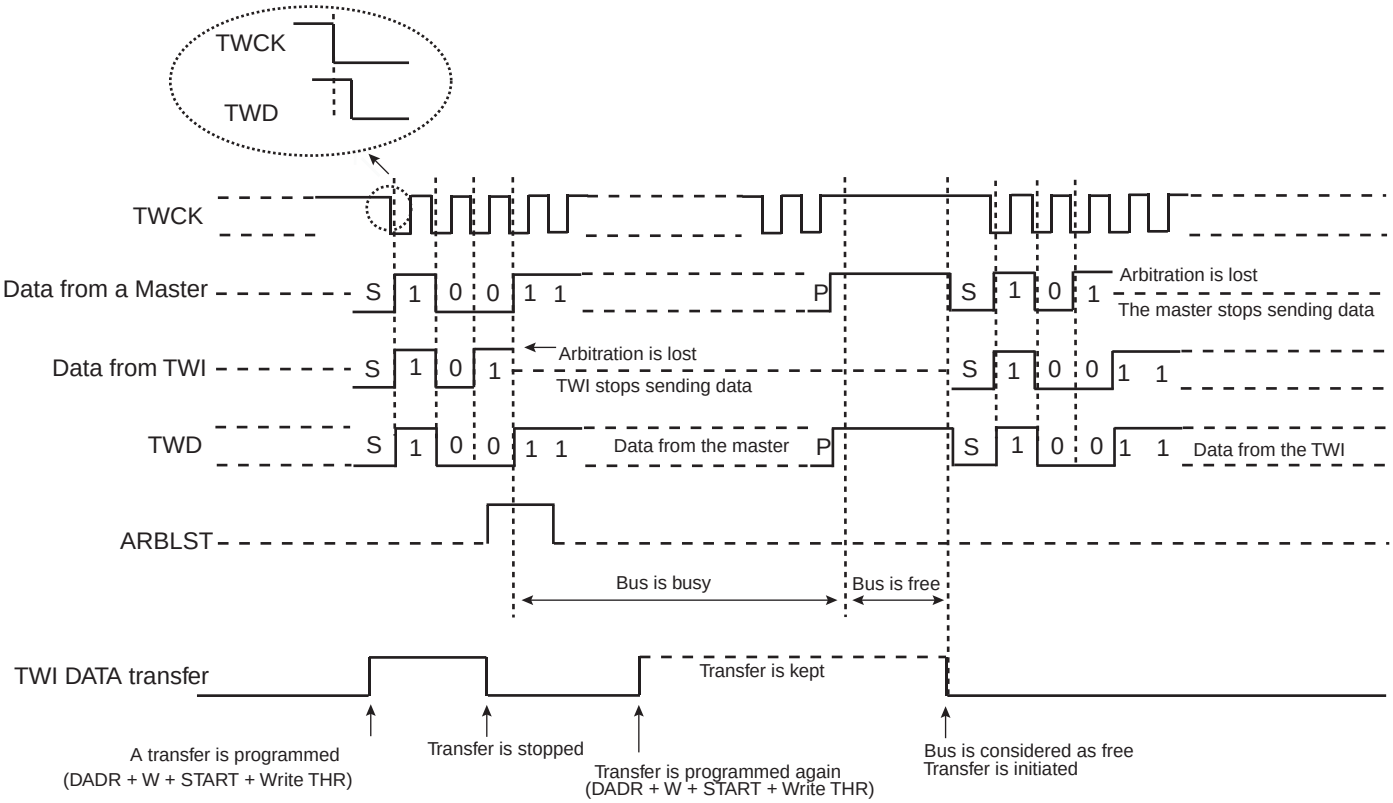
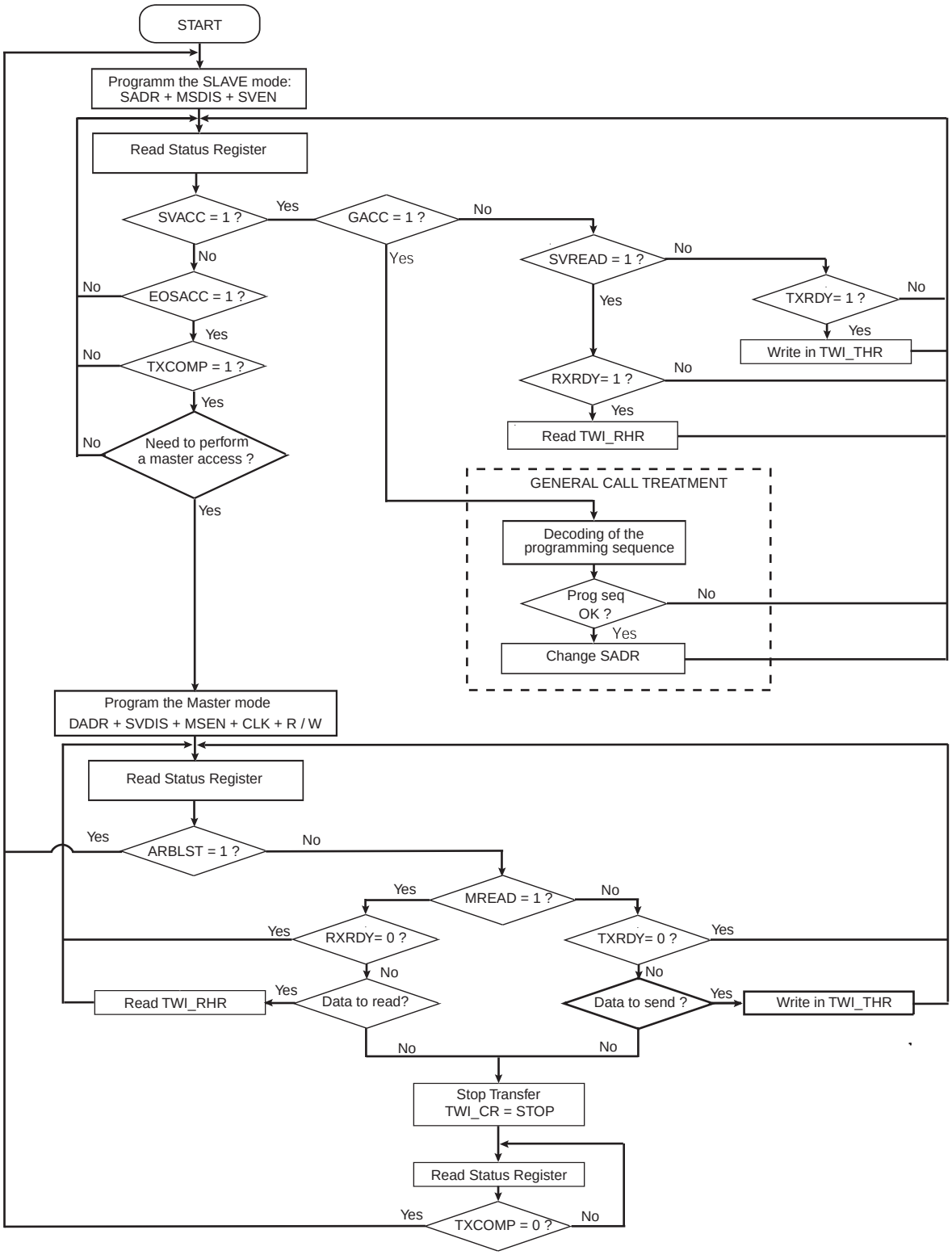


Figure 33-22. Arbitration Cases



The flowchart shown in Figure 33-23 on page 727 gives an example of read and write operations in Multi-master mode.

Figure 33-23. Multi-master Flowchart



33.10 Slave Mode

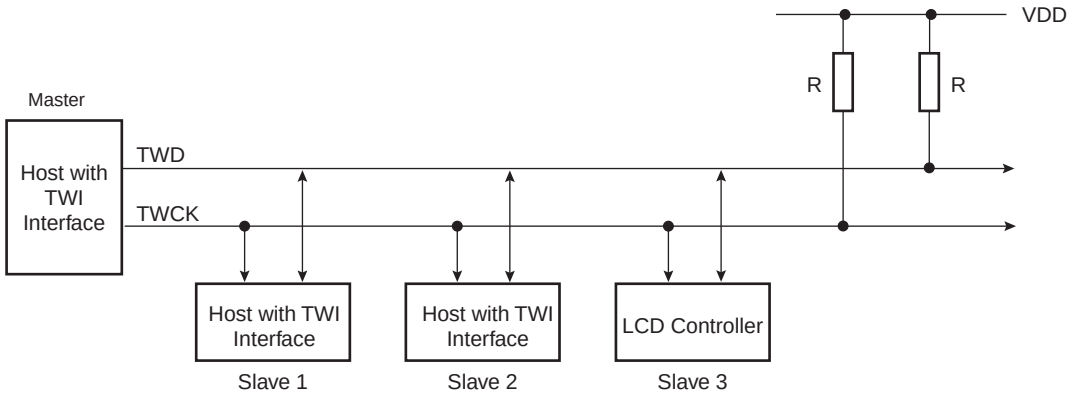
33.10.1 Definition

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).

33.10.2 Application Block Diagram

Figure 33-24. Slave Mode Typical Application Block Diagram



33.10.3 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.

33.10.4 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.

33.10.4.1Read 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 33-25 on page 730.

33.10.4.2Write 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 33-26 on page 730.

33.10.4.3Clock 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 33-28 on page 731 and Figure 33-29 on page 732.

33.10.4.4General 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 33-27 on page 731.

33.10.4.5PDC

As it is impossible to know the exact number of data to receive/send, the use of PDC is NOT recommended in SLAVE mode.

33.10.5 Data Transfer

33.10.5.1Read Operation

The read mode is defined as a data requirement from the master.

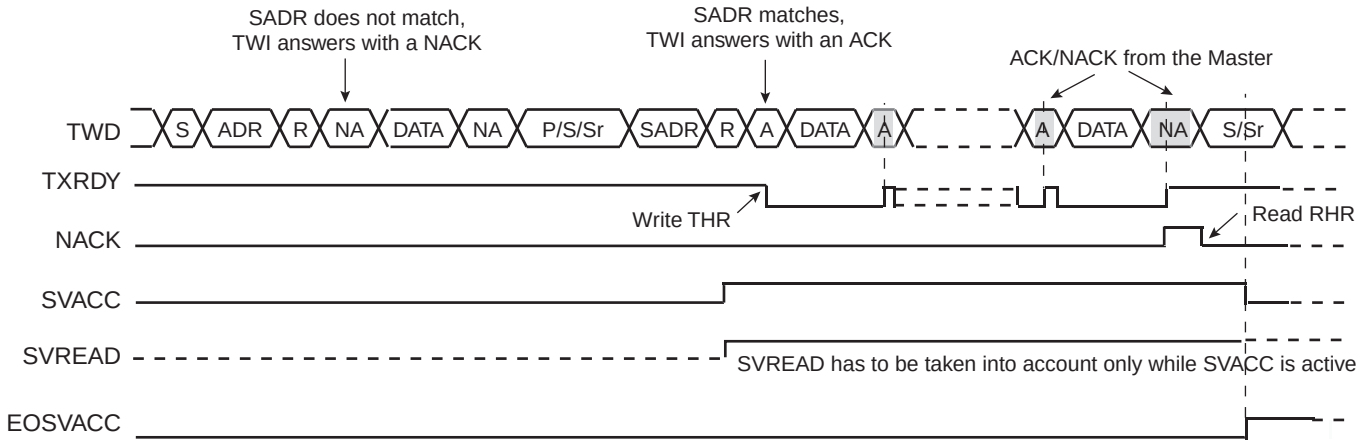
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

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 33-25 on page 730 describes the write operation.

Figure 33-25. 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.

33.10.5.2Write Operation

The write mode is defined as a data transmission from the master.

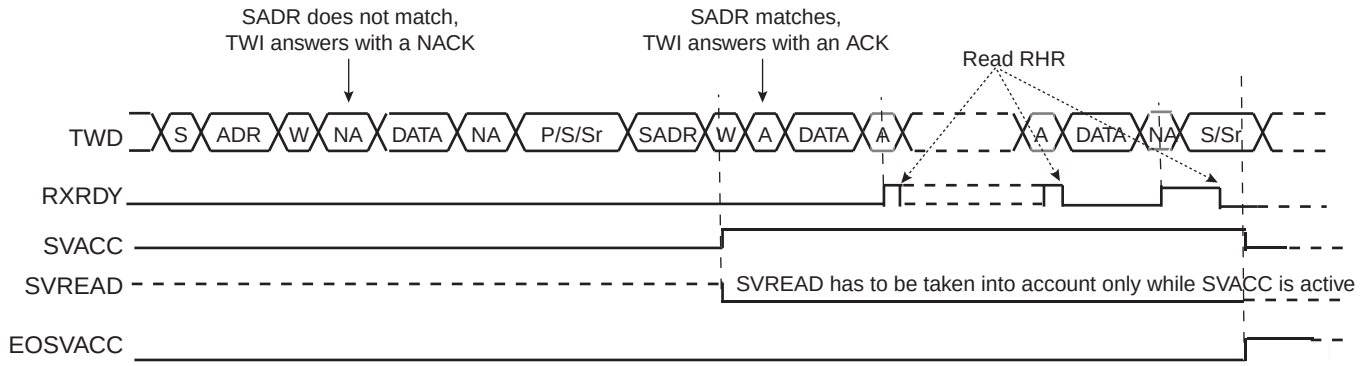
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

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 33-26 on page 730 describes the Write operation.

Figure 33-26. 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.

33.10.5.3General Call

The general call is performed in order to change the address of the slave.

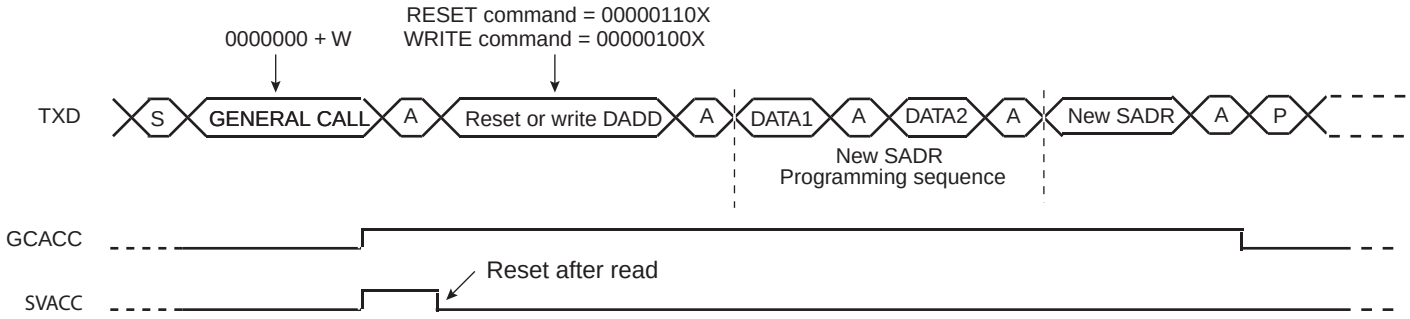
If a GENERAL CALL is detected, GACC is set.

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 33-27 on page 731 describes the General Call access.

Figure 33-27. 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.

33.10.5.4Clock Synchronization

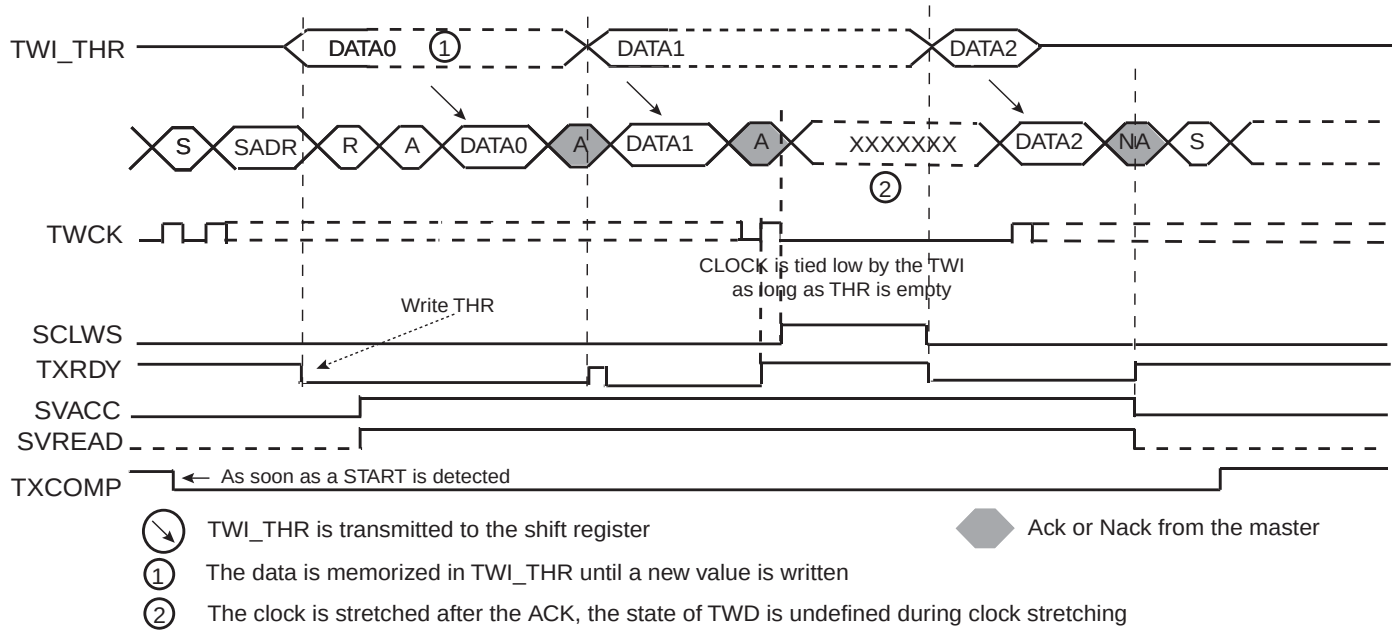
In both read and write modes, it may happen that TWI_THR/TWI_RHR buffer is not filled /emptied before the emission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching mechanism is implemented.

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 33-28 on page 731 describes the clock synchronization in Read mode.

Figure 33-28. 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.

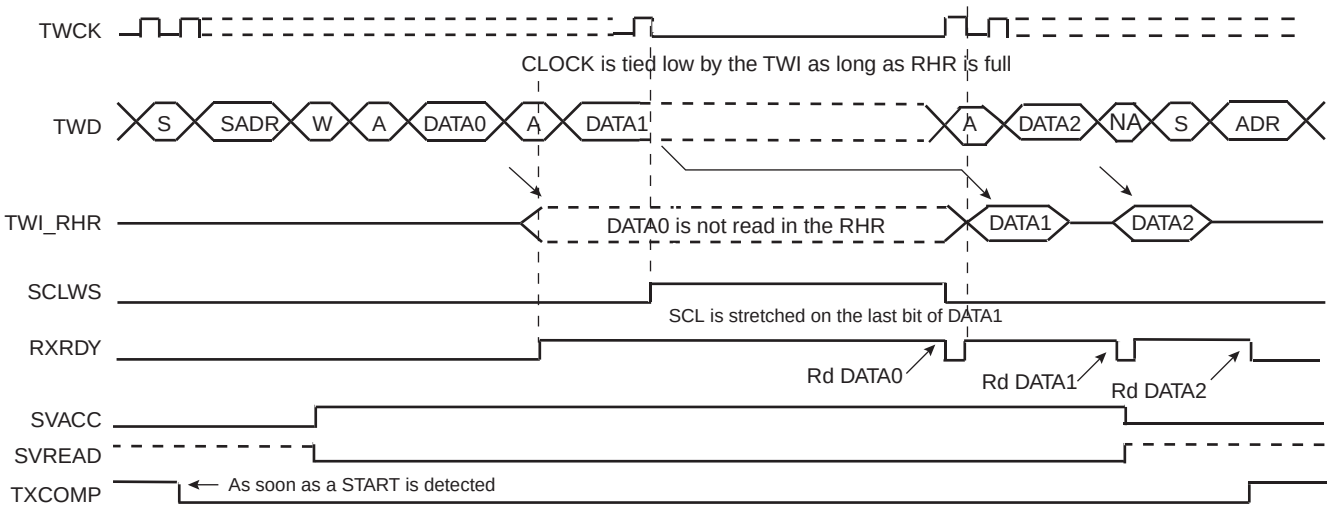
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock synchronization mechanism is started.

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 33-29 on page 732 describes the clock synchronization in Read mode.

Figure 33-29. 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.

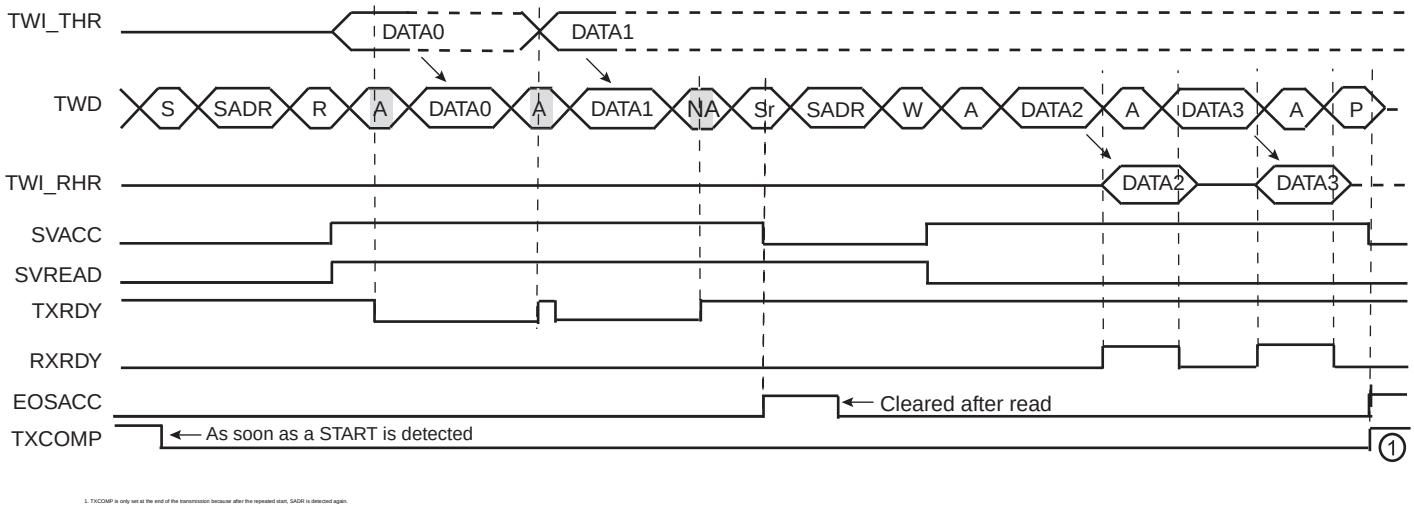
33.10.5.5Reversal after a Repeated Start

Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

Figure 33-30 on page 733 describes the repeated start + reversal from Read to Write mode.

Figure 33-30. Repeated Start + Reversal from Read to Write Mode

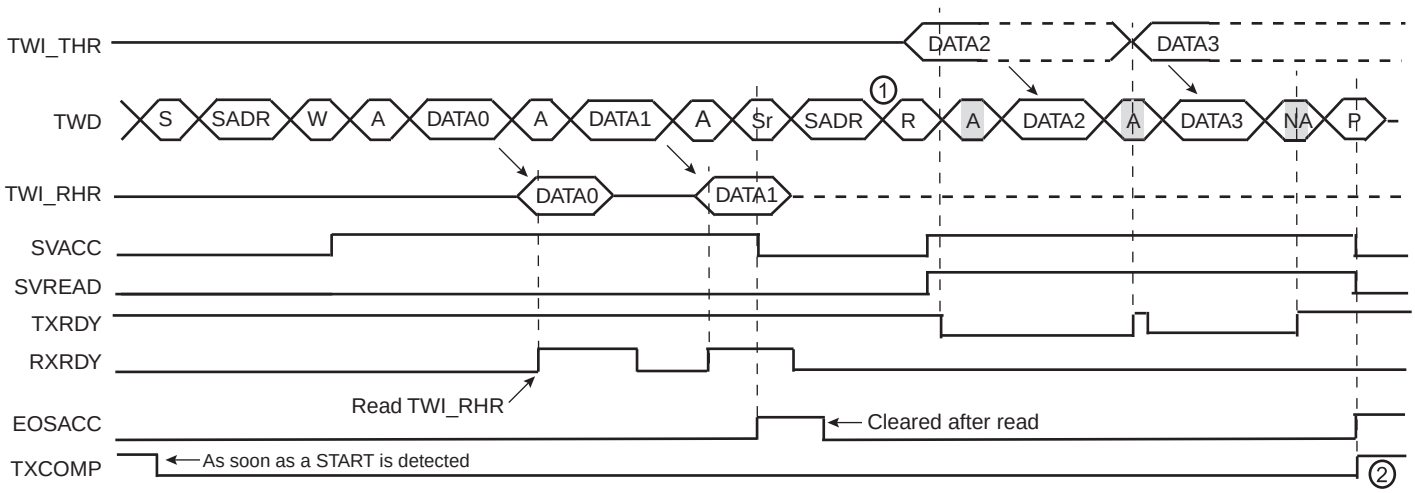


1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command.Figure 33-31 on page 733 describes the repeated start + reversal from Write to Read mode.

Figure 33-31. Repeated Start + Reversal from Write to Read Mode

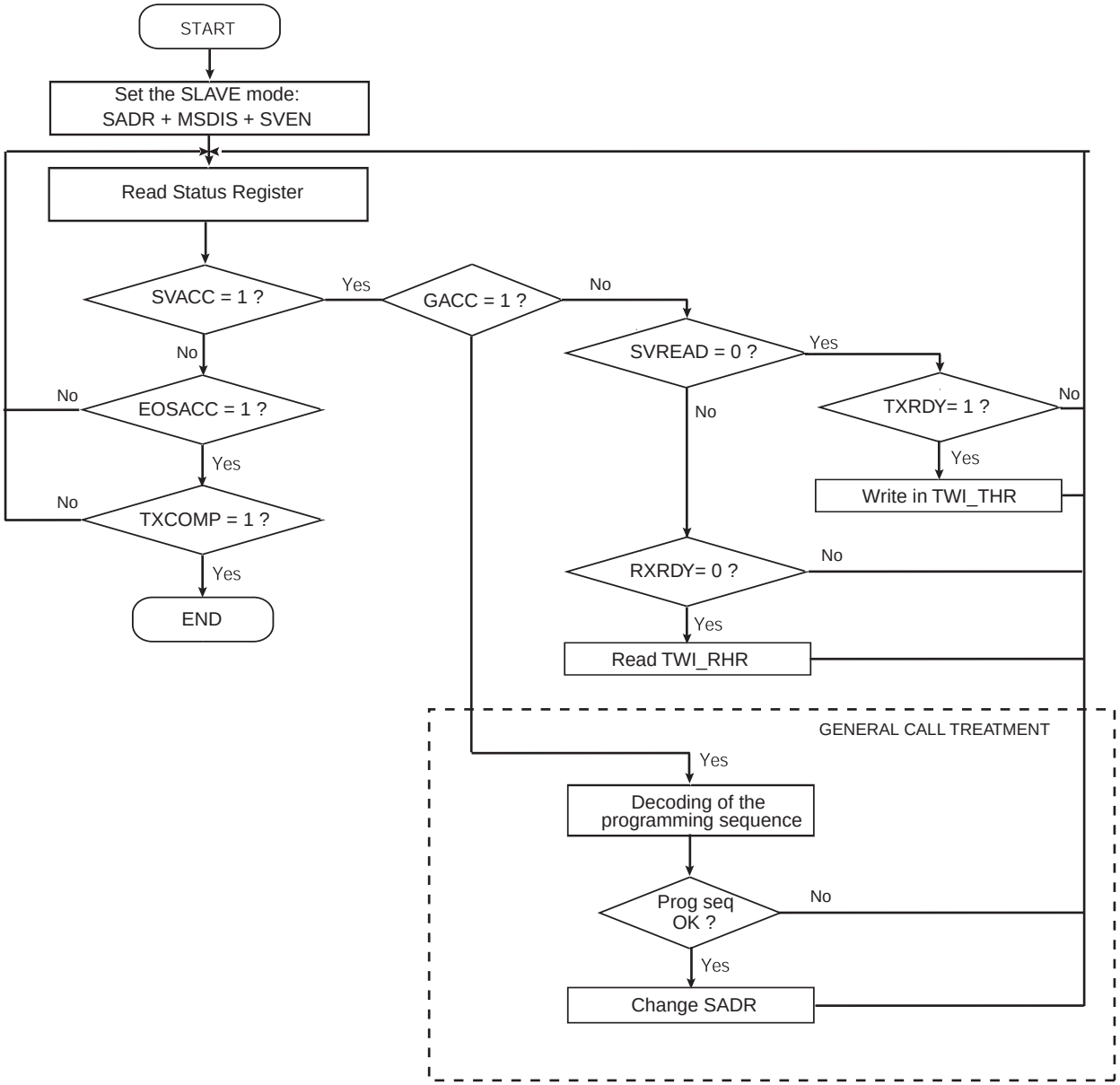


Notes:
1. In this case, if TWI_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.
2. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

33.10.6 Read Write Flowcharts

The flowchart shown in Figure 33-32 on page 734 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 33-32. Read Write Flowchart in Slave Mode



33.11 Two-wire Interface (TWI) User Interface

Table 33-6. 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
0xEC - 0xFC(1)	Reserved	–	–	–
0x100 - 0x124	Reserved for the PDC	–	–	–

Note: 1. All unlisted offset values are conisedered as “reserved”.

33.11.1 TWI Control Register

Name: TWI_CR

Address: 0x4008C000 (0), 0x40090000 (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.

33.11.2 TWI Master Mode Register

Name: TWI_MMR

Address: 0x4008C004 (0), 0x40090004 (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_BYTE	One-byte internal device address
2	2_BYTE	Two-byte internal device address
3	3_BYTE	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.

33.11.3 TWI Slave Mode Register

Name: TWI_SMR

Address: 0x4008C008 (0), 0x40090008 (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
–	–	–	–	–	–	–	–

- **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.

33.11.4 TWI Internal Address Register

Name: TWI_IADR

Address: 0x4008C00C (0), 0x4009000C (1)

Access: Read-write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
IADR							

15	14	13	12	11	10	9	8
IADR							

7	6	5	4	3	2	1	0
IADR							

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

33.11.5 TWI Clock Waveform Generator Register

Name: TWI_CWGR

Address: 0x4008C010 (0), 0x40090010 (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							

TWI_CWGR is only used in Master mode.

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$T_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 4) \times T_{MCK}$$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$T_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 4) \times T_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

33.11.6 TWI Status Register

Name: TWI_SR

Address: 0x4008C020 (0), 0x40090020 (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 33-8 on page 715 and in Figure 33-10 on page 716.

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 33-28 on page 731, Figure 33-29 on page 732, Figure 33-30 on page 733 and Figure 33-31 on page 733.

- **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 33-10 on page 716.

RXRDY behavior in Slave mode can be seen in Figure 33-26 on page 730, Figure 33-29 on page 732, Figure 33-30 on page 733 and Figure 33-31 on page 733.

- **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 33-8 on page 715.

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 33-25 on page 730, Figure 33-28 on page 731, Figure 33-30 on page 733 and Figure 33-31 on page 733.

- **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 33-25 on page 730, Figure 33-26 on page 730, Figure 33-30 on page 733 and Figure 33-31 on page 733.

- **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 33-25 on page 730, Figure 33-26 on page 730, Figure 33-30 on page 733 and Figure 33-31 on page 733.

- **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 33-27 on page 731.

- **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 33-28 on page 731 and Figure 33-29 on page 732.

- **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 33-30 on page 733 and Figure 33-31 on page 733

- **ENDRX: End of RX buffer**

This bit is only used in Master mode.

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**

This bit is only used in Master mode.

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**

This bit is only used in Master mode.

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**

This bit is only used in Master mode.

0 = TWI_TCR or TWI_TNCR have a value other than 0.

1 = Both TWI_TCR and TWI_TNCR have a value of 0.

33.11.7 TWI Interrupt Enable Register

Name: TWI_IER

Address: 0x4008C024 (0), 0x40090024 (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.

33.11.8 TWI Interrupt Disable Register

Name: TWI_IDR

Address: 0x4008C028 (0), 0x40090028 (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.

33.11.9 TWI Interrupt Mask Register

Name: TWI_IMR

Address: 0x4008C02C (0), 0x4009002C (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.

33.11.10TWI Receive Holding Register

Name: TWI_RHR

Address: 0x4008C030 (0), 0x40090030 (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**

33.11.11TWI Transmit Holding Register

Name: TWI_THR

Address: 0x4008C034 (0), 0x40090034 (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**

34. Universal Asynchronous Receiver Transceiver (UART)

34.1 Description

The Universal Asynchronous Receiver Transmitter 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 two peripheral DMA controller (PDC) channels permits packet handling for these tasks with processor time reduced to a minimum.

34.2 Embedded Characteristics

- Two-pin UART
  - Implemented Features are USART Compatible
  - 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
  - Interrupt Generation
  - Support for Two PDC Channels with Connection to Receiver and Transmitter

34.3 Block Diagram

Figure 34-1. UART Functional Block Diagram

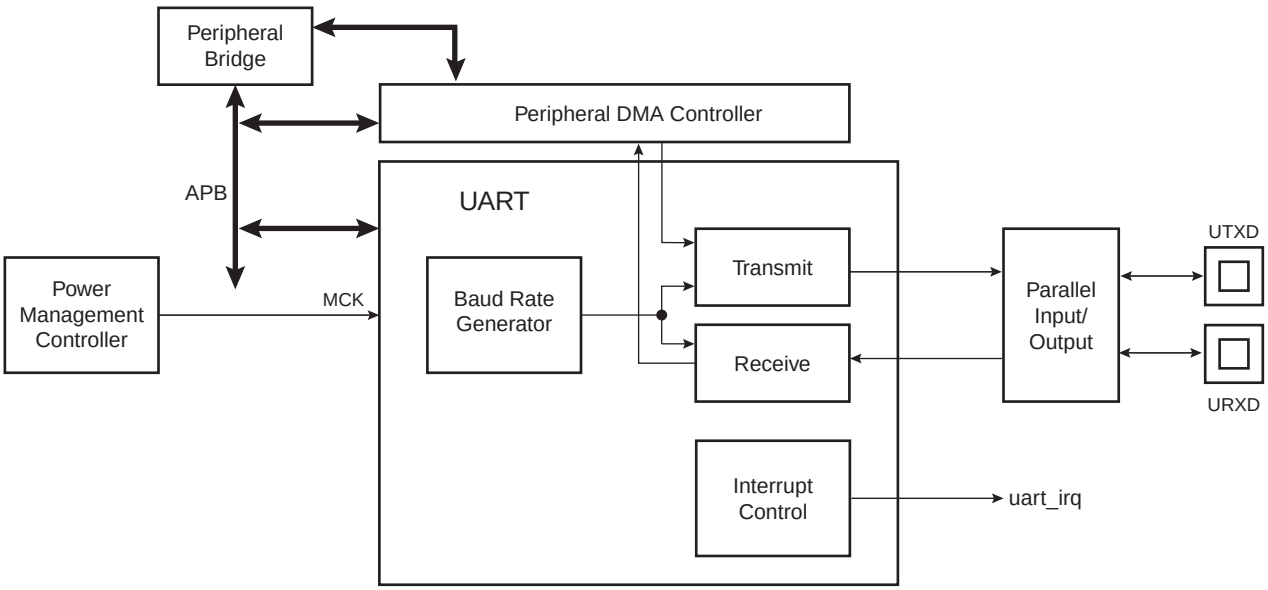


Table 34-1. UART Pin Description

Pin Name	Description	Type
URXD	UART Receive Data	Input
UTXD	UART Transmit Data	Output

34.4 Product Dependencies

34.4.1 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 34-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
UART	URXD	PA8	A
UART	UTXD	PA9	A

34.4.2 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.

34.4.3 Interrupt Source

The UART interrupt line is connected to one of the interrupt sources of the Nested Vectored Interrupt Controller (NVIC). Interrupt handling requires programming of the NVIC before configuring the UART.

34.5 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.

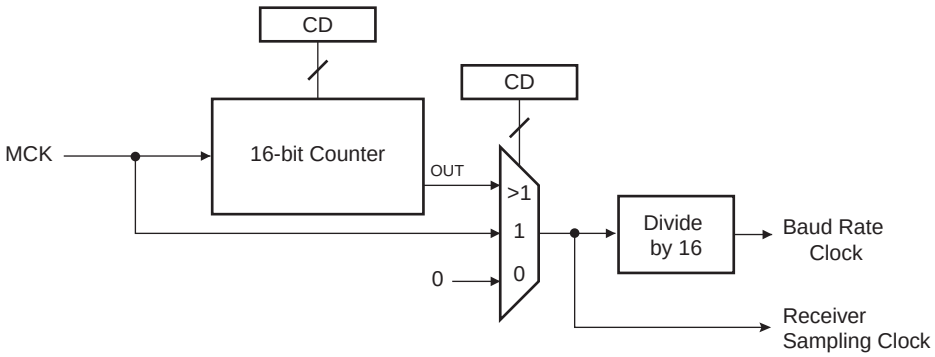
34.5.1 Baud Rate Generator

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 34-2. Baud Rate Generator



34.5.2 Receiver

34.5.2.1 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.

34.5.2.2 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 34-3. Start Bit Detection

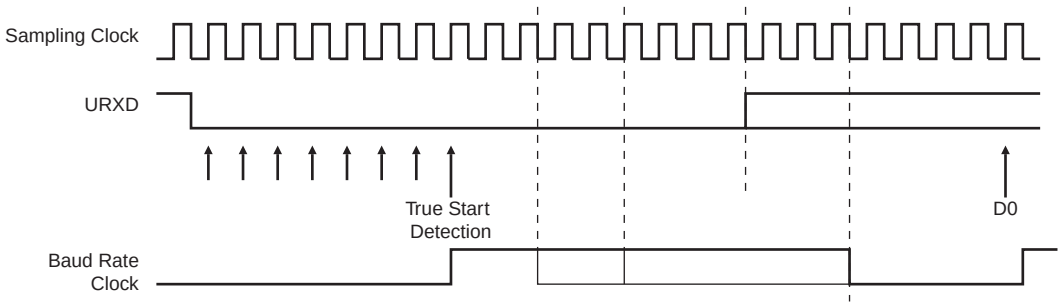
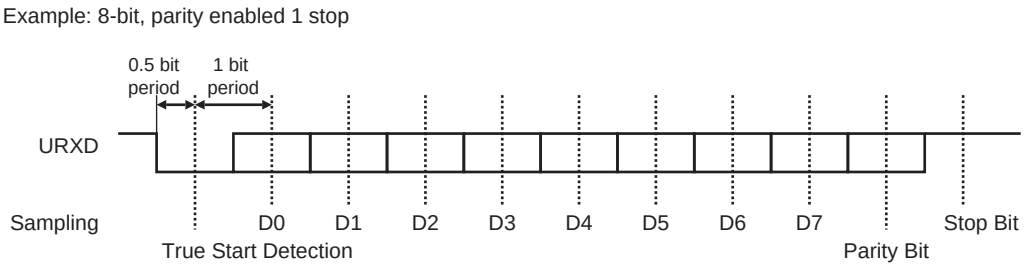


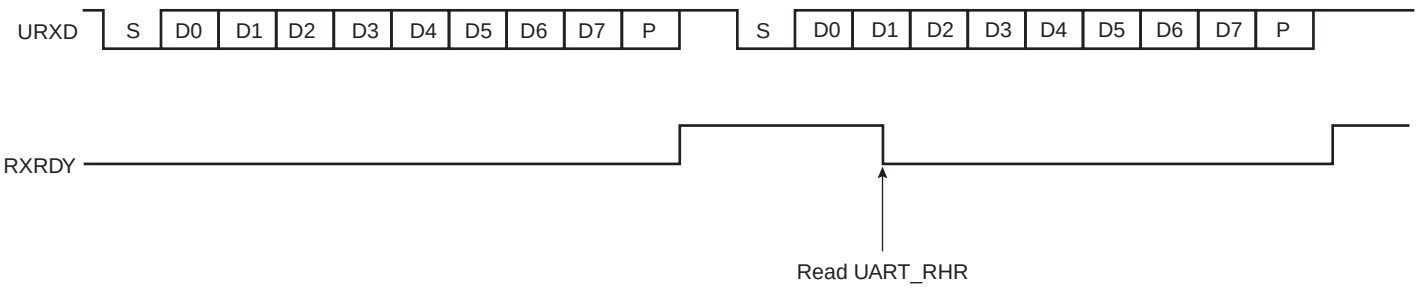
Figure 34-4. Character Reception



34.5.2.3 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.

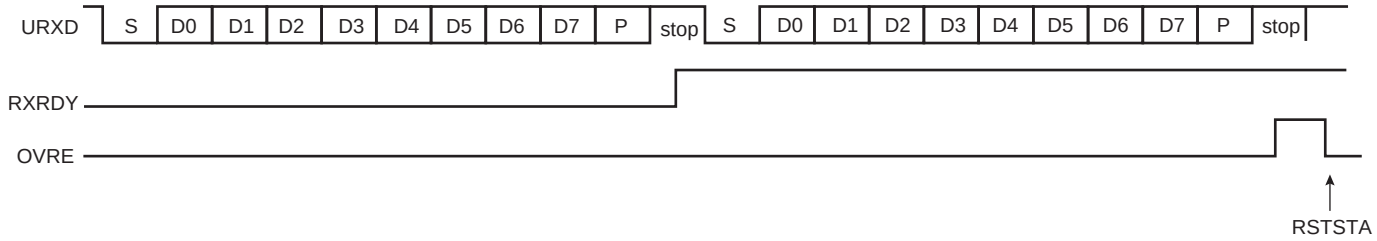
Figure 34-5. Receiver Ready



34.5.2.4 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.

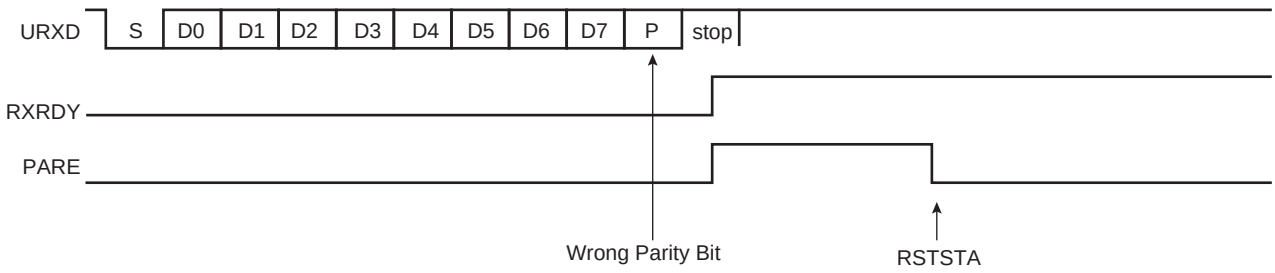
Figure 34-6. Receiver Overrun



34.5.2.5 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.

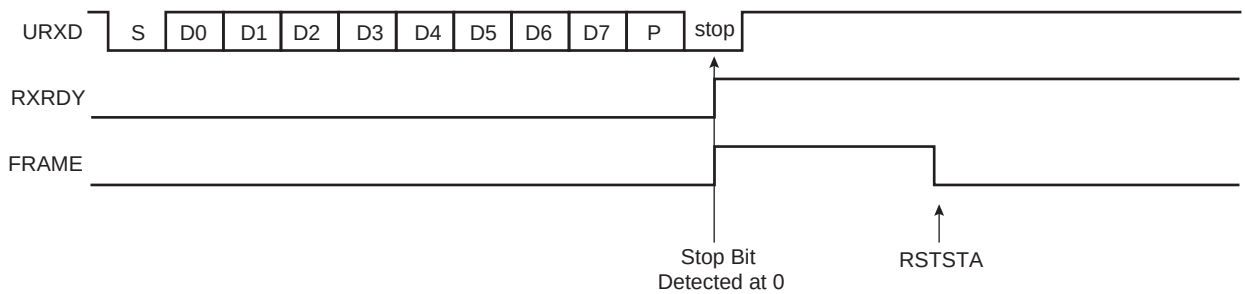
Figure 34-7. Parity Error



34.5.2.6 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 34-8. Receiver Framing Error



34.5.3 Transmitter

34.5.3.1 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.

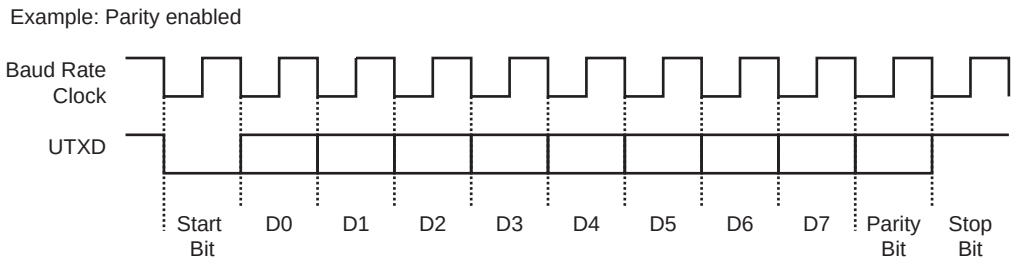
The programmer can disable the transmitter by writing UART_CR with the bit TXDIS at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the Shift Register and/or a character has been written in the Transmit Holding Register, the characters 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.

34.5.3.2 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 34-9. Character Transmission

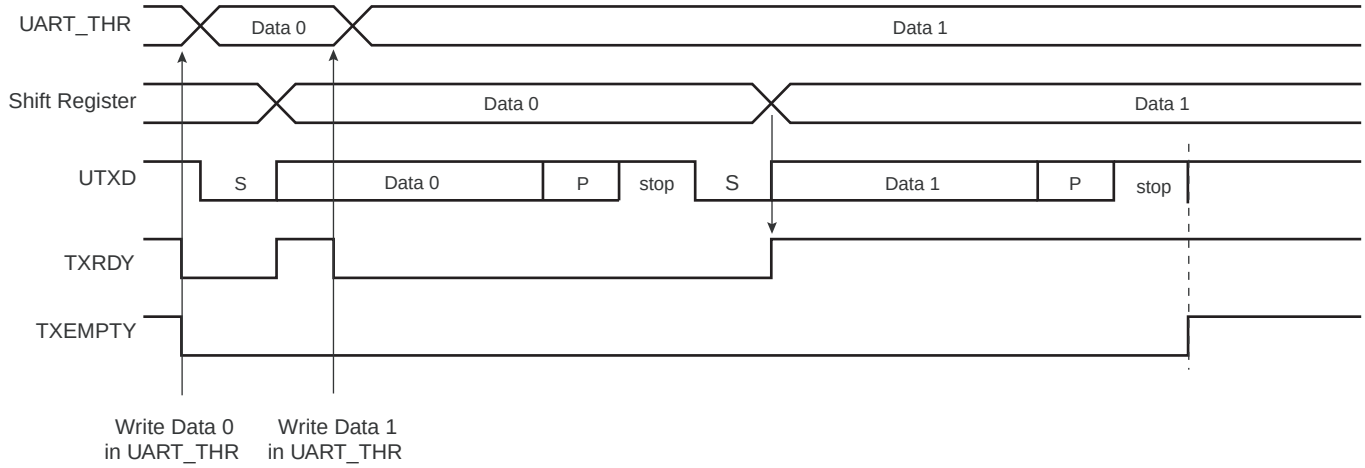


34.5.3.3 Transmitter Control

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 34-10. Transmitter Control



34.5.4 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.

34.5.5 Test Modes

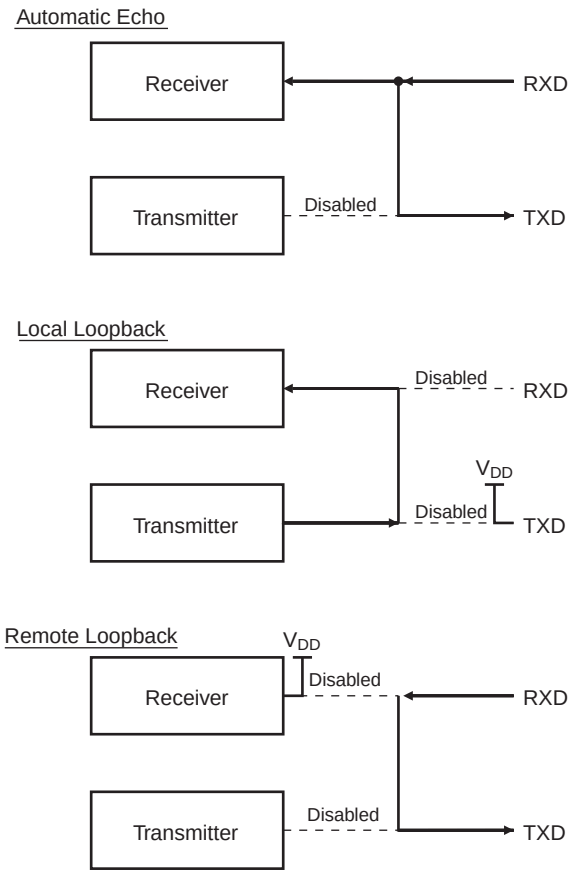
The UART supports three test modes. These modes of operation are programmed by using the field CHMODE (Channel Mode) in the mode register (UART_MR).

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 34-11. Test Modes



34.6 Universal Asynchronous Receiver Transceiver (UART) User Interface

Table 34-3. 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	PDC Area	–	–	–

34.6.1 UART Control Register

Name: UART_CR

Address: 0x400E0800

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.

34.6.2 UART Mode Register

Name: UART_MR

Address: 0x400E0804

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

34.6.3 UART Interrupt Enable Register

Name: UART_IER

Address: 0x400E0808

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.

34.6.4 UART Interrupt Disable Register

Name: UART_IDR

Address: 0x400E080C

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.

34.6.5 UART Interrupt Mask Register

Name: UART_IMR

Address: 0x400E0810

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.

34.6.6 UART Status Register

Name: UART_SR

Address: 0x400E0814

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.

34.6.7 UART Receiver Holding Register

Name: UART_RHR

Address: 0x400E0818

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.

34.6.8 UART Transmit Holding Register

Name: UART_THR

Address: 0x400E081C

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.

34.6.9 UART Baud Rate Generator Register

Name: UART_BRGR

Address: 0x400E0820

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)

35. Universal Synchronous Asynchronous Receiver Transmitter (USART)

35.1 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, LIN and SPI buses, with ISO7816 T = 0 or T = 1 smart card slots and infrared transceivers. 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 Peripheral DMA Controller, which enables data transfers to the transmitter and from the receiver. The PDC provides chained buffer management without any intervention of the processor.

35.2 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
  - 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
- LIN Mode (USART0 only)
  - Compliant with LIN 1.3 and LIN 2.0 specifications
  - Master or Slave
  - Processing of frames with up to 256 data bytes
  - Response Data length can be configurable or defined automatically by the Identifier
  - Self synchronization in Slave node configuration
  - Automatic processing and verification of the “Synch Break” and the “Synch Field”
  - The “Synch Break” is detected even if it is partially superimposed with a data byte
  - Automatic Identifier parity calculation/sending and verification
  - Parity sending and verification can be disabled
  - Automatic Checksum calculation/sending and verification
  - Checksum sending and verification can be disabled
  - Support both “Classic” and “Enhanced” checksum types
  - Full LIN error checking and reporting
  - Frame Slot Mode: the Master allocates slots to the scheduled frames automatically.
  - Generation of the Wakeup signal
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of Two Peripheral DMA Controller Channels (PDC)
  - Offers Buffer Transfer without Processor Intervention

35.3 Block Diagram

Figure 35-1. USART Block Diagram

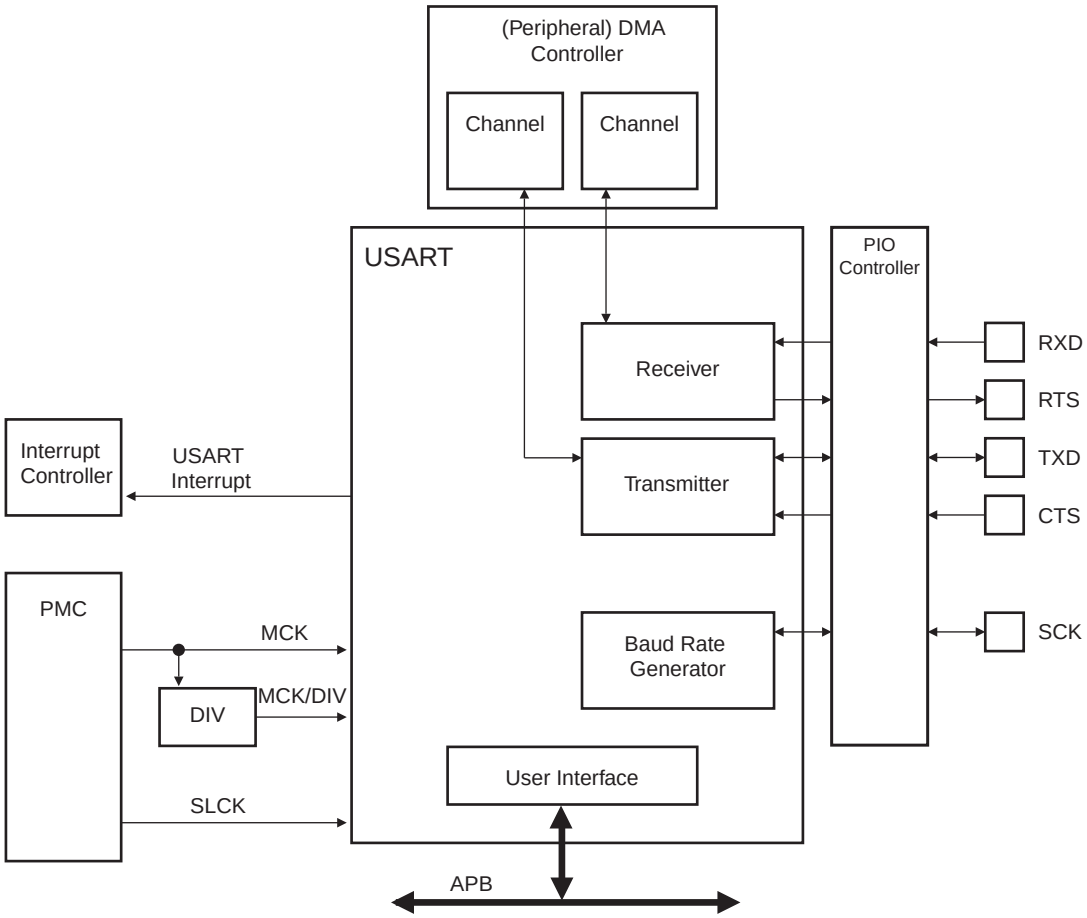
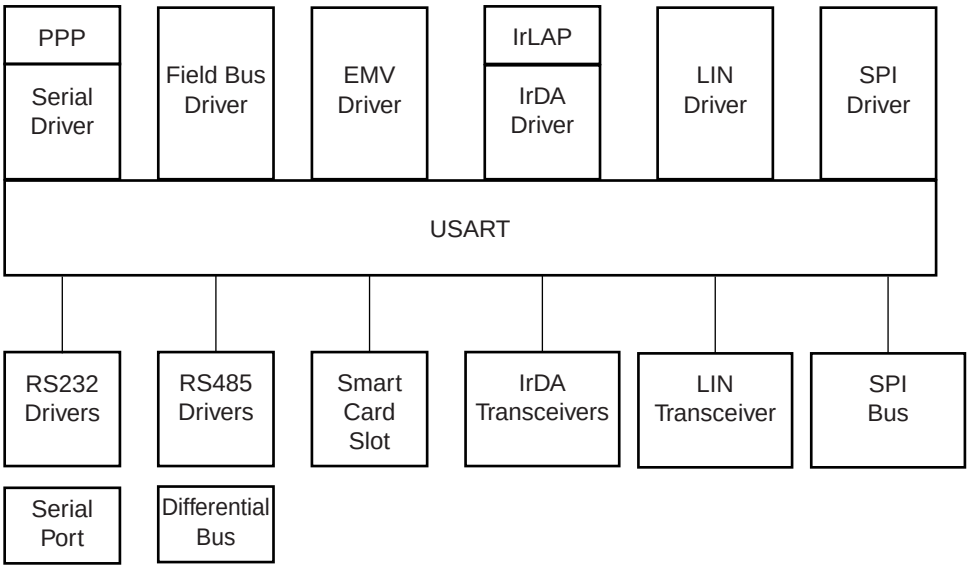


Table 35-1. SPI Operating Mode

PIN	USART	SPI Slave	SPI Master
RXD	RXD	MOSI	MISO
TXD	TXD	MISO	MOSI
RTS	RTS	–	CS
CTS	CTS	CS	–

35.4 Application Block Diagram

Figure 35-2. Application Block Diagram



35.5 I/O Lines Description

Table 35-2. 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	
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

35.6 Product Dependencies

35.6.1 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 is used, the internal pull up on TXD must also be enabled.

Table 35-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
USART0	CTS0	PB26	A
USART0	RTS0	PB25	A
USART0	RXD0	PA10	A
USART0	SCK0	PA17	B
USART0	TXD0	PA11	A
USART1	CTS1	PA15	A
USART1	RTS1	PA14	A
USART1	RXD1	PA12	A
USART1	SCK1	PA16	A
USART1	TXD1	PA13	A
USART2	CTS2	PB23	A
USART2	RTS2	PB22	A
USART2	RXD2	PB21	A
USART2	SCK2	PB24	A
USART2	TXD2	PB20	A
USART3	CTS3	PF4	A
USART3	RTS3	PF5	A
USART3	RXD3	PD5	B
USART3	SCK3	PE16	B
USART3	TXD3	PD4	B

35.6.2 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.

35.6.3 Interrupt

The USART interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the USART 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.

Table 35-4. Peripheral IDs

Instance	ID
USART0	17
USART1	18
USART2	19
USART3	20

35.7 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 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 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
- LIN Mode (USART0 only)
  - Compliant with LIN 1.3 and LIN 2.0 specifications
  - Master or Slave
  - Processing of frames with up to 256 data bytes
  - Response Data length can be configurable or defined automatically by the Identifier
  - Self synchronization in Slave node configuration
  - Automatic processing and verification of the “Synch Break” and the “Synch Field”
  - The “Synch Break” is detected even if it is partially superimposed with a data byte
  - Automatic Identifier parity calculation/sending and verification
  - Parity sending and verification can be disabled
  - Automatic Checksum calculation/sending and verification
  - Checksum sending and verification can be disabled
  - Support both “Classic” and “Enhanced” checksum types
  - Full LIN error checking and reporting
  - Frame Slot Mode: the Master allocates slots to the scheduled frames automatically.
  - Generation of the Wakeup signal
- Test modes

– Remote loopback, local loopback, automatic echo

35.7.1 Baud Rate Generator

The Baud Rate Generator provides the bit period clock named the Baud Rate Clock to both the receiver and the transmitter.

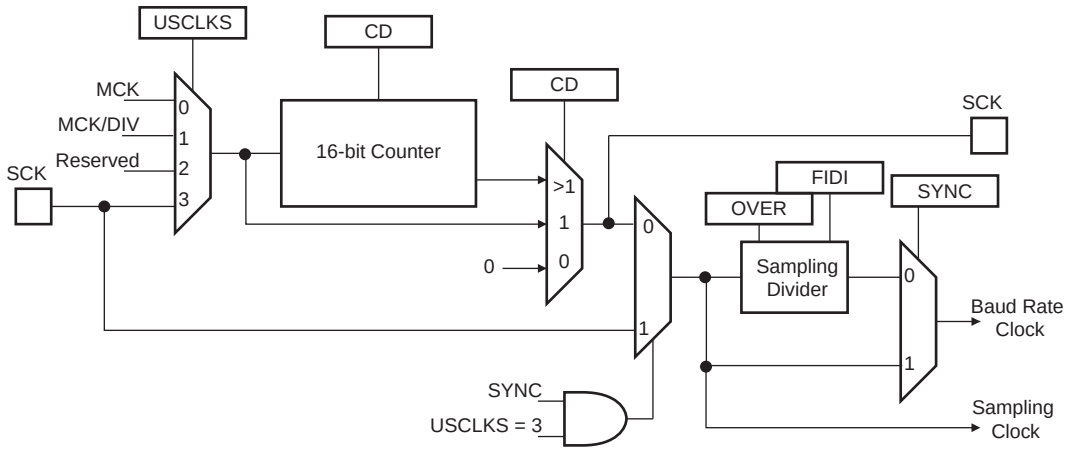
The Baud Rate Generator clock source can be selected by setting the USCLKS field in the Mode Register (US_MR) between:

- 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 in SPI mode.

Figure 35-3. Baud Rate Generator



35.7.1.1 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.

$$Baudrate = \frac{SelectedClock}{(8(2-Over)CD)}$$

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 35-5 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 35-5. 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%
60 000 000	38 400	97.66	98	38 265.31	0.35%
70 000 000	38 400	113.93	114	38 377.19	0.06%

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%.

$$Error = 1 - \left(\frac{ExpectedBaudRate}{ActualBaudRate} \right)$$

35.7.1.2 Fractional Baud Rate in Asynchronous Mode

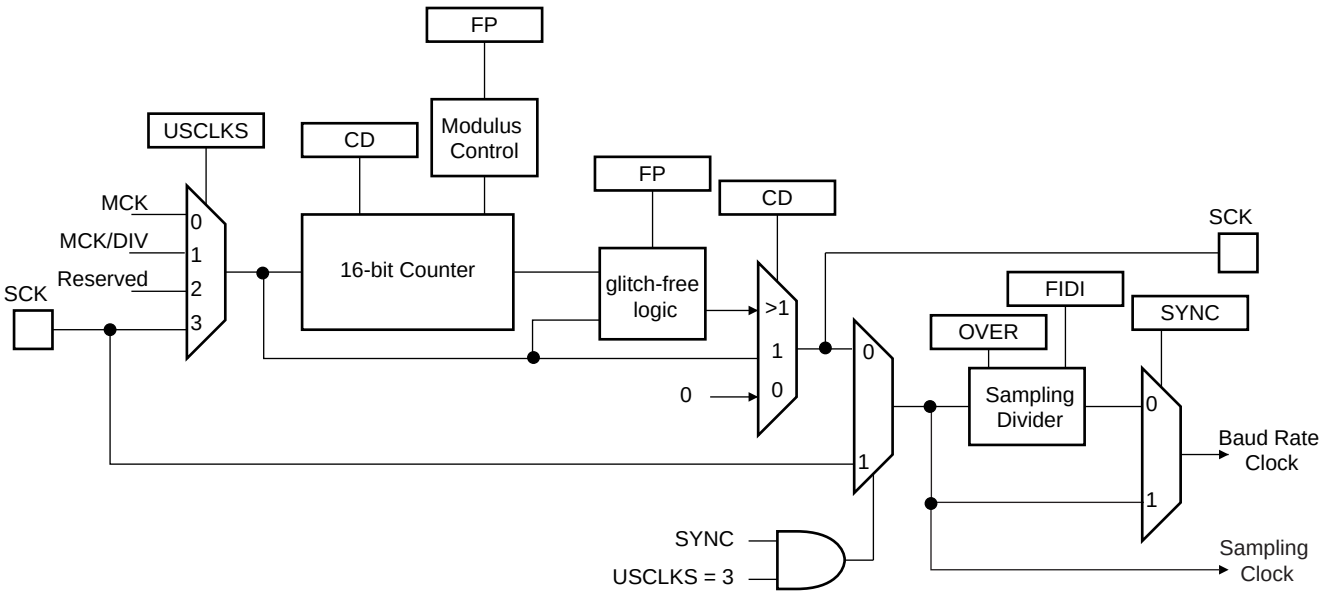
The Baud Rate generator previously defined is subject to the following limitation: the output frequency changes by only integer multiples of the reference frequency. An approach to this problem is to integrate a fractional N clock generator that has a high resolution. The generator architecture is modified 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 35-4. Fractional Baud Rate Generator



35.7.1.3 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.

$$BaudRate = \frac{SelectedClock}{CD}$$

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.

35.7.1.4 Baud Rate in ISO 7816 Mode

The ISO7816 specification defines the bit rate with the following formula:

$$B = \frac{Di}{Fi} \times f$$

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 35-6.

Table 35-6. 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

Fi is a binary value encoded on a 4-bit field, named FI, as represented in Table 35-7.

Table 35-7. Binary and Decimal Values for Fi

FI field	0000	0001	0010	0011	0100	0101	0110	1001	1010	1011	1100	1101
Fi (decimal)	372	372	558	744	1116	1488	1860	512	768	1024	1536	2048

Table 35-8 shows the resulting Fi/Di Ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

Table 35-8. Possible Values for the Fi/Di Ratio

Fi/Di	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

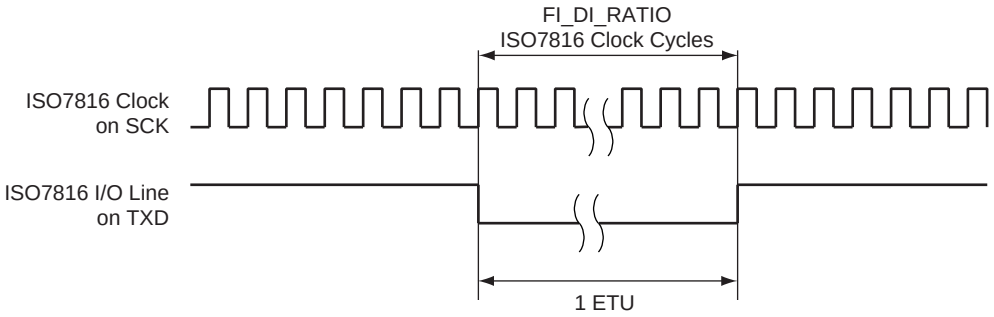
If the USART is configured in ISO7816 Mode, the clock selected by the USCLKS field in the Mode Register (US_MR) is first divided by the value programmed in the field CD in the Baud Rate Generator Register (US_BRGR). The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that the CLKO bit can be set in US_MR.

This clock is then divided by the value programmed in the FI_DI_RATIO field in the FI_DI_Ratio register (US_FIDI). This is performed by the Sampling Divider, which performs a division by up to 2047 in ISO7816 Mode. The non-integer values of the Fi/Di Ratio are not supported and the user must program the FI_DI_RATIO field to a value as close as possible to the expected value.

The FI_DI_RATIO field resets to the value 0x174 (372 in decimal) and is the most common divider between the ISO7816 clock and the bit rate (Fi = 372, Di = 1).

Figure 35-5 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

Figure 35-5. Elementary Time Unit (ETU)



35.7.2 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.

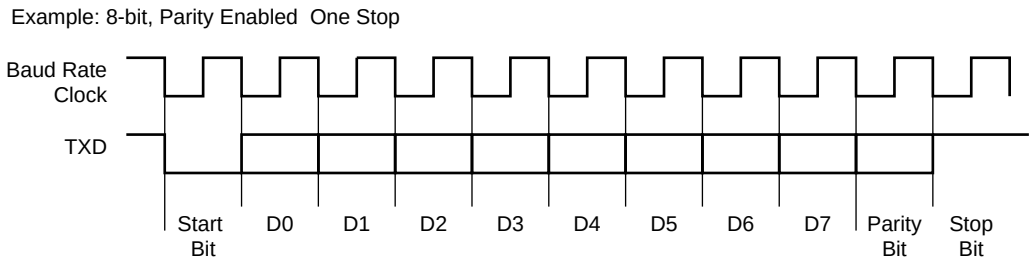
35.7.3 Synchronous and Asynchronous Modes

35.7.3.1 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.

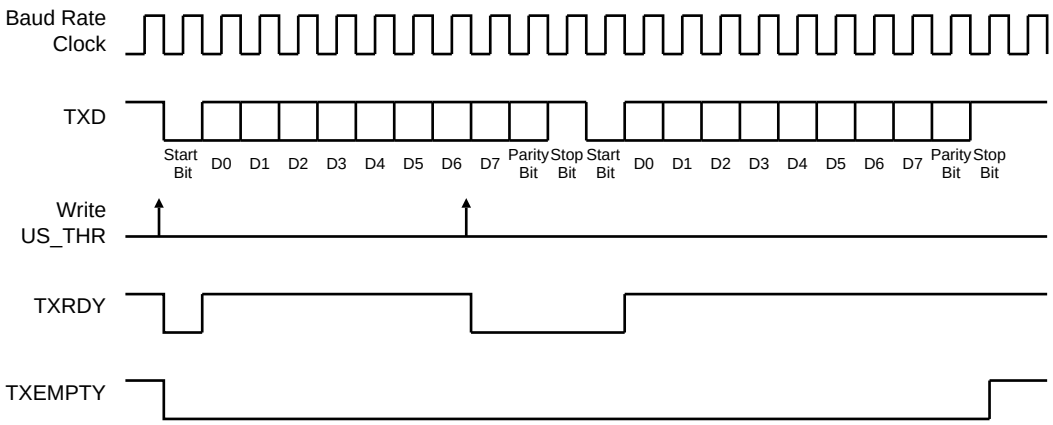
Figure 35-6. Character Transmit



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.

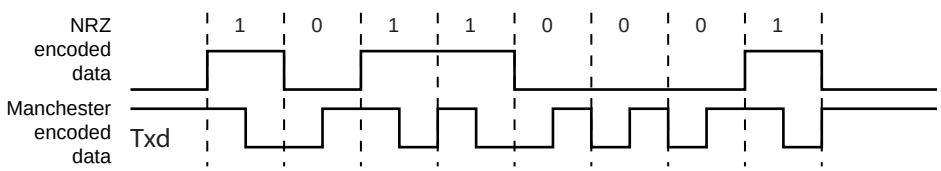
Figure 35-7. Transmitter Status



35.7.3.2 Manchester Encoder

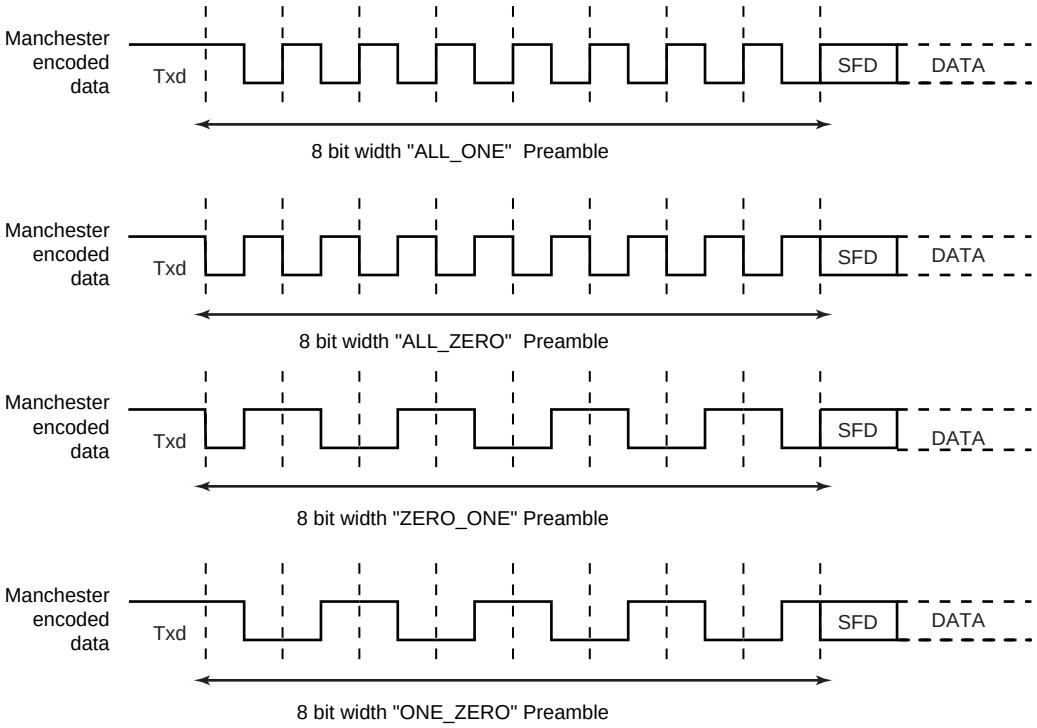
When the Manchester encoder is in use, characters transmitted through the USART are encoded based on biphase Manchester II format. To enable this mode, set the MAN field in the US_MR register to 1. Depending on polarity configuration, a logic level (zero or one), is transmitted as a coded signal one-to-zero 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 35-8 illustrates this coding scheme.

Figure 35-8. NRZ to Manchester Encoding



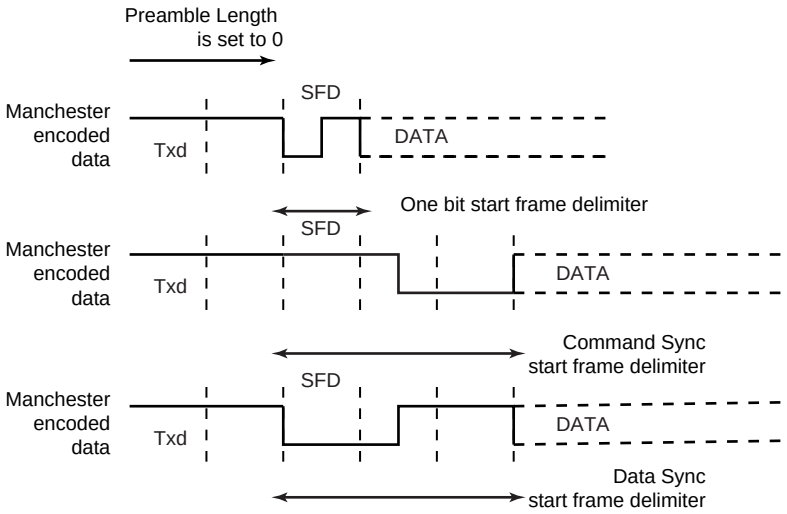
The Manchester encoded character can also be encapsulated by adding both a configurable preamble and a start frame delimiter pattern. Depending on the configuration, the preamble is a training sequence, composed of a pre-defined pattern with a programmable length from 1 to 15 bit times. If the preamble 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 35-9 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 35-9. 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 35-10 illustrates these patterns. If the start frame delimiter, also known as 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.

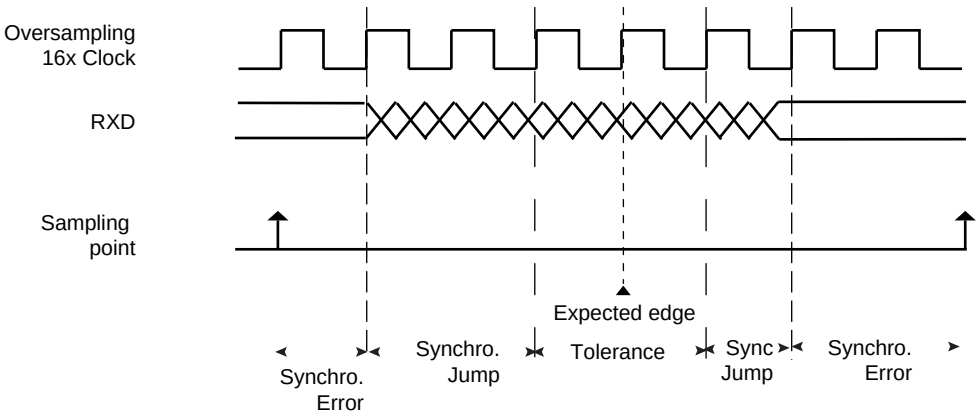
Figure 35-10. Start Frame Delimiter



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 is 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 35-11. Bit Resynchronization



35.7.3.3 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 35-12 and Figure 35-13 illustrate start detection and character reception when USART operates in asynchronous mode.

Figure 35-12. Asynchronous Start Detection

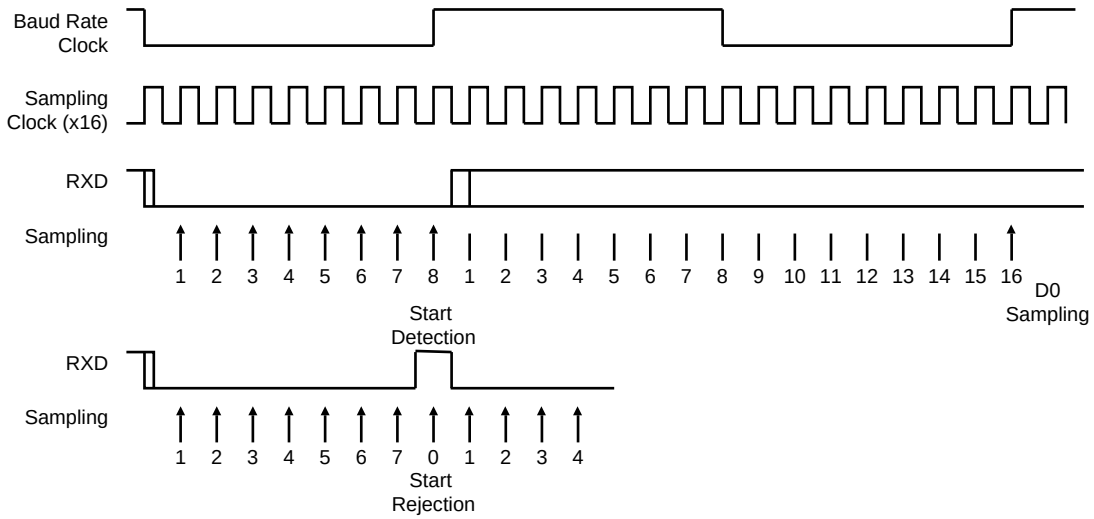
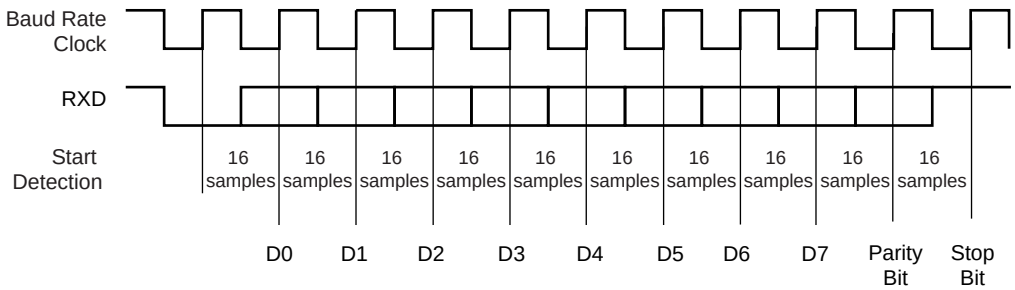


Figure 35-13. Asynchronous Character Reception

Example: 8-bit, Parity Enabled



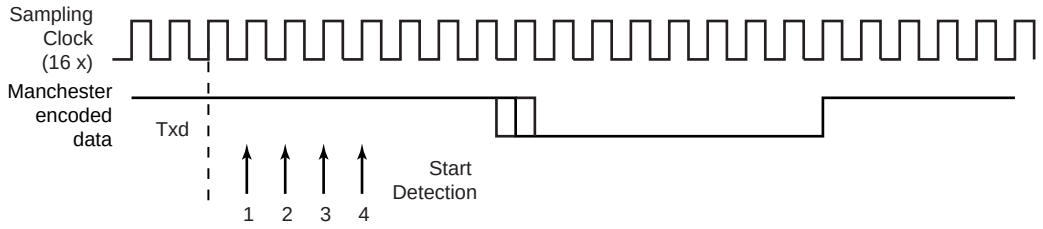
35.7.3.4 Manchester Decoder

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 35-9 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 35-14. The sample pulse rejection mechanism applies.

Figure 35-14. 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 re-synchronizes 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 35-15 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 35-16 for an example of Manchester error detection during data phase.

Figure 35-15. Preamble Pattern Mismatch

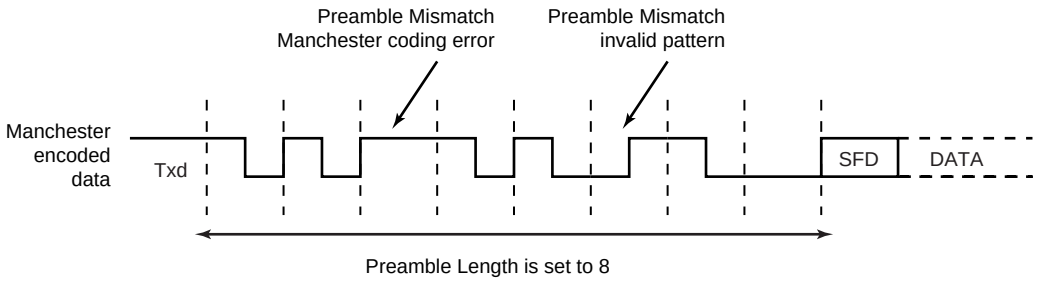
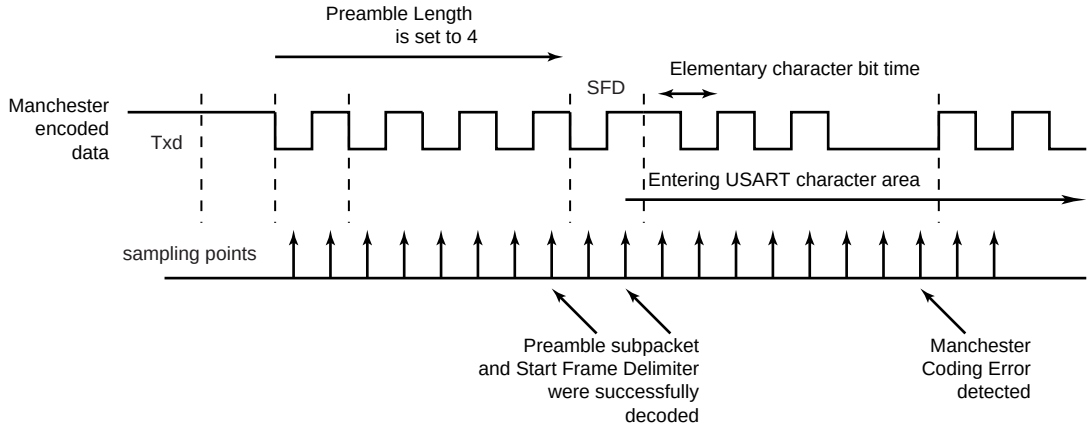


Figure 35-16. Manchester Error Flag



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.

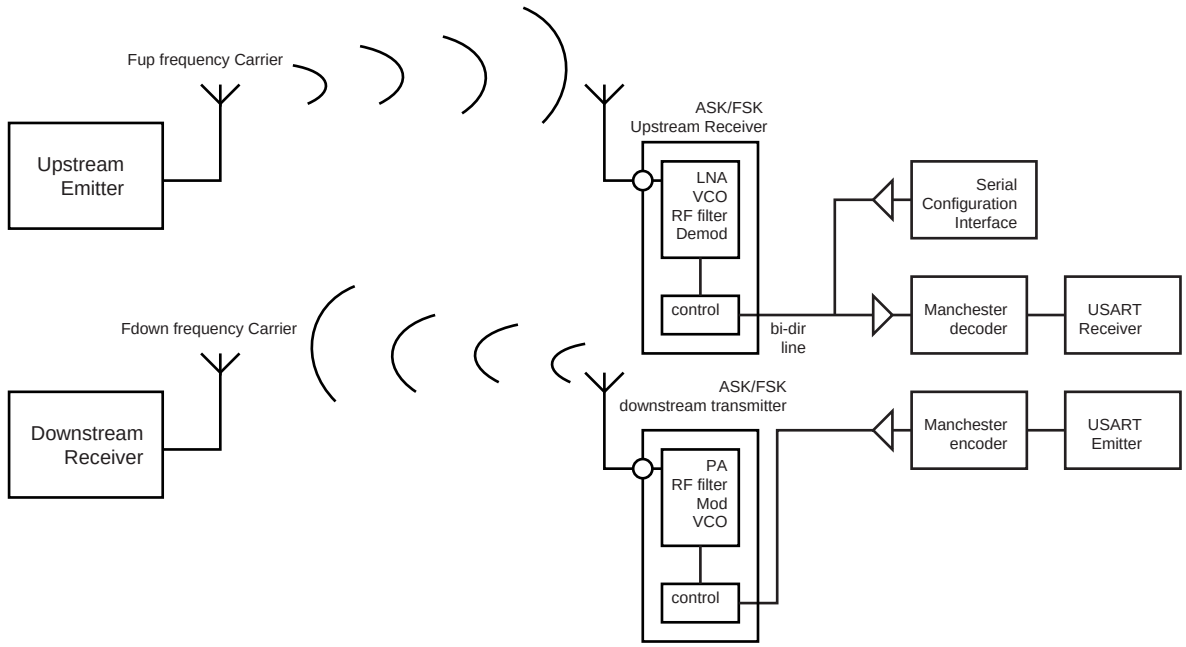
As the decoder is setup to be used in unipolar mode, the first bit of the frame has to be a zero-to-one transition.

35.7.3.5 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 35-17.

Figure 35-17. 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 35-18 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 F0 and switches to F1 if the data sent is a 0. See Figure 35-19.

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 35-18. ASK Modulator Output

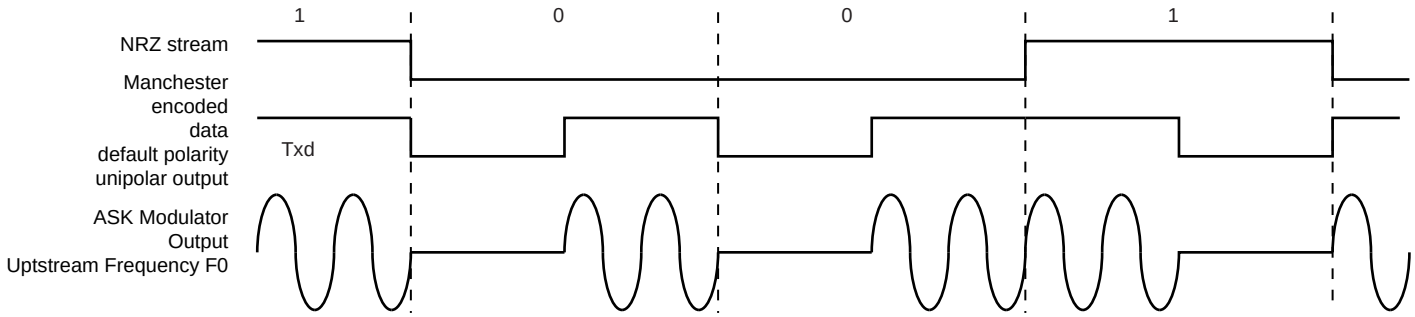
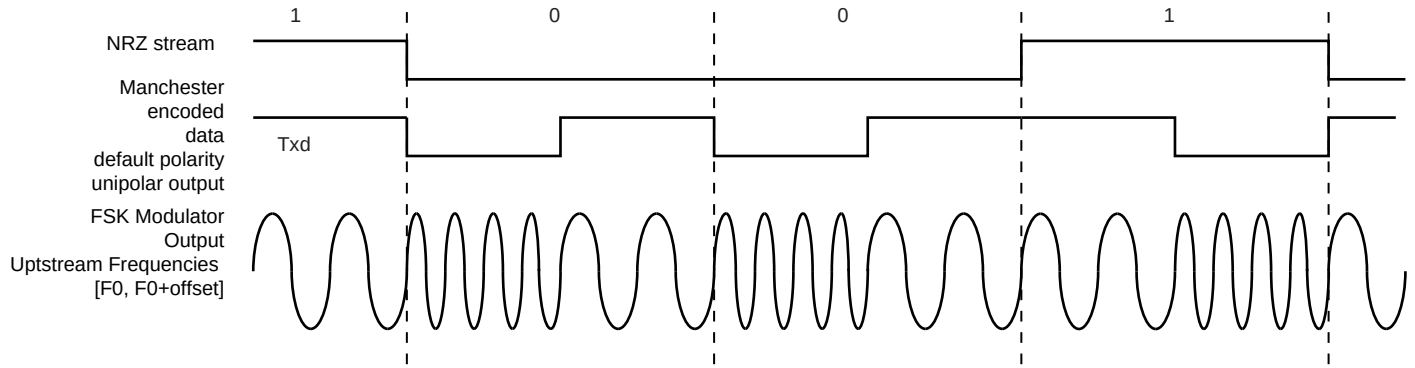


Figure 35-19. FSK Modulator Output



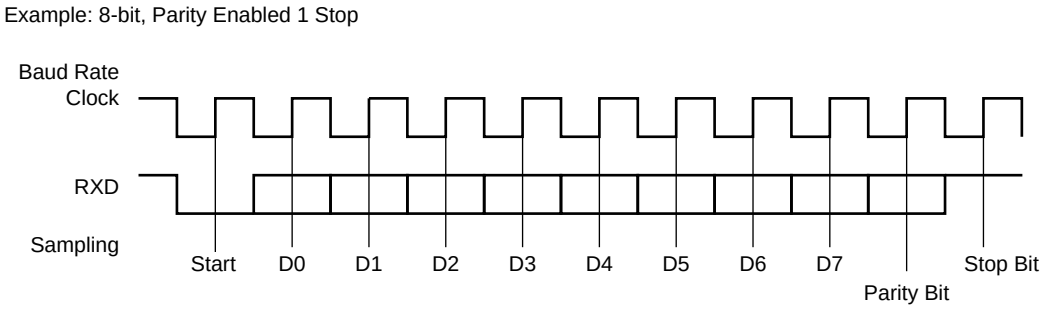
35.7.3.6 Synchronous Receiver

In synchronous mode (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 35-20 illustrates a character reception in synchronous mode.

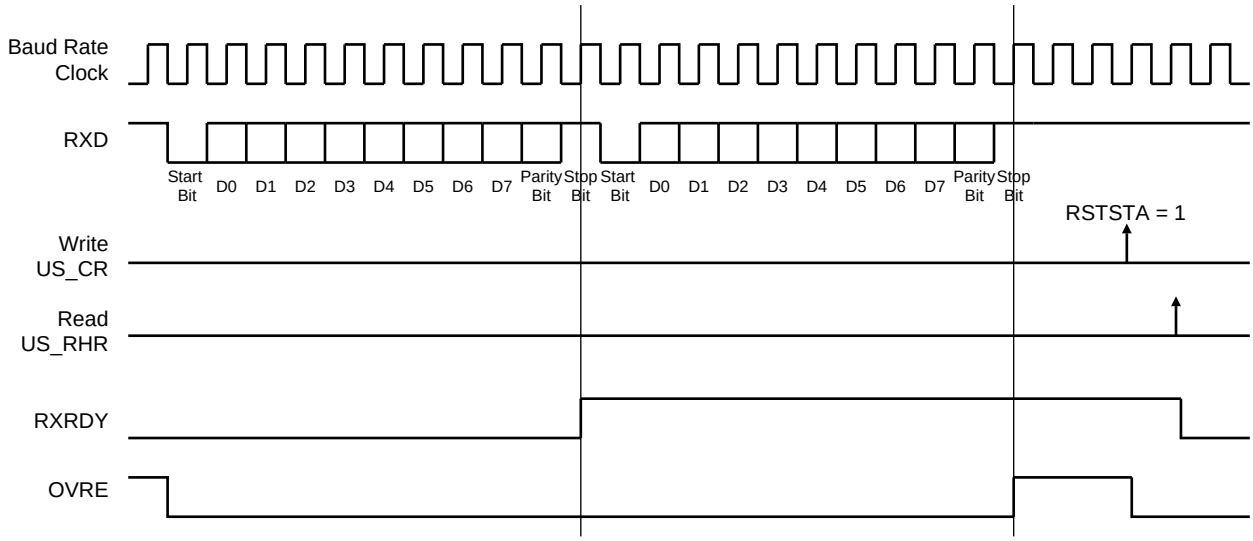
Figure 35-20. Synchronous Mode Character Reception



35.7.3.7 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 35-21. Receiver Status



35.7.3.8 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 790. 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.

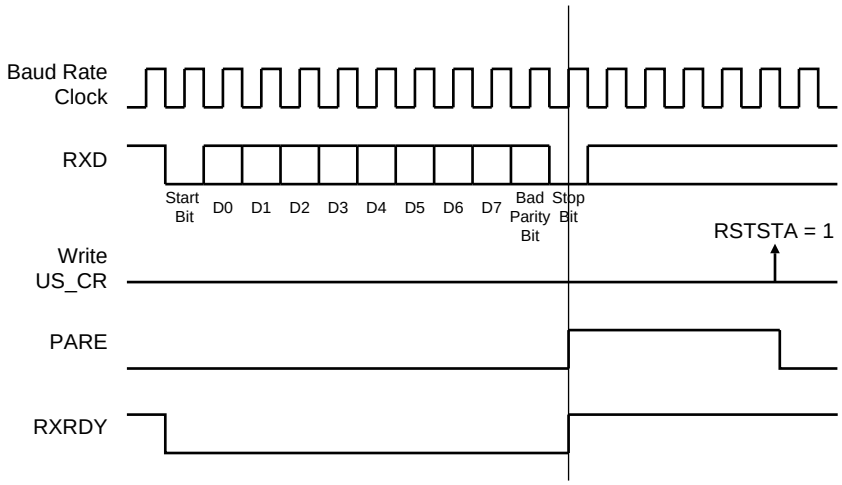
Table 35-9 shows an example of the parity bit for the character 0x41 (character ASCII “A”) depending on the configuration of the USART. Because there are two bits to 1, 1 bit is added when a parity is odd, or 0 is added when a parity is even.

Table 35-9. 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 35-22 illustrates the parity bit status setting and clearing.

Figure 35-22. Parity Error



35.7.3.9 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.

35.7.3.10 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 35-23, 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 35-23. Timeguard Operations

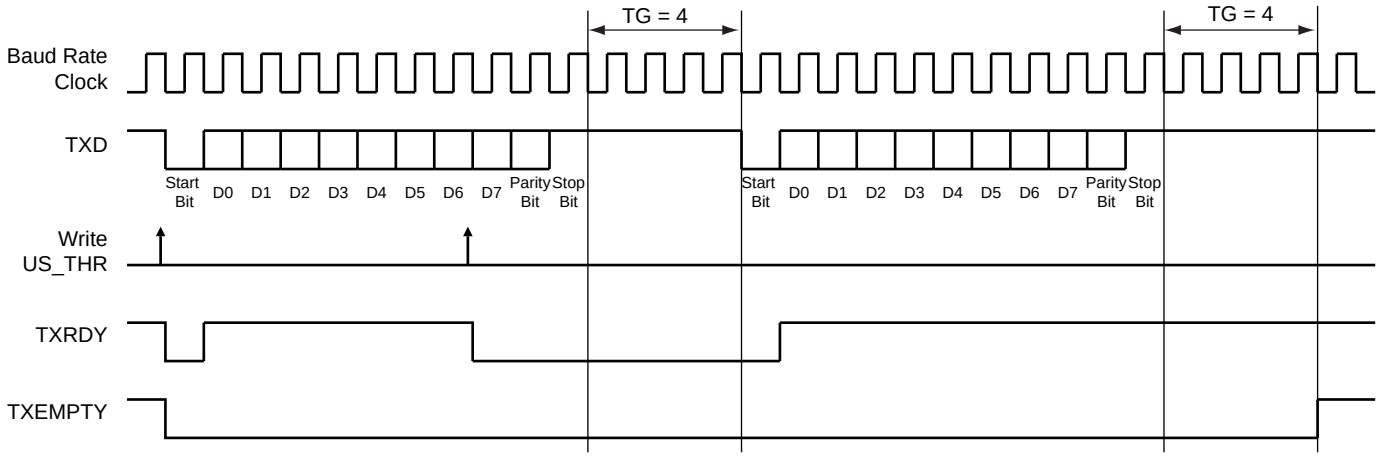


Table 35-10 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the Baud Rate.

Table 35-10. 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
14400	69.4	17.71
19200	52.1	13.28
28800	34.7	8.85
33400	29.9	7.63
56000	17.9	4.55
57600	17.4	4.43
115200	8.7	2.21

35.7.3.11 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 35-24 shows the block diagram of the Receiver Time-out feature.

Figure 35-24. Receiver Time-out Block Diagram

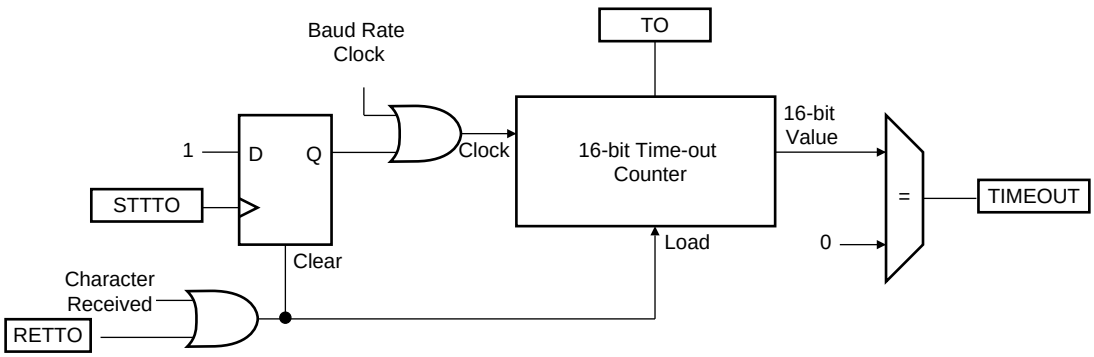


Table 35-11 gives the maximum time-out period for some standard baud rates.

Table 35-11. 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

Table 35-11. Maximum Time-out Period (Continued)

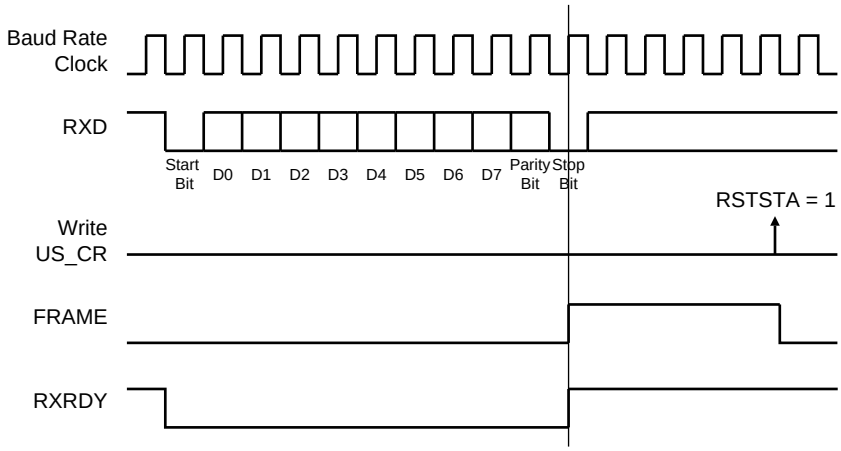
Baud Rate	Bit Time	Time-out
14400	69	4 551
19200	52	3 413
28800	35	2 276
33400	30	1 962
56000	18	1 170
57600	17	1 138
200000	5	328

35.7.3.12 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 35-25. Framing Error Status



35.7.3.13 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.

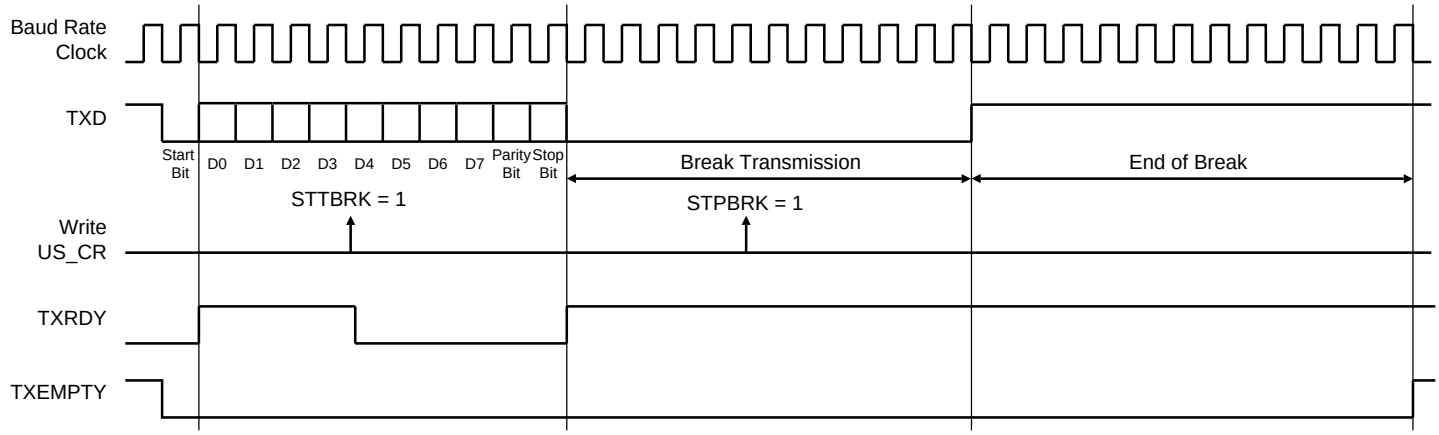
Writing US_CR with both STTBRK and STPBRK bits to 1 can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into the Transmit Holding Register while a break is pending, but not started, is ignored.

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 35-26 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

Figure 35-26. Break Transmission



35.7.3.14 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.

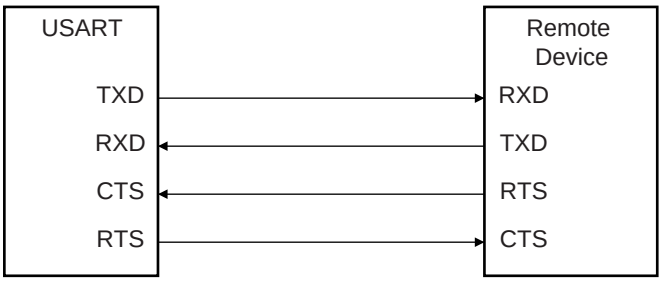
When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. This bit may be cleared by writing the Control Register (US_CR) with the bit RSTSTA to 1.

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.

35.7.3.15 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 35-27.

Figure 35-27. 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 channel for reception. The transmitter can handle hardware handshaking in any case.

Figure 35-28 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 35-28. Receiver Behavior when Operating with Hardware Handshaking

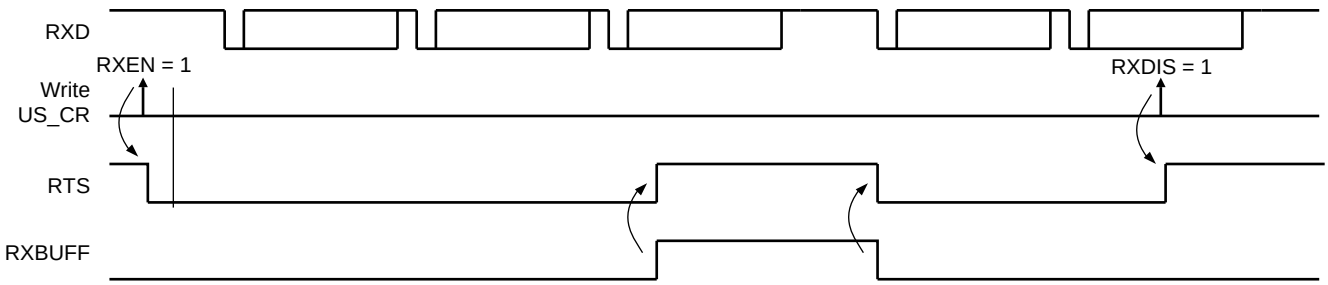
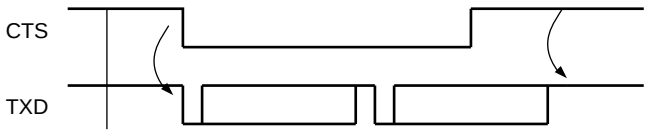


Figure 35-29 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processing, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

Figure 35-29. Transmitter Behavior when Operating with Hardware Handshaking



35.7.4 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.

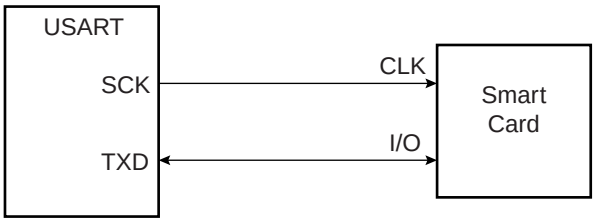
Setting the USART in ISO7816 mode is performed by writing the USART_MODE field in the Mode Register (US_MR) to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

35.7.4.1 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 777).

The USART connects to a smart card as shown in Figure 35-30. 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 35-30. 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 827 and “PAR: Parity Type” on page 828.

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. The USART does not support this format and the user has to perform an exclusive OR on the data before writing it in the Transmit Holding Register (US_THR) or after reading it in the Receive Holding Register (US_RHR).

35.7.4.2 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 35-31.

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 35-32. 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 35-31. T = 0 Protocol without Parity Error

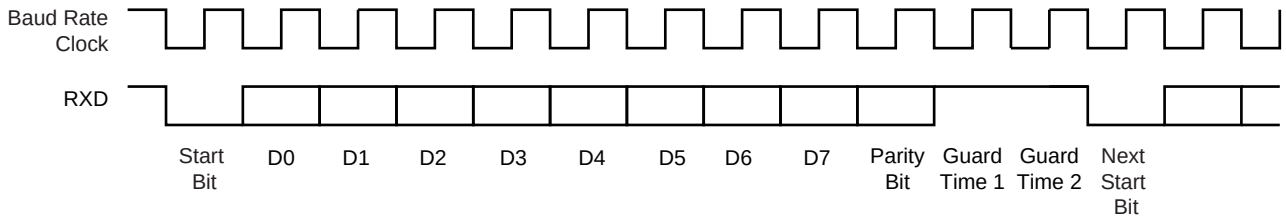
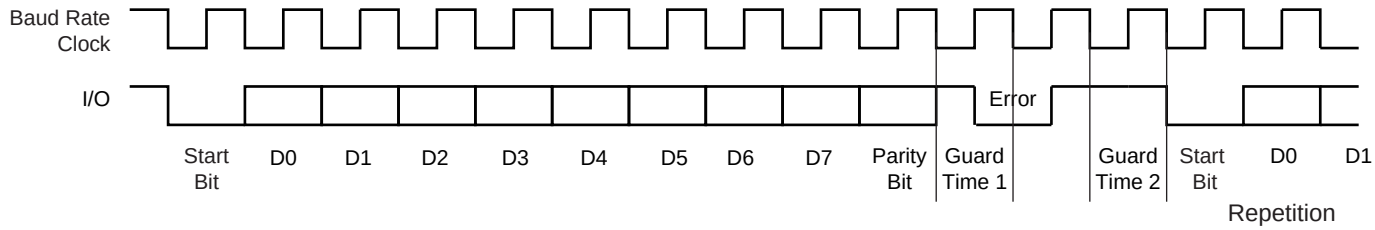


Figure 35-32. T = 0 Protocol with Parity Error



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.

35.7.4.3 Protocol T = 1

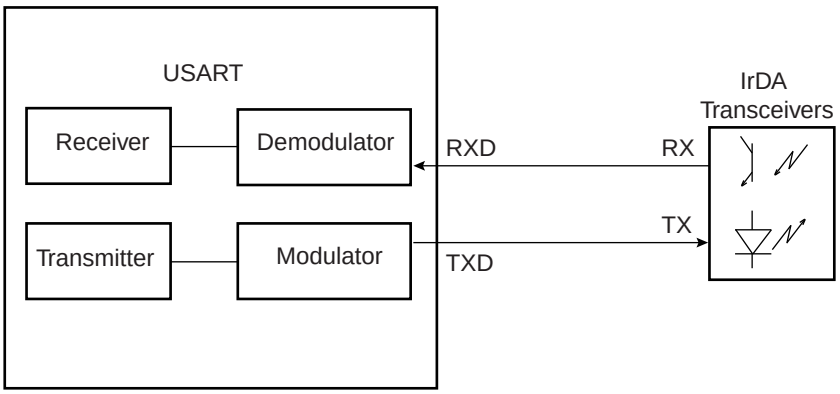
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the Channel Status Register (US_CSR).

35.7.5 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 35-33. 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 35-33. 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

35.7.5.1 IrDA Modulation

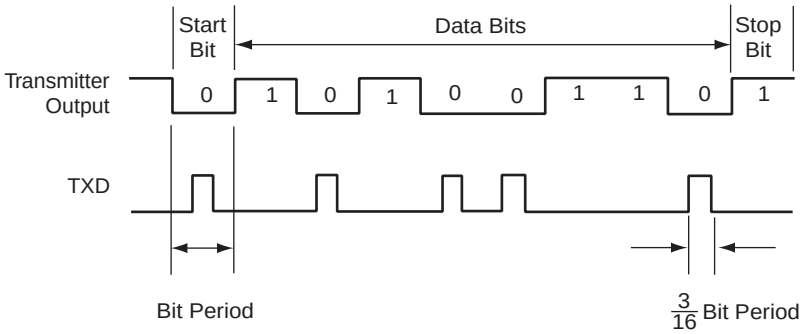
For baud rates up to and including 115.2 Kbits/sec, the RZI modulation scheme is used. “0” is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 35-12.

Table 35-12. 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 35-34 shows an example of character transmission.

Figure 35-34. IrDA Modulation



35.7.5.2 IrDA Baud Rate

Table 35-13 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 35-13. IrDA Baud Rate Error

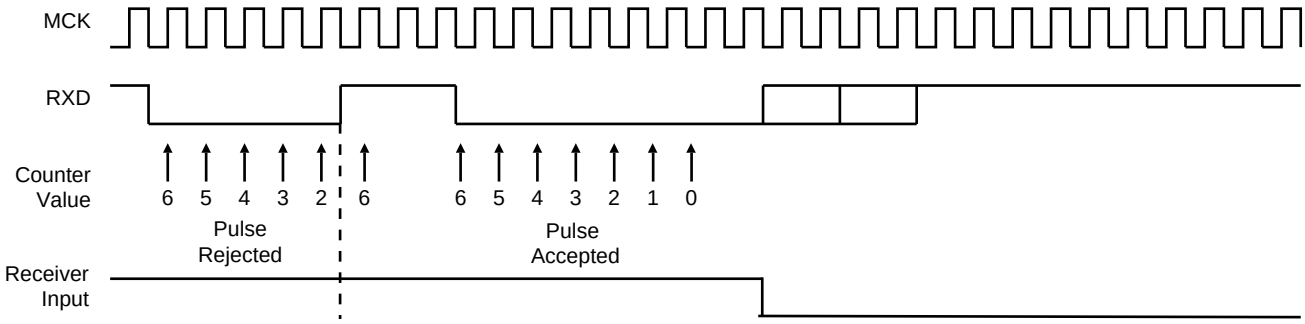
Peripheral Clock	Baud Rate	CD	Baud Rate Error	Pulse Time
3 686 400	115 200	2	0.00%	1.63
20 000 000	115 200	11	1.38%	1.63
32 768 000	115 200	18	1.25%	1.63
40 000 000	115 200	22	1.38%	1.63
3 686 400	57 600	4	0.00%	3.26
20 000 000	57 600	22	1.38%	3.26
32 768 000	57 600	36	1.25%	3.26
40 000 000	57 600	43	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
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

35.7.5.3 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 35-35 illustrates the operations of the IrDA demodulator.

Figure 35-35. IrDA Demodulator Operations

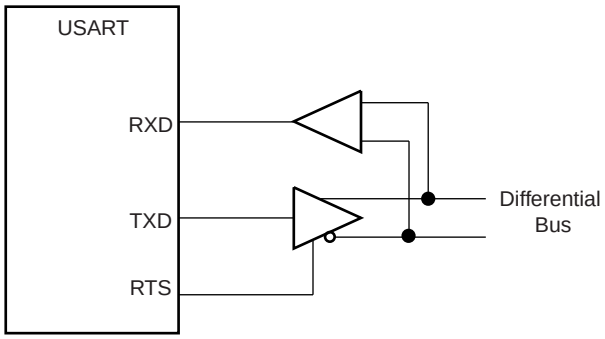


As the IrDA mode uses the same logic as the ISO7816, note that the FI_DI_RATIO field in US_FIDI must be set to a value higher than 0 in order to assure IrDA communications operate correctly.

35.7.6 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 35-36.

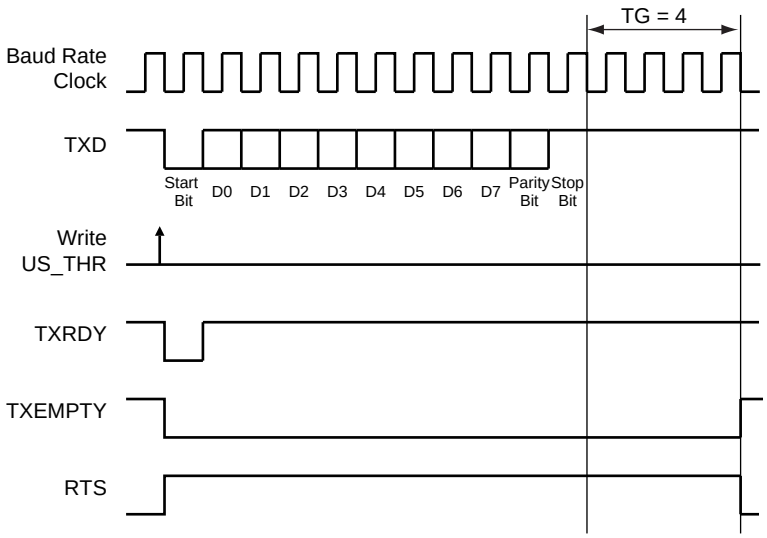
Figure 35-36. Typical Connection to a RS485 Bus



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 35-37 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

Figure 35-37. Example of RTS Drive with Timeguard



35.7.7 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.

35.7.7.1 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 35.7.8.3).

35.7.7.2 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 779. 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.

35.7.7.3 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 35-14. SPI Bus Protocol Mode

SPI Bus Protocol Mode	CPOL	CPHA
0	0	1
1	0	0
2	1	1
3	1	0

Figure 35-38. SPI Transfer Format (CPHA=1, 8 bits per transfer)

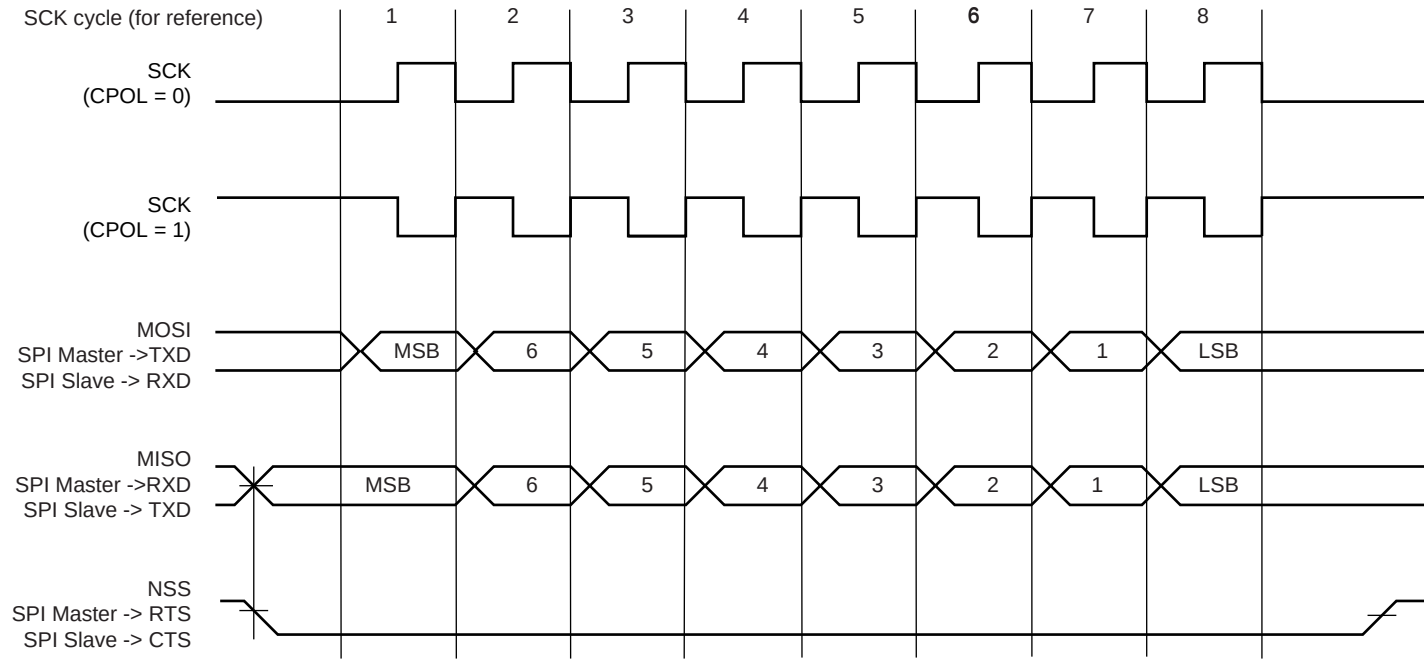
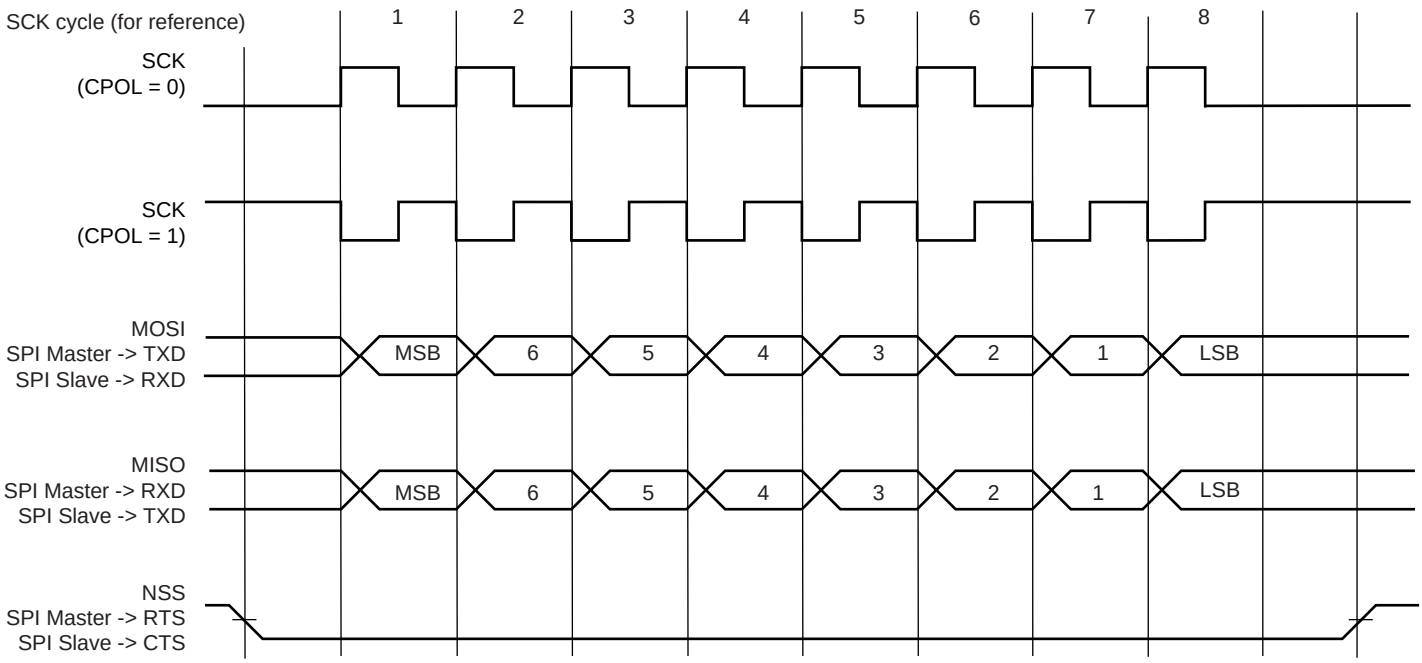


Figure 35-39. SPI Transfer Format (CPHA=0, 8 bits per transfer)



35.7.7.4 Receiver and Transmitter Control

See “Receiver and Transmitter Control” on page 781.

35.7.7.5 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 INACK equals 0, the character is transmitted whatever the receiver status. If INACK 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.

35.7.7.6 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.

35.7.7.7 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).

35.7.8 LIN Mode

The LIN Mode provides Master node and Slave node connectivity on a LIN bus.

The LIN (Local Interconnect Network) is a serial communication protocol which efficiently supports the control of mechatronic nodes in distributed automotive applications.

The main properties of the LIN bus are:

- Single Master/Multiple Slaves concept
- Low cost silicon implementation based on common UART/SCI interface hardware, an equivalent in software, or as a pure state machine.
- Self synchronization without quartz or ceramic resonator in the slave nodes
- Deterministic signal transmission
- Low cost single-wire implementation
- Speed up to 20 kbit/s

LIN provides cost efficient bus communication where the bandwidth and versatility of CAN are not required.

The LIN Mode enables processing LIN frames with a minimum of action from the microprocessor.

35.7.8.1 Modes of Operation

The USART can act either as a LIN Master node or as a LIN Slave node.

The node configuration is chosen by setting the USART_MODE field in the USART Mode register (US_MR):

- LIN Master Node (USART_MODE=0xA)
- LIN Slave Node (USART_MODE=0xB)

In order to avoid unpredicted behavior, any change of the LIN node configuration must be followed by a software reset of the transmitter and of the receiver (except the initial node configuration after a hardware reset). (See Section 35.7.8.3)

35.7.8.2 Baud Rate Configuration

See “Baud Rate in Asynchronous Mode” on page 777.

The baud rate is configured in the Baud Rate Generator register (US_BRGR).

35.7.8.3 Receiver and Transmitter Control

See “Receiver and Transmitter Control” on page 781.

35.7.8.4 Character Transmission

See “Transmitter Operations” on page 781.

35.7.8.5 Character Reception

See “Receiver Operations” on page 789.

35.7.8.6 Header Transmission (Master Node Configuration)

All the LIN Frames start with a header which is sent by the master node and consists of a Synch Break Field, Synch Field and Identifier Field.

So in Master node configuration, the frame handling starts with the sending of the header.

The header is transmitted as soon as the identifier is written in the LIN Identifier register (US_LINIR). At this moment the flag TXRDY falls.

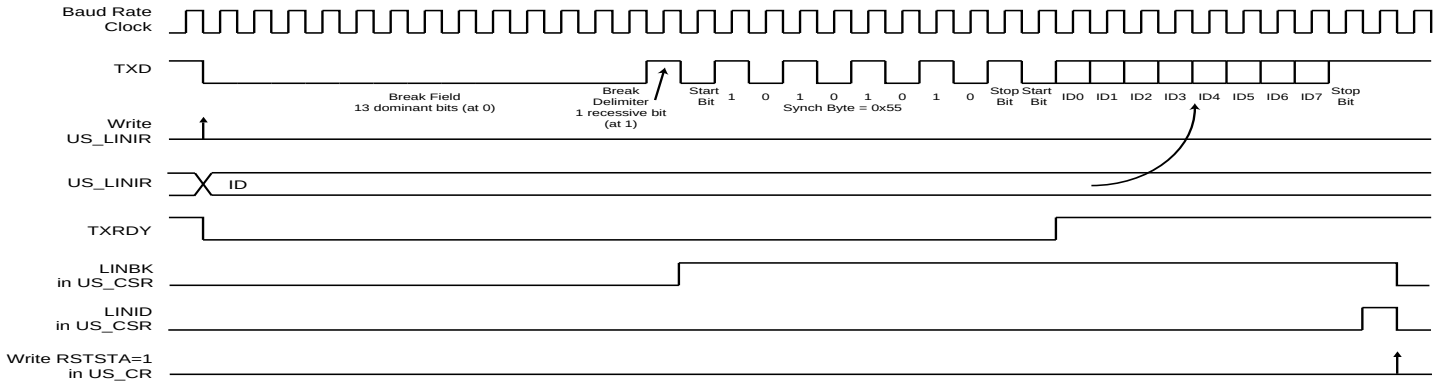
The Break Field, the Synch Field and the Identifier Field are sent automatically one after the other.

The Break Field consists of 13 dominant bits and 1 recessive bit, the Synch Field is the character 0x55 and the Identifier corresponds to the character written in the LIN Identifier Register (US_LINIR). The Identifier parity bits can be automatically computed and sent (see Section 35.7.8.9).

The flag TXRDY rises when the identifier character is transferred into the Shift Register of the transmitter.

As soon as the Synch Break Field is transmitted, the flag LINBK in the Channel Status register (US_CSR) is set to 1. Likewise, as soon as the Identifier Field is sent, the flag LINID in the Channel Status register (US_CSR) is set to 1. These flags are reset by writing the bit RSTSTA to 1 in the Control register (US_CR).

Figure 35-40. Header Transmission



35.7.8.7 Header Reception (Slave Node Configuration)

All the LIN Frames start with a header which is sent by the master node and consists of a Synch Break Field, Synch Field and Identifier Field.

In Slave node configuration, the frame handling starts with the reception of the header.

The USART uses a break detection threshold of 11 nominal bit times at the actual baud rate. At any time, if 11 consecutive recessive bits are detected on the bus, the USART detects a Break Field. As long as a Break Field has not been detected, the USART stays idle and the received data are not taken in account.

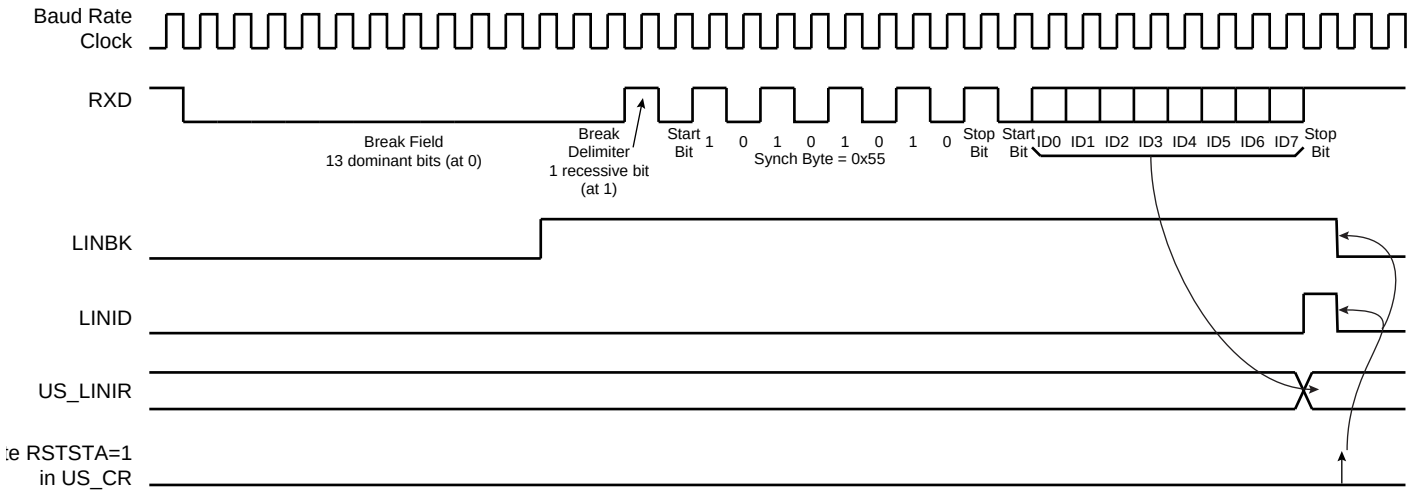
When a Break Field has been detected, the flag LINBK in the Channel Status register (US_CSR) is set to 1 and the USART expects the Synch Field character to be 0x55. This field is used to update the actual baud rate in order to stay synchronized (see Section 35.7.8.8). If the received Synch character is not 0x55, an Inconsistent Synch Field error is generated (see Section 35.7.8.14).

After receiving the Synch Field, the USART expects to receive the Identifier Field.

When the Identifier Field has been received, the flag LINID in the Channel Status register (US_CSR) is set to 1. At this moment the field IDCHR in the LIN Identifier register (US_LINIR) is updated with the received character. The Identifier parity bits can be automatically computed and checked (see Section 35.7.8.9).

The flags LINID and LINBK are reset by writing the bit RSTSTA to 1 in the Control register (US_CR).

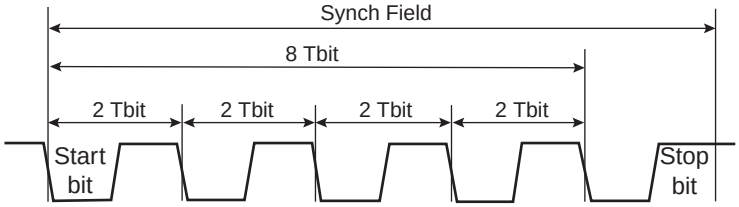
Figure 35-41. Header Reception



35.7.8.8 Slave Node Synchronization

The synchronization is done only in Slave node configuration. The procedure is based on time measurement between falling edges of the Synch Field. The falling edges are available in distances of 2, 4, 6 and 8 bit times.

Figure 35-42. Synch Field



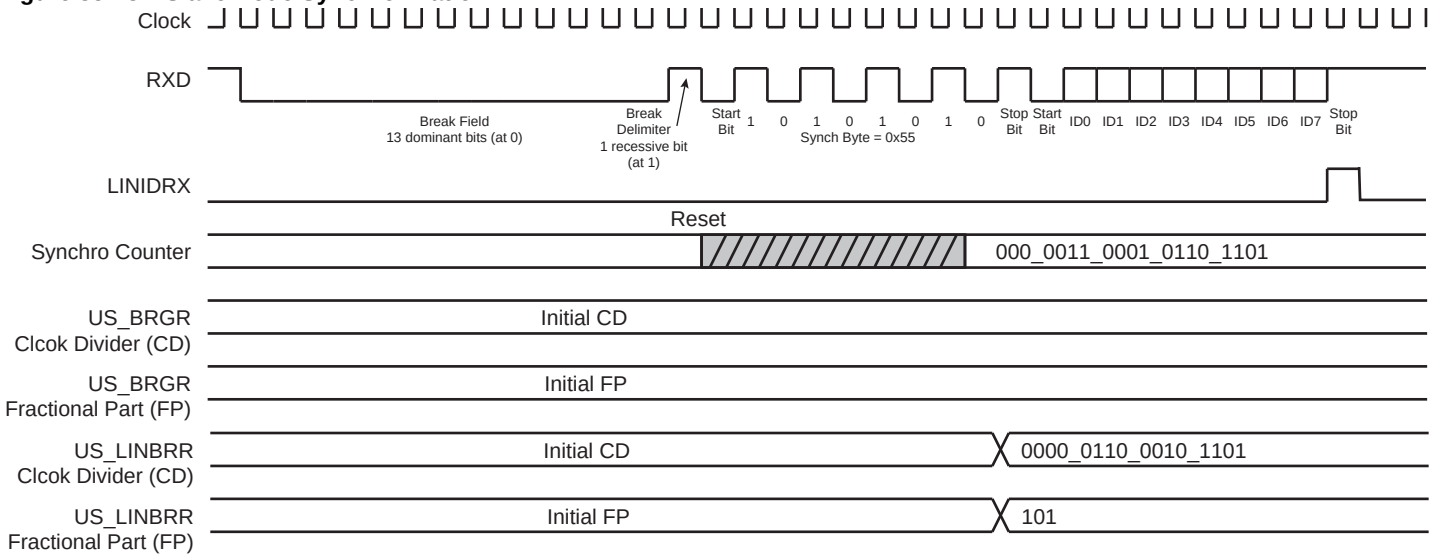
The time measurement is made by a 19-bit counter clocked by the sampling clock (see Section 35.7.1).

When the start bit of the Synch Field is detected, the counter is reset. Then during the next 8 Tbits of the Synch Field, the counter is incremented. At the end of these 8 Tbits, the counter is stopped. At this moment, the 16 most significant bits of the counter (value divided by 8) give the new clock divider (LINCD) and the 3 least significant bits of this value (the remainder) give the new fractional part (LINFP).

When the Synch Field has been received, the clock divider (CD) and the fractional part (FP) are updated in the Baud Rate Generator register (US_BRGR).

If it appears that the sampled Synch character is not equal to 0x55, then the error flag LINISFE in the Channel Status register (US_CSR) is set to 1. It is reset by writing bit RSTSTA to 1 in the Control register (US_CR).

Figure 35-43. Slave Node Synchronization



The accuracy of the synchronization depends on several parameters:

- The nominal clock frequency (F_{Nom}) (the theoretical slave node clock frequency)
- The Baud Rate
- The oversampling (Over=0 => 16X or Over=0 => 8X)

The following formula is used to compute the deviation of the slave bit rate relative to the master bit rate after synchronization (F_{SLAVE} is the real slave node clock frequency).

$$\text{Baudrate_deviation} = \left(100 \times \frac{[\alpha \times 8 \times (2 - \text{Over}) + \beta] \times \text{Baudrate}}{8 \times F_{SLAVE}}\right)\%$$

$$\text{Baudrate_deviation} = \left(100 \times \frac{[\alpha \times 8 \times (2 - \text{Over}) + \beta] \times \text{Baudrate}}{8 \times \left(\frac{F_{TOL_UNSYNCH}}{100}\right) \text{x} F_{Nom}}\right)\%$$

$$-0.5 \le \alpha \le +0.5 \quad -1 < \beta < +1$$

$F_{TOL_UNSYNCH}$ is the deviation of the real slave node clock from the nominal clock frequency. The LIN Standard imposes that it must not exceed ±15%. The LIN Standard imposes also that for communication between two nodes, their bit rate must not differ by more than ±2%. This means that the Baudrate_deviation must not exceed ±1%.

It follows from that, a minimum value for the nominal clock frequency:

$$F_{NOM}(\text{min}) = \left(100 \times \frac{[0.5 \times 8 \times (2 - \text{Over}) + 1] \times \text{Baudrate}}{8 \times \left(\frac{-15}{100} + 1\right) \times 1\%}\right)\text{Hz}$$

Examples:

- Baudrate = 20 kbit/s, Over=0 (Oversampling 16X) => F_{Nom}(min) = 2.64 MHz
- Baudrate = 20 kbit/s, Over=1 (Oversampling 8X) => F_{Nom}(min) = 1.47 MHz
- Baudrate = 1 kbit/s, Over=0 (Oversampling 16X) => F_{Nom}(min) = 132 kHz
- Baudrate = 1 kbit/s, Over=1 (Oversampling 8X) => F_{Nom}(min) = 74 kHz

35.7.8.9 Identifier Parity

A protected identifier consists of two sub-fields; the identifier and the identifier parity. Bits 0 to 5 are assigned to the identifier and bits 6 and 7 are assigned to the parity.

The USART interface can generate/check these parity bits, but this feature can also be disabled. The user can choose between two modes by the PARDIS bit of the LIN Mode register (US_LINMR):

- PARDIS = 0:

During header transmission, the parity bits are computed and sent with the 6 least significant bits of the IDCHR field of the LIN Identifier register (US_LINIR). The bits 6 and 7 of this register are discarded.

During header reception, the parity bits of the identifier are checked. If the parity bits are wrong, an Identifier Parity error occurs (see Section 35.7.3.8). Only the 6 least significant bits of the IDCHR field are updated with the received Identifier. The bits 6 and 7 are stuck to 0.

- PARDIS = 1:

During header transmission, all the bits of the IDCHR field of the LIN Identifier register (US_LINIR) are sent on the bus.

During header reception, all the bits of the IDCHR field are updated with the received Identifier.

35.7.8.10 Node Action

In function of the identifier, the node is concerned, or not, by the LIN response. Consequently, after sending or receiving the identifier, the USART must be configured. There are three possible configurations:

- PUBLISH: the node sends the response.
- SUBSCRIBE: the node receives the response.
- IGNORE: the node is not concerned by the response, it does not send and does not receive the response.

This configuration is made by the field, Node Action (NACT), in the US_LINMR register (see Section 35.8.16).

Example: a LIN cluster that contains a Master and two Slaves:

- Data transfer from the Master to the Slave 1 and to the Slave 2:

  NACT(Master)=PUBLISH

  NACT(Slave1)=SUBSCRIBE

  NACT(Slave2)=SUBSCRIBE
- Data transfer from the Master to the Slave 1 only:

  NACT(Master)=PUBLISH

  NACT(Slave1)=SUBSCRIBE

  NACT(Slave2)=IGNORE
- Data transfer from the Slave 1 to the Master:

  NACT(Master)=SUBSCRIBE

  NACT(Slave1)=PUBLISH

  NACT(Slave2)=IGNORE
- Data transfer from the Slave1 to the Slave2:

  NACT(Master)=IGNORE

  NACT(Slave1)=PUBLISH

  NACT(Slave2)=SUBSCRIBE
- Data transfer from the Slave2 to the Master and to the Slave1:

  NACT(Master)=SUBSCRIBE

  NACT(Slave1)=SUBSCRIBE

  NACT(Slave2)=PUBLISH

35.7.8.11 Response Data Length

The LIN response data length is the number of data fields (bytes) of the response excluding the checksum.

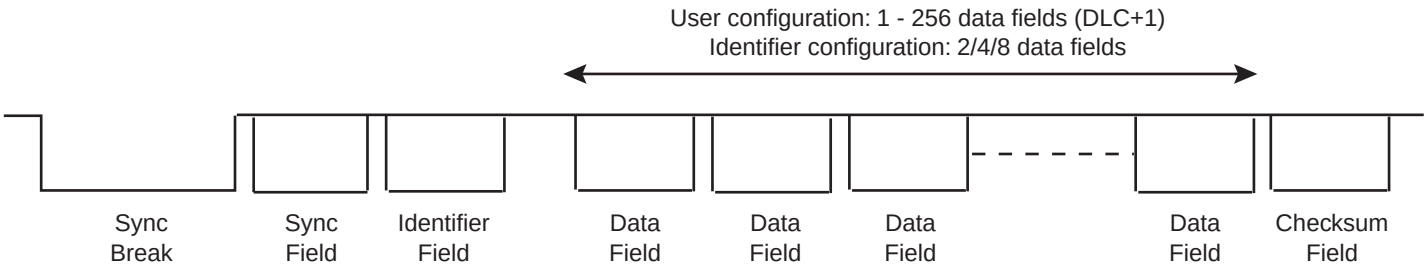
The response data length can either be configured by the user or be defined automatically by bits 4 and 5 of the Identifier (compatibility to LIN Specification 1.1). The user can choose between these two modes by the DLM bit of the LIN Mode register (US_LINMR):

- DLM = 0: the response data length is configured by the user via the DLC field of the LIN Mode register (US_LINMR). The response data length is equal to (DLC + 1) bytes. DLC can be programmed from 0 to 255, so the response can contain from 1 data byte up to 256 data bytes.
- DLM = 1: the response data length is defined by the Identifier (IDCHR in US_LINIR) according to the table below. The DLC field of the LIN Mode register (US_LINMR) is discarded. The response can contain 2 or 4 or 8 data bytes.

Table 35-15. Response Data Length if DLM = 1

IDCHR[5]	IDCHR[4]	Response Data Length [bytes]
0	0	2
0	1	2
1	0	4
1	1	8

Figure 35-44. Response Data Length



35.7.8.12 Checksum

The last field of a frame is the checksum. The checksum contains the inverted 8- bit sum with carry, over all data bytes or all data bytes and the protected identifier. Checksum calculation over the data bytes only is called classic checksum and it is used for communication with LIN 1.3 slaves. Checksum calculation over the data bytes and the protected identifier byte is called enhanced checksum and it is used for communication with LIN 2.0 slaves.

The USART can be configured to:

- Send/Check an Enhanced checksum automatically (CHKDIS = 0 & CHKTYP = 0)
- Send/Check a Classic checksum automatically (CHKDIS = 0 & CHKTYP = 1)
- Not send/check a checksum (CHKDIS = 1)

This configuration is made by the Checksum Type (CHKTYP) and Checksum Disable (CHKDIS) fields of the LIN Mode register (US_LINMR).

If the checksum feature is disabled, the user can send it manually all the same, by considering the checksum as a normal data byte and by adding 1 to the response data length (see Section 35.7.8.11).

35.7.8.13 Frame Slot Mode

This mode is useful only for Master nodes. It respects the following rule: each frame slot shall be longer than or equal to TFrame_Maximum.

If the Frame Slot Mode is enabled (FSDIS = 0) and a frame transfer has been completed, the TXRDY flag is set again only after TFrame_Maximum delay, from the start of frame. So the Master node cannot send a new header if the frame slot duration of the previous frame is inferior to TFrame_Maximum.

If the Frame Slot Mode is disabled (FSDIS = 1) and a frame transfer has been completed, the TXRDY flag is set again immediately.

The TFrame_Maximum is calculated as below:

If the Checksum is sent (CHKDIS = 0):

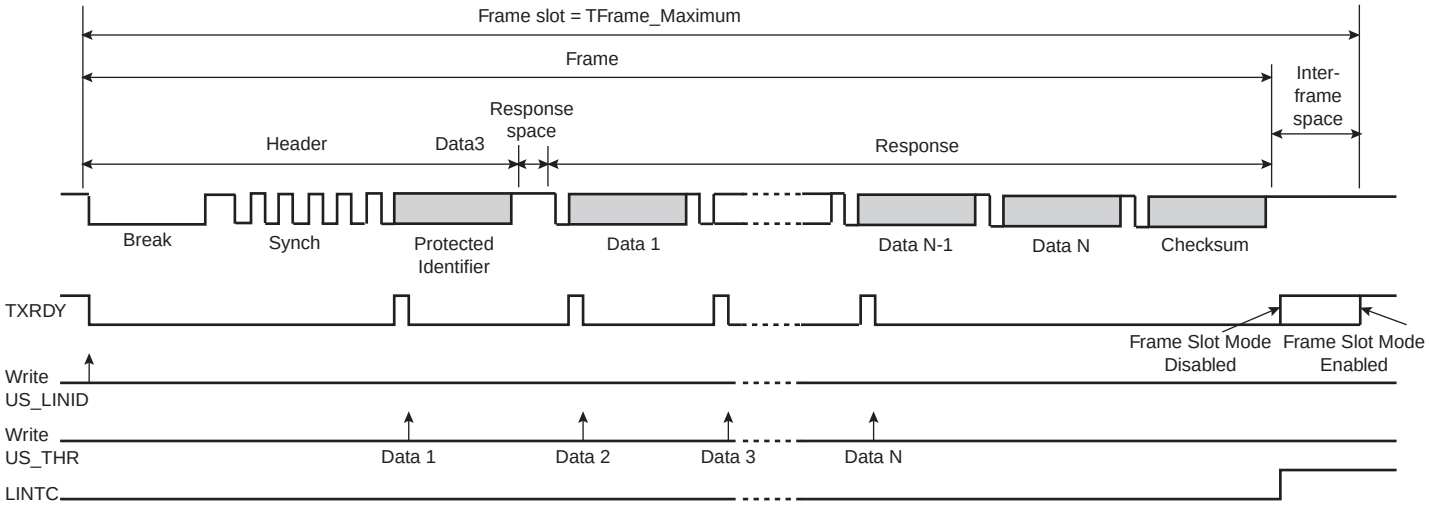
- THeader_Nominal = 34 x Tbit
- TResponse_Nominal = 10 x (NData + 1) x Tbit
- TFrame_Maximum = 1.4 x (THeader_Nominal + TResponse_Nominal + 1)(Note:)
- TFrame_Maximum = 1.4 x (34 + 10 x (DLC + 1 + 1) + 1) x Tbit
- TFrame_Maximum = (77 + 14 x DLC) x Tbit

If the Checksum is not sent (CHKDIS = 1):

- THeader_Nominal = 34 x Tbit
- TResponse_Nominal = 10 x NData x Tbit
- TFrame_Maximum = 1.4 x (THeader_Nominal + TResponse_Nominal + 1(Note:))
- TFrame_Maximum = 1.4 x (34 + 10 x (DLC + 1) + 1) x Tbit
- TFrame_Maximum = (63 + 14 x DLC) x Tbit

Note: The term “+1” leads to an integer result for TFrame_Max (LIN Specification 1.3)

Figure 35-45. Frame Slot Mode



35.7.8.14 LIN Errors

Bit Error

This error is generated in Master of Slave node configuration, when the USART is transmitting and if the transmitted value on the Tx line is different from the value sampled on the Rx line. If a bit error is detected, the transmission is aborted at the next byte border.

This error is reported by flag LINBE in the Channel Status Register (US_CSR).

Inconsistent Synch Field Error

This error is generated in Slave node configuration, if the Synch Field character received is other than 0x55.

This error is reported by flag LINISFE in the Channel Status Register (US_CSR).

Identifier Parity Error

This error is generated in Slave node configuration, if the parity of the identifier is wrong. This error can be generated only if the parity feature is enabled (PARDIS = 0).

This error is reported by flag LINIPE in the Channel Status Register (US_CSR).

Checksum Error

This error is generated in Master of Slave node configuration, if the received checksum is wrong. This flag can be set to “1” only if the checksum feature is enabled (CHKDIS = 0).

This error is reported by flag LINCE in the Channel Status Register (US_CSR).

Slave Not Responding Error

This error is generated in Master of Slave node configuration, when the USART expects a response from another node (NACT = SUBSCRIBE) but no valid message appears on the bus within the time given by the maximum length of the message frame, TFrame_Maximum (see Section 35.7.8.13). This error is disabled if the USART does not expect any message (NACT = PUBLISH or NACT = IGNORE).

This error is reported by flag LINSNRE in the Channel Status Register (US_CSR).

35.7.8.15 LIN Frame Handling

Master Node Configuration

- Write TXEN and RXEN in US_CR to enable both the transmitter and the receiver.
- Write USART_MODE in US_MR to select the LIN mode and the Master Node configuration.
- Write CD and FP in US_BRGR to configure the baud rate.
- Write NACT, PARDIS, CHKDIS, CHKTYPE, DLCM, FSDIS and DLC in US_LINMR to configure the frame transfer.
- Check that TXRDY in US_CSR is set to “1”
- Write IDCHR in US_LINIR to send the header

What comes next depends on the NACT configuration:

- Case 1: NACT = PUBLISH, the USART sends the response
  - Wait until TXRDY in US_CSR rises
  - Write TCHR in US_THR to send a byte
  - If all the data have not been written, redo the two previous steps
  - Wait until LINTC in US_CSR rises
  - Check the LIN errors
- Case 2: NACT = SUBSCRIBE, the USART receives the response
  - Wait until RXRDY in US_CSR rises
  - Read RCHR in US_RHR
  - If all the data have not been read, redo the two previous steps
  - Wait until LINTC in US_CSR rises
  - Check the LIN errors
- Case 3: NACT = IGNORE, the USART is not concerned by the response
  - Wait until LINTC in US_CSR rises
  - Check the LIN errors

Figure 35-46. Master Node Configuration, NACT = PUBLISH

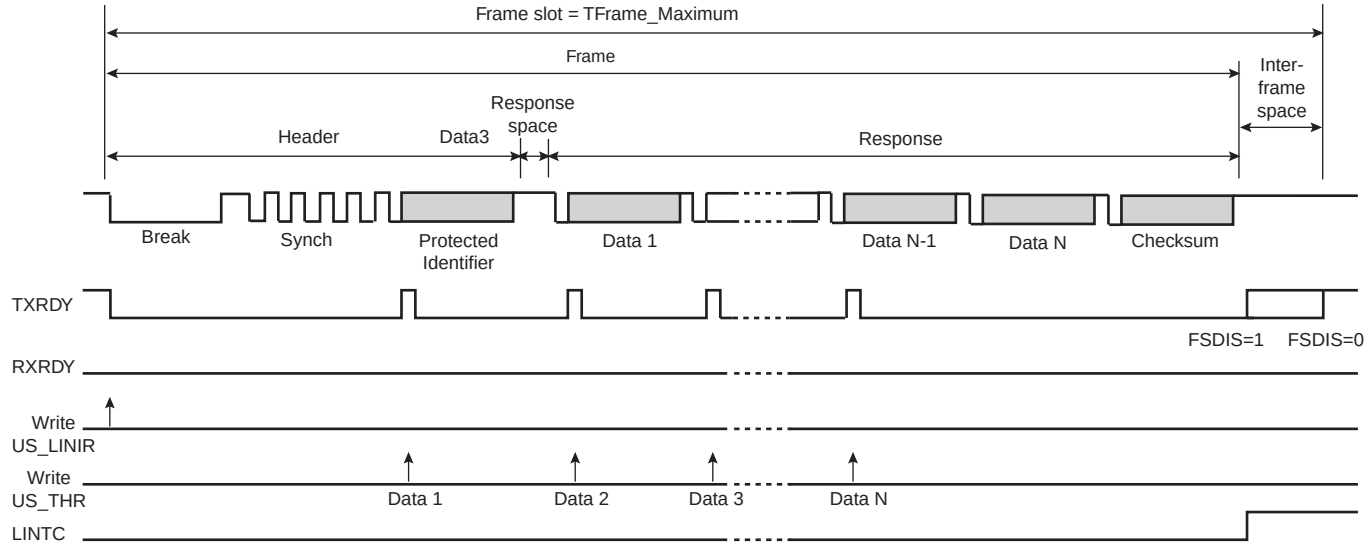


Figure 35-47. Master Node Configuration, NACT=SUBSCRIBE

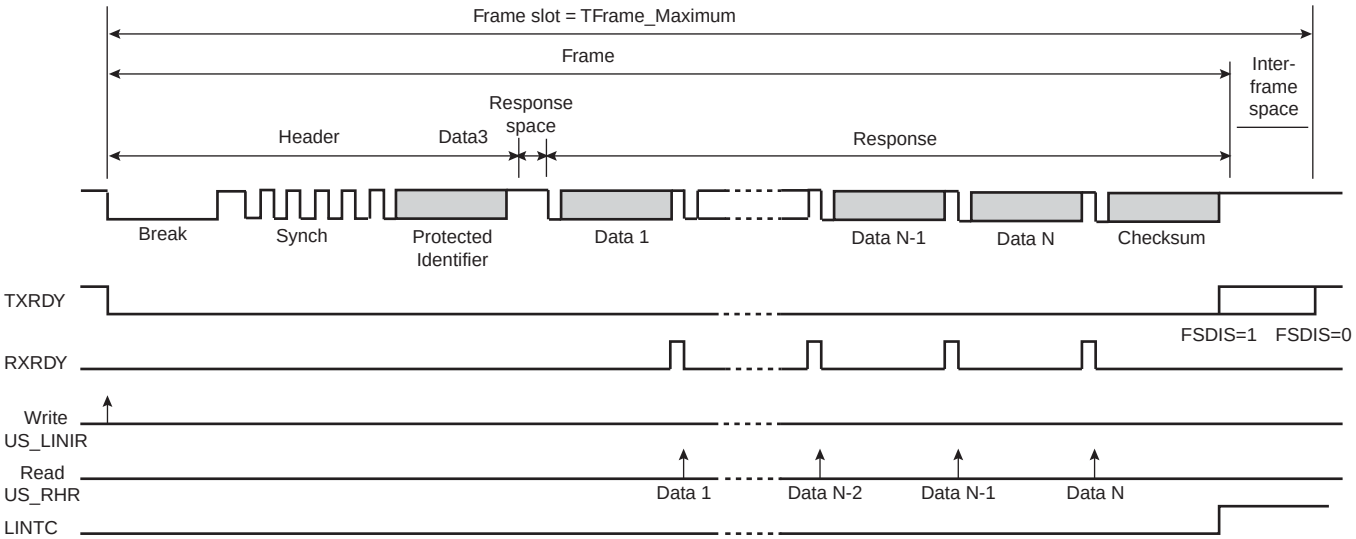
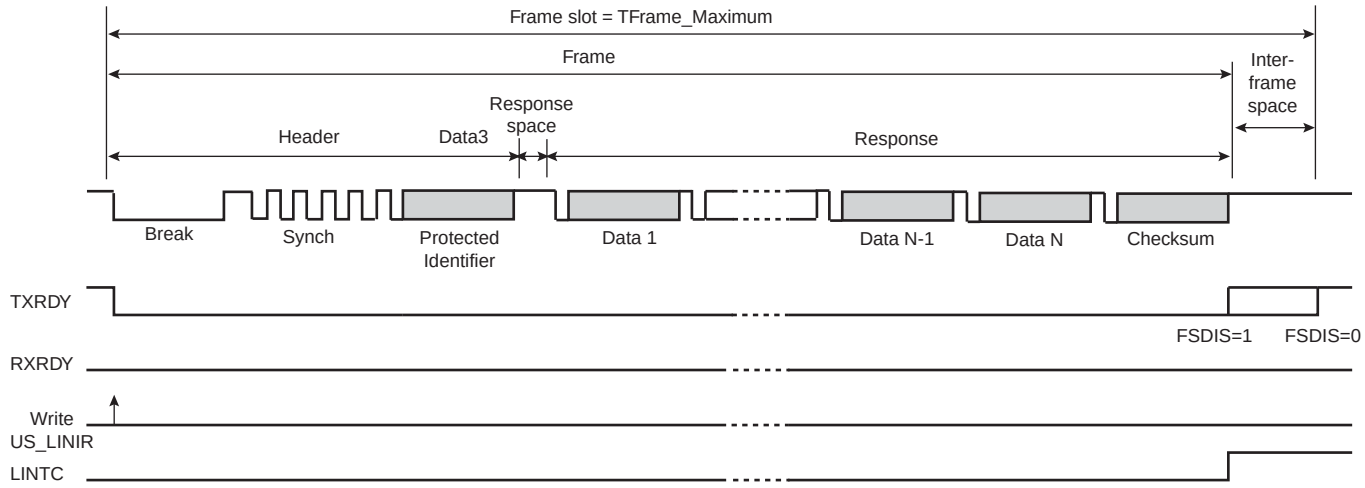


Figure 35-48. Master Node Configuration, NACT=IGNORE



Slave Node Configuration

- Write TXEN and RXEN in US_CR to enable both the transmitter and the receiver.
- Write USART_MODE in US_MR to select the LIN mode and the Slave Node configuration.
- Write CD and FP in US_BRGR to configure the baud rate.
- Wait until LINID in US_CSR rises
- Check LINISFE and LINPE errors
- Read IDCHR in US_RHR
- Write NACT, PARDIS, CHKDIS, CHKTYPE, DLCM and DLC in US_LINMR to configure the frame transfer.

IMPORTANT: if the NACT configuration for this frame is PUBLISH, the US_LINMR register, must be write with NACT = PUBLISH even if this field is already correctly configured, in order to set the TXREADY flag and the corresponding write transfer request.

What comes next depends on the NACT configuration:

- Case 1: NACT = PUBLISH, the LIN controller sends the response
  - Wait until TXRDY in US_CSR rises
  - Write TCHR in US_THR to send a byte
  - If all the data have not been written, redo the two previous steps
  - Wait until LINTC in US_CSR rises
  - Check the LIN errors
- Case 2: NACT = SUBSCRIBE, the USART receives the response
  - Wait until RXRDY in US_CSR rises
  - Read RCHR in US_RHR
  - If all the data have not been read, redo the two previous steps
  - Wait until LINTC in US_CSR rises
  - Check the LIN errors
- Case 3: NACT = IGNORE, the USART is not concerned by the response
  - Wait until LINTC in US_CSR rises
  - Check the LIN errors

Figure 35-49. Slave Node Configuration, NACT = PUBLISH

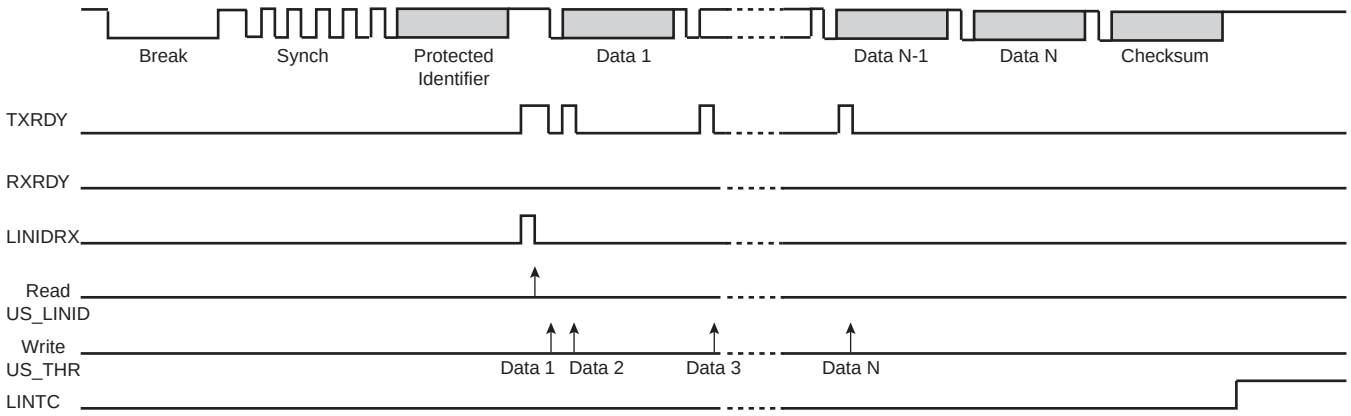


Figure 35-50. Slave Node Configuration, NACT = SUBSCRIBE

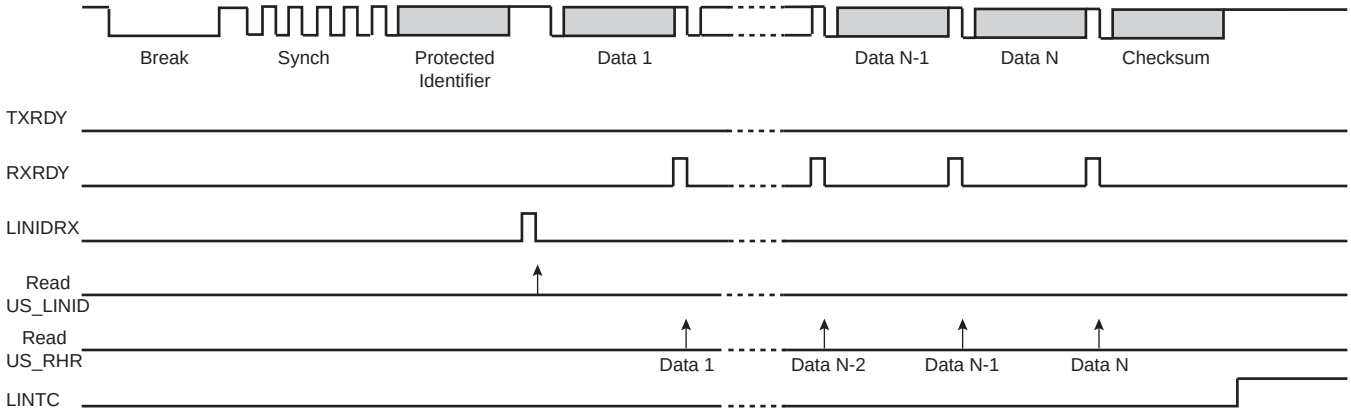
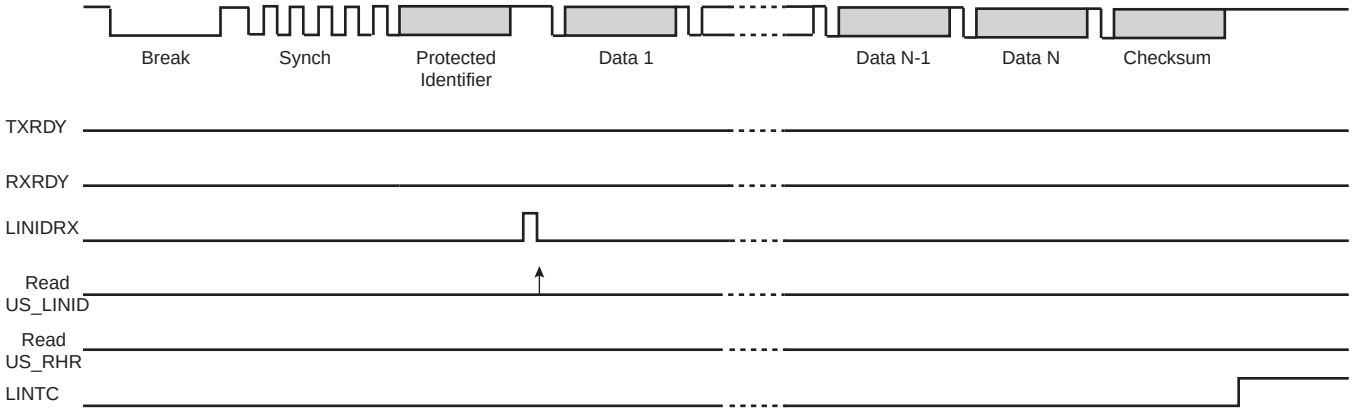


Figure 35-51. Slave Node Configuration, NACT = IGNORE



35.7.8.16 LIN Frame Handling With The PDC

The USART can be used in association with the PDC in order to transfer data directly into/from the on- and off-chip memories without any processor intervention.

The PDC uses the trigger flags, TXRDY and RXRDY, to write or read into the USART. The PDC always writes in the Transmit Holding register (US_THR) and it always reads in the Receive Holding register (US_RHR). The size of the data written or read by the PDC in the USART is always a byte.

Master Node Configuration

The user can choose between two PDC modes by the PDCM bit in the LIN Mode register (US_LINMR):

- PDCM = 1: the LIN configuration is stored in the WRITE buffer and it is written by the PDC in the Transmit Holding register US_THR (instead of the LIN Mode register US_LINMR). Because the PDC transfer size is limited to a byte, the transfer is split into two accesses. During the first access the bits, NACT, PARDIS, CHKDIS, CHKTYP, DLM and FSDIS are written. During the second access the 8-bit DLC field is written.
- PDCM = 0: the LIN configuration is not stored in the WRITE buffer and it must be written by the user in the LIN Mode register (US_LINMR).

The WRITE buffer also contains the Identifier and the DATA, if the USART sends the response (NACT = PUBLISH).

The READ buffer contains the DATA if the USART receives the response (NACT = SUBSCRIBE).

Figure 35-52. Master Node with PDC (PDCM = 1)

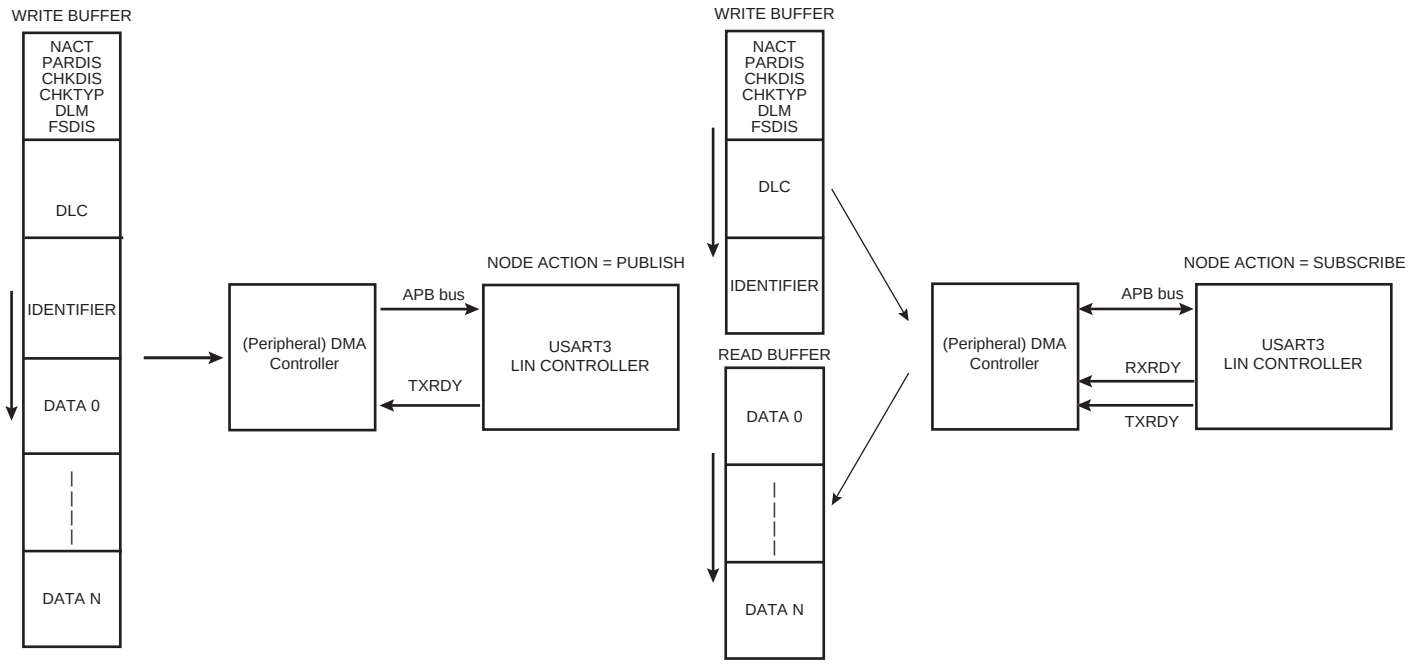
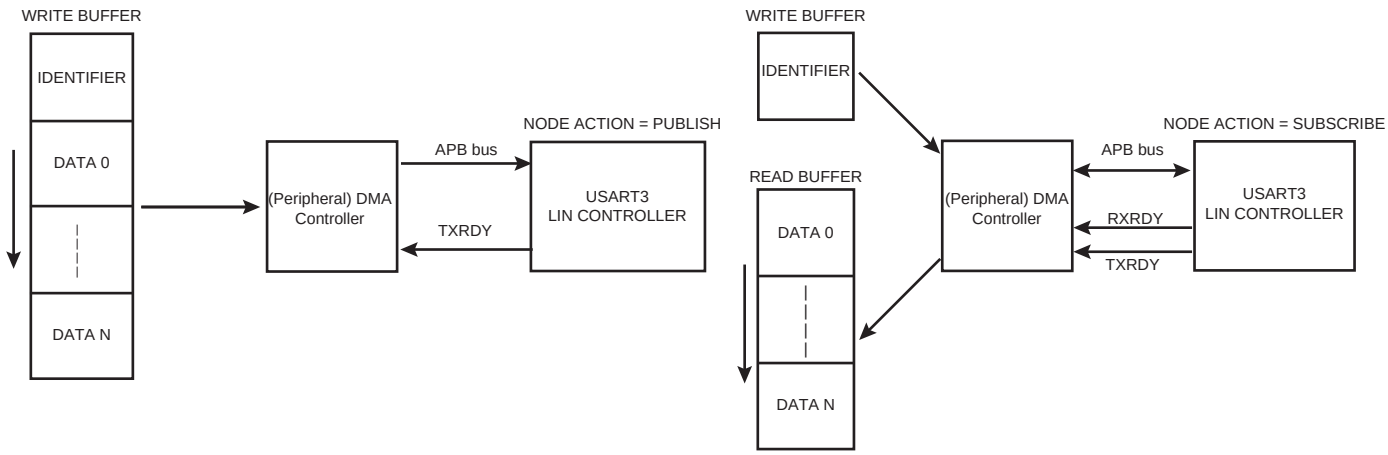


Figure 35-53. Master Node with PDC (PDCM = 0)



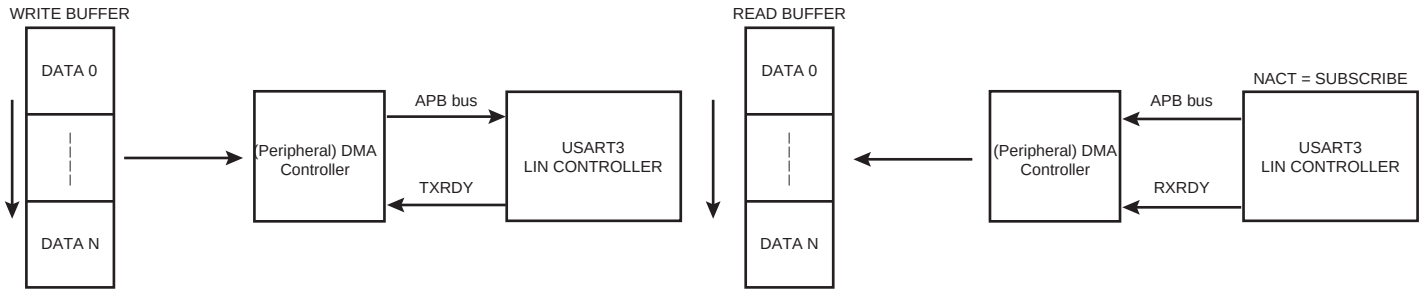
Slave Node Configuration

In this configuration, the PDC transfers only the DATA. The Identifier must be read by the user in the LIN Identifier register (US_LINIR). The LIN mode must be written by the user in the LIN Mode register (US_LINMR).

The WRITE buffer contains the DATA if the USART sends the response (NACT=PUBLISH).

The READ buffer contains the DATA if the USART receives the response (NACT=SUBSCRIBE).

Figure 35-54. Slave Node with PDC



35.7.8.17 Wake-up Request

Any node in a sleeping LIN cluster may request a wake-up.

In the LIN 2.0 specification, the wakeup request is issued by forcing the bus to the dominant state from 250 µs to 5 ms. For this, it is necessary to send the character 0xF0 in order to impose 5 successive dominant bits. Whatever the baud rate is, this character respects the specified timings.

- Baud rate min = 1 kbit/s -> Tbit = 1ms -> 5 Tbits = 5 ms
- Baud rate max = 20 kbit/s -> Tbi t= 50 µs -> 5 Tbits = 250 µs

In the LIN 1.3 specification, the wakeup request should be generated with the character 0x80 in order to impose 8 successive dominant bits.

The user can choose by the WKUPTYP bit in the LIN Mode register (US_LINMR) either to send a LIN 2.0 wakeup request (WKUPTYP=0) or to send a LIN 1.3 wakeup request (WKUPTYP=1).

A wake-up request is transmitted by writing the Control Register (US_CR) with the LINWKUP bit to 1. Once the transfer is completed, the LINTC flag is asserted in the Status Register (US_SR). It is cleared by writing the Control Register (US_CR) with the RSTSTA bit to 1.

35.7.8.18 Bus Idle Time-out

If the LIN bus is inactive for a certain duration, the slave nodes shall automatically enter in sleep mode. In the LIN 2.0 specification, this time-out is fixed at 4 seconds. In the LIN 1.3 specification, it is fixed at 25000 Tbits.

In Slave Node configuration, the Receiver Time-out 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 to go into sleep mode.

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 17-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.

If STTTO is performed, the counter clock is stopped until a first character is received.

If RETTO is performed, the counter starts counting down immediately from the value TO.

Table 35-16. Receiver Time-out programming

LIN Specification	Baud Rate	Time-out period	TO
2.0	1 000 bit/s	4s	4 000
	2 400 bit/s		9 600
	9 600 bit/s		38 400
	19 200 bit/s		76 800
	20 000 bit/s		80 000
1.3	-	25 000 Tbits	25 000

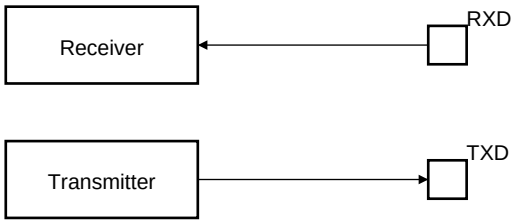
35.7.9 Test Modes

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.

35.7.9.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

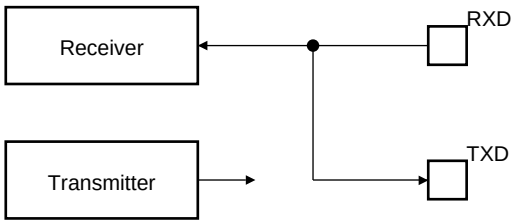
Figure 35-55. Normal Mode Configuration



35.7.9.2 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 35-56. 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.

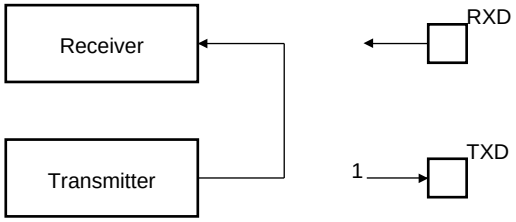
Figure 35-56. Automatic Echo Mode Configuration



35.7.9.3 Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 35-57. 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.

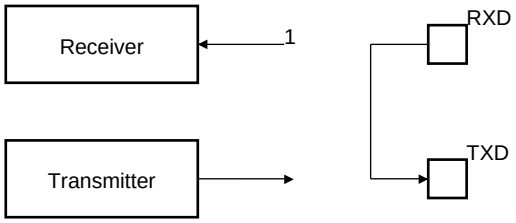
Figure 35-57. Local Loopback Mode Configuration



35.7.9.4 Remote Loopback Mode

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 35-58. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

Figure 35-58. Remote Loopback Mode Configuration



35.7.10 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 WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing 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"

35.8 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface

Table 35-17. 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	0x30011004
0x0054	LIN Mode Register	US_LINMR	Read-write	0x0
0x0058	LIN Identifier Register	US_LINIR	Read-write(1)	0x0
0xE4	Write Protect Mode Register	US_WPMR	Read-write	0x0
0xE8	Write Protect Status Register	US_WPSR	Read-only	0x0
0x5C - 0xF8	Reserved	–	–	–
0xFC	Version Register	US_VERSION	Read-only	0x–(2)
0x100 - 0x128	Reserved for PDC Registers	–	–	–

Notes: 1. Write is possible only in LIN Master node configuration.
2. Values in the Version Register vary with the version of the IP block implementation.

35.8.1 USART Control Register

Name: US_CR

Address: 0x40098000 (0), 0x4009C000 (1), 0x400A0000 (2), 0x400A4000 (3)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	LINWKUP	LINABT	RTSDIS/RCS	RTSEN/FCS	–	–
15	14	13	12	11	10	9	8
RETTO	RSTNACK	RSTIT	SENDA	STTTO	STPBRK	STTBRK	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

- **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, LINBE, LINISFE, LINIPE, LINCE, LINSNRE, LINID, LINTC, LINBK, **UNRE** 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.

- **RETTO: Rearm Time-out**

0: No effect

1: Restart Time-out

- **RTSEN: Request to Send Enable**

0: No effect.

1: Drives the pin RTS to 0.

- **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).

- **RTSDIS: Request to Send Disable**

0: No effect.

1: Drives the pin RTS to 1.

- **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).

- **LINABT: Abort LIN Transmission**

0: No effect.

1: Abort the current LIN transmission.

- **LINWKUP: Send LIN Wakeup Signal**

0: No effect:

1: Sends a wakeup signal on the LIN bus.

35.8.2 USART Mode Register

Name: US_MR

Address: 0x40098004 (0), 0x4009C004 (1), 0x400A0004 (2), 0x400A4004 (3)

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/CPOL
15	14	13	12	11	10	9	8
CHMODE		NBSTOP		PAR			SYNC/CPHA
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 854.

- **USART_MODE**

Value	Name	Description
0x0	NORMAL	Normal mode
0x1	RS485	RS485
0x2	HW_HANDSHAKING	Hardware Handshaking
0x4	IS07816_T_0	IS07816 Protocol: T = 0
0x6	IS07816_T_1	IS07816 Protocol: T = 1
0x8	IRDA	IrDA
0xA	LIN_MASTER	LIN Master
0xB	LIN_SLAVE	LIN Slave
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=8) 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.

- **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.

- **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.

- **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.

- **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.

Note: In SPI master mode, if INACK = 0 the character transmission starts as soon as a character is written into US_THR register (assuming TXRDY was set). When INACK is 1, an additional condition must be met. The character transmission starts when a character is written and only if RXRDY flag is cleared (Receiver Holding Register has been read).

- **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**

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.

35.8.3 USART Interrupt Enable Register

Name: US_IER

Address: 0x40098008 (0), 0x4009C008 (1), 0x400A0008 (2), 0x400A4008 (3)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	LINSNRE	LINCE	LINIPE	LINISFE	LINBE	MANE
23	22	21	20	19	18	17	16
–	–	–		CTSIC	–	–	–
15	14	13	12	11	10	9	8
LINTC	LINID	NACK/LINBK	RXBUFF	TXBUFE	ITER/UNRE	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **RXBRK: Receiver Break Interrupt Enable**
- **ENDRX: End of Receive Transfer Interrupt Enable**
- **ENDTX: End of Transmit Interrupt Enable**
- **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**
- **UNRE: SPI Underrun Error**
- **TXBUFE: Buffer Empty Interrupt Enable**
- **RXBUFF: Buffer Full Interrupt Enable**
- **NACK: Non Acknowledge Interrupt Enable**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Enable**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Enable**

- **LINTC: LIN Transfer Completed Interrupt Enable**
- **CTSIC: Clear to Send Input Change Interrupt Enable**
- **MANE: Manchester Error Interrupt Enable**
- **LINBE: LIN Bus Error Interrupt Enable**
- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Enable**
- **LINIPE: LIN Identifier Parity Interrupt Enable**
- **LINCE: LIN Checksum Error Interrupt Enable**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Enable**

35.8.4 USART Interrupt Disable Register

Name: US_IDR

Address: 0x4009800C (0), 0x4009C00C (1), 0x400A000C (2), 0x400A400C (3)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	LINSNRE	LINCE	LINIPE	LINISFE	LINBE	MANE
23	22	21	20	19	18	17	16
–	–	–		CTSIC	–	–	–
15	14	13	12	11	10	9	8
LINTC	LINID	NACK/LINBK	RXBUFF	TXBUFE	ITER/UNRE	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **RXBRK: Receiver Break Interrupt Disable**
- **ENDRX: End of Receive Transfer Interrupt Disable**
- **ENDTX: End of Transmit Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **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 Disable**
- **UNRE: SPI Underrun Error Disable**
- **TXBUFE: Buffer Empty Interrupt Disable**
- **RXBUFF: Buffer Full Interrupt Disable**
- **NACK: Non Acknowledge Interrupt Disable**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Disable**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Disable**

- **LINTC: LIN Transfer Completed Interrupt Disable**
- **CTSIC: Clear to Send Input Change Interrupt Disable**
- **MANE: Manchester Error Interrupt Disable**
- **LINBE: LIN Bus Error Interrupt Disable**
- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Disable**
- **LINIPE: LIN Identifier Parity Interrupt Disable**
- **LINCE: LIN Checksum Error Interrupt Disable**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Disable**

35.8.5 USART Interrupt Mask Register

Name: US_IMR

Address: 0x40098010 (0), 0x4009C010 (1), 0x400A0010 (2), 0x400A4010 (3)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	LINSNRE	LINCE	LINIPE	LINISFE	LINBE	MANE
23	22	21	20	19	18	17	16
–	–	–		CTSIC	–	–	–
15	14	13	12	11	10	9	8
LINTC	LINID	NACK/LINBK	RXBUFF	TXBUFE	ITER/UNRE	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **RXBRK: Receiver Break Interrupt Mask**
- **ENDRX: End of Receive Transfer Interrupt Mask**
- **ENDTX: End of Transmit Interrupt Mask**
- **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 Mask**
- **UNRE: SPI Underrun Error Mask**
- **TXBUFE: Buffer Empty Interrupt Mask**
- **RXBUFF: Buffer Full Interrupt Mask**
- **NACK: Non Acknowledge Interrupt Mask**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Mask**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Mask**

- **LINTC: LIN Transfer Completed Interrupt Mask**
- **CTSIC: Clear to Send Input Change Interrupt Mask**
- **MANE: Manchester Error Interrupt Mask**
- **LINBE: LIN Bus Error Interrupt Mask**
- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Mask**
- **LINIPE: LIN Identifier Parity Interrupt Mask**
- **LINCE: LIN Checksum Error Interrupt Mask**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Mask**

35.8.6 USART Channel Status Register

Name: US_CSR

Address: 0x40098014 (0), 0x4009C014 (1), 0x400A0014 (2), 0x400A4014 (3)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	LINSNRE	LINCE	LINIPE	LINISFE	LINBE	MANERR
23	22	21	20	19	18	17	16
CTS/LINBLS	–	–	–	CTSIC	–	–	–
15	14	13	12	11	10	9	8
LINTC	LINID	NACK/LINBK	RXBUFF	TXBUFE	ITER/UNRE	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **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.

- **UNRE: SPI Underrun Error**

– Applicable if USART operates in SPI Slave Mode (USART_MODE = 0xF):

UNRE = 0: No SPI underrun error has occurred since the last RSTSTA.

UNRE = 1: At least one SPI underrun error has occurred 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 Acknowledge Interrupt**

0: Non Acknowledge has not been detected since the last RSTNACK.

1: At least one Non Acknowledge has been detected since the last RSTNACK.

- **LINBK: LIN Break Sent or LIN Break Received**

– Applicable if USART operates in LIN Master Mode (USART_MODE = 0xA):

0: No LIN Break has been sent since the last RSTSTA.

1: At least one LIN Break has been sent since the last RSTSTA

– If USART operates in LIN Slave Mode (USART_MODE = 0xB):

0: No LIN Break has received sent since the last RSTSTA.

1: At least one LIN Break has been received since the last RSTSTA.

- **LINID: LIN Identifier Sent or LIN Identifier Received**
  – If USART operates in LIN Master Mode (USART_MODE = 0xA):

0: No LIN Identifier has been sent since the last RSTSTA.

1: At least one LIN Identifier has been sent since the last RSTSTA.

  – If USART operates in LIN Slave Mode (USART_MODE = 0xB):

0: No LIN Identifier has been received since the last RSTSTA.

1: At least one LIN Identifier has been received since the last RSTSTA

- **LINTC: LIN Transfer Completed**

0: The USART is idle or a LIN transfer is ongoing.

1: A LIN transfer has been completed since the last RSTSTA.

- **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.

- **CTS: Image of CTS Input**

0: CTS is set to 0.

1: CTS is set to 1.

- **LINBLS: LIN Bus Line Status**
  – Applicable if USART operates in LIN Mode (USART_MODE = 0xA or USART_MODE = 0xB):

0: LIN Bus Line is set to 0.

1: LIN Bus Line 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.

- **LINBE: LIN Bit Error**

0: No Bit Error has been detected since the last RSTSTA.

1: A Bit Error has been detected since the last RSTSTA.

- **LINISFE: LIN Inconsistent Synch Field Error**

0: No LIN Inconsistent Synch Field Error has been detected since the last RSTSTA

1: The USART is configured as a Slave node and a LIN Inconsistent Synch Field Error has been detected since the last RSTSTA.

- **LINIPE: LIN Identifier Parity Error**

0: No LIN Identifier Parity Error has been detected since the last RSTSTA.

1: A LIN Identifier Parity Error has been detected since the last RSTSTA.

- **LINCE: LIN Checksum Error**

0: No LIN Checksum Error has been detected since the last RSTSTA.

1: A LIN Checksum Error has been detected since the last RSTSTA.

- **LINSNRE: LIN Slave Not Responding Error**

0: No LIN Slave Not Responding Error has been detected since the last RSTSTA.

1: A LIN Slave Not Responding Error has been detected since the last RSTSTA.

35.8.7 USART Receive Holding Register

Name: US_RHR

Address: 0x40098018 (0), 0x4009C018 (1), 0x400A0018 (2), 0x400A4018 (3)

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.

35.8.8 USART Transmit Holding Register

Name: US_THR

Address: 0x4009801C (0), 0x4009C01C (1), 0x400A001C (2), 0x400A401C (3)

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.

35.8.9 USART Baud Rate Generator Register

Name: US_BRGR

Address: 0x40098020 (0), 0x4009C020 (1), 0x400A0020 (2), 0x400A4020 (3)

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 854.

- **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: Baudrate resolution, defined by FP x 1/8.

35.8.10 USART Receiver Time-out Register

Name: US_RTOR

Address: 0x40098024 (0), 0x4009C024 (1), 0x400A0024 (2), 0x400A4024 (3)

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	TO

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 854.

- **TO: Time-out Value**

0: The Receiver Time-out is disabled.

1 - 131071: The Receiver Time-out is enabled and the Time-out delay is TO x Bit Period.

35.8.11 USART Transmitter Timeguard Register

Name: US_TTGR

Address: 0x40098028 (0), 0x4009C028 (1), 0x400A0028 (2), 0x400A4028 (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
–	–	–	–	–	–	–	–
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 854.

- **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.

35.8.12 USART FI DI RATIO Register

Name: US_FIDI

Address: 0x40098040 (0), 0x4009C040 (1), 0x400A0040 (2), 0x400A4040 (3)

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 854.

- **FI_DI_RATIO: FI Over DI Ratio Value**

0: If ISO7816 mode is selected, the Baud Rate Generator generates no signal.

1 - 2047: If ISO7816 mode is selected, the Baud Rate is the clock provided on SCK divided by FI_DI_RATIO.

35.8.13 USART Number of Errors Register

Name: US_NER

Address: 0x40098044 (0), 0x4009C044 (1), 0x400A0044 (2), 0x400A4044 (3)

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							

- **NB_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

35.8.14 USART IrDA FILTER Register

Name: US_IF

Address: 0x4009804C (0), 0x4009C04C (1), 0x400A004C (2), 0x400A404C (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
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
IRDA_FILTER							

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 854.

- **IRDA_FILTER: IrDA Filter**

Sets the filter of the IrDA demodulator.

35.8.15 USART Manchester Configuration Register

Name: US_MAN

Address: 0x40098050 (0), 0x4009C050 (1), 0x400A0050 (2), 0x400A4050 (3)

Access: Read-write

31	30	29	28	27	26	25	24
–	DRIFT	1	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 854.

- **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.

- **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.

35.8.16 USART LIN Mode Register

Name: US_LINMR

Address: 0x40098054 (0), 0x4009C054 (1), 0x400A0054 (2), 0x400A4054 (3)

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	PDCM

15	14	13	12	11	10	9	8
DLC							

7	6	5	4	3	2	1	0
WKUPTYP	FSDIS	DLM	CHKTYP	CHKDIS	PARDIS	NACT	

- **NACT: LIN Node Action**

Value	Name	Description
00	PUBLISH	The USART transmits the response.
01	SUBSCRIBE	The USART receives the response.
10	IGNORE	The USART does not transmit and does not receive the response.

Values which are not listed in the table must be considered as “reserved”.

- **PARDIS: Parity Disable**

0: In Master node configuration, the Identifier Parity is computed and sent automatically. In Master node and Slave node configuration, the parity is checked automatically.

1:Whatever the node configuration is, the Identifier parity is not computed/sent and it is not checked.

- **CHKDIS: Checksum Disable**

0: In Master node configuration, the checksum is computed and sent automatically. In Slave node configuration, the checksum is checked automatically.

1: Whatever the node configuration is, the checksum is not computed/sent and it is not checked.

- **CHKTYP: Checksum Type**

0: LIN 2.0 “Enhanced” Checksum

1: LIN 1.3 “Classic” Checksum

- **DLM: Data Length Mode**

0: The response data length is defined by the field DLC of this register.

1: The response data length is defined by the bits 5 and 6 of the Identifier (IDCHR in US_LINIR).

- **FSDIS: Frame Slot Mode Disable**

0: The Frame Slot Mode is enabled.

1: The Frame Slot Mode is disabled.

- **WKUPTYP: Wakeup Signal Type**

0: setting the bit LINWKUP in the control register sends a LIN 2.0 wakeup signal.

1: setting the bit LINWKUP in the control register sends a LIN 1.3 wakeup signal.

- **DLC: Data Length Control**

0 - 255: Defines the response data length if DLM=0,in that case the response data length is equal to DLC+1 bytes.

- **PDCM: PDC Mode**

0: The LIN mode register US_LINMR is not written by the PDC.

1: The LIN mode register US_LINMR (excepting that flag) is written by the PDC.

35.8.17 USART LIN Identifier Register

Name: US_LINIR

Address: 0x40098058 (0), 0x4009C058 (1), 0x400A0058 (2), 0x400A4058 (3)

Access: Read-write or 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
IDCHR							

- **IDCHR: Identifier Character**

If USART_MODE=0xA (Master node configuration):

IDCHR is Read-write and its value is the Identifier character to be transmitted.

if USART_MODE=0xB (Slave node configuration):

IDCHR is Read-only and its value is the last Identifier character that has been received.

35.8.18 USART Write Protect Mode Register

Name: US_WPMR

Address: 0x400980E4 (0), 0x4009C0E4 (1), 0x400A00E4 (2), 0x400A40E4 (3)

Access: Read-write

Reset: See Table 35-17

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 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 827
- “USART Baud Rate Generator Register” on page 843
- “USART Receiver Time-out Register” on page 844
- “USART Transmitter Timeguard Register” on page 845
- “USART FI DI RATIO Register” on page 846
- “USART IrDA FILTER Register” on page 848
- “USART Manchester Configuration Register” on page 849

- **WPKEY: Write Protect KEY**

Should be written at value 0x555341 (“USA” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

35.8.19 USART Write Protect Status Register

Name: US_WPSR

Address: 0x400980E8 (0), 0x4009C0E8 (1), 0x400A00E8 (2), 0x400A40E8 (3)

Access: Read-only

Reset: See Table 35-17

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 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.

36. Timer Counter (TC)

36.1 Description

A Timer Counter (TC) module includes three identical TC channels. The number of implemented TC modules is device-specific.

Each TC 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 TC embeds a quadrature decoder (QDEC) connected in front of the timers and driven by TIOA0, TIOB0 and TIOB1 inputs. When enabled, the QDEC 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 TC block has two global registers which act upon all TC channels:

- Block Control Register (TC_BCR)—allows channels to be started simultaneously with the same instruction
- Block Mode Register (TC_BMR)—defines the external clock inputs for each channel, allowing them to be chained

36.2 Embedded Characteristics

- Total number of TC channels: 9
- TC channel size: 32-bit
- Wide range of functions including:
  - Frequency measurement
  - Event counting
  - Interval measurement
  - Pulse generation
  - Delay timing
  - Pulse Width Modulation
  - Up/down capabilities
  - Quadrature decoder
  - 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
- Internal interrupt signal
- Compare event fault generation for PWM
- Register Write Protection

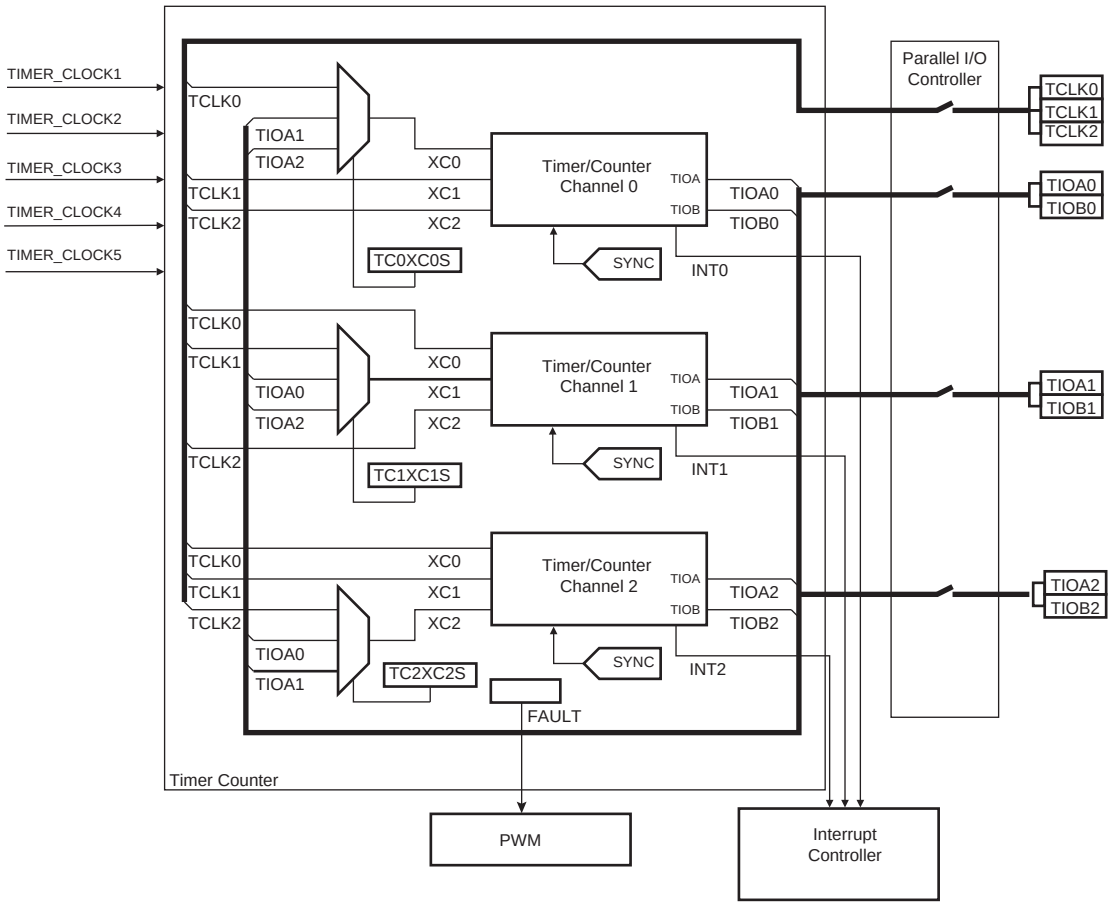
36.3 Block Diagram

Table 36-1. Timer Counter Clock Assignment

Name	Definition
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5	SLCK

Note: 1. When SLCK is selected for Peripheral Clock (CSS = 0 in PMC Master Clock Register), SLCK input is equivalent to Peripheral Clock.

Figure 36-1. Timer Counter Block Diagram



Note: The QDEC connections are detailed in Figure 36-15.

Table 36-2. 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)

36.4 Pin Name List

Table 36-3. 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

36.5 Product Dependencies

36.5.1 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 36-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TC0	TCLK0	PB26	B
TC0	TCLK1	PA4	A
TC0	TCLK2	PA7	A
TC0	TIOA0	PB25	B
TC0	TIOA1	PA2	A
TC0	TIOA2	PA5	A
TC0	TIOB0	PB27	B
TC0	TIOB1	PA3	A
TC0	TIOB2	PA6	A
TC1	TCLK3	PA22	B
TC1	TCLK4	PA23	B
TC1	TCLK5	PB16	A
TC1	TIOA3	PB0	B
TC1	TIOA3	PE9	A

Table 36-4. I/O Lines (Continued)

Instance	Signal	I/O Line	Peripheral
TC1	TIOA4	PB2	B
TC1	TIOA4	PE11	A
TC1	TIOA5	PB4	B
TC1	TIOA5	PE13	A
TC1	TIOB3	PB1	B
TC1	TIOB3	PE10	A
TC1	TIOB4	PB3	B
TC1	TIOB4	PE12	A
TC1	TIOB5	PB5	B
TC1	TIOB5	PE14	A
TC2	TCLK6	PC27	B
TC2	TCLK7	PC30	B
TC2	TCLK8	PD9	B
TC2	TIOA6	PC25	B
TC2	TIOA7	PC28	B
TC2	TIOA8	PD7	B
TC2	TIOB6	PC26	B
TC2	TIOB7	PC29	B
TC2	TIOB8	PD8	B

36.5.2 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.

36.5.3 Interrupt Sources

The TC has an interrupt line connected to the interrupt controller. Handling the TC interrupt requires programming the interrupt controller before configuring the TC.

Table 36-5. Peripheral IDs

Instance	ID
TC0	27
TC1	28
TC2	29

36.5.4 Fault Output

The TC has the FAULT output internally connected to the fault input of PWM. Refer to Section 36.6.16 ”Fault Mode”, and to the implementation of the Pulse Width Modulation (PWM) in this product.

36.6 Functional Description

36.6.1 Description

All channels of the Timer Counter are independent and identical in operation except when the QDEC is enabled. The registers for channel programming are listed in Table 36-6 “Register Mapping”.

36.6.2 32-bit Counter

Each 32-bit 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 2^{32}-1 and passes to zero, an overflow occurs and the COVFS bit in the TC Status Register (TC_SR) is set.

The current value of the counter is accessible in real time by reading the TC Counter Value Register (TC_CV). The counter can be reset by a trigger. In this case, the counter value passes to zero on the next valid edge of the selected clock.

36.6.3 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 Block Mode Register (TC_BMR). See Figure 36-2.

Each channel can independently select an internal or external clock source for its counter:

- External clock signals(1): XC0, XC1 or XC2
- Internal clock signals: MCK/2, MCK/8, MCK/32, MCK/128, SLCK

This selection is made by the TCCLKS bits in the TC Channel Mode Register (TC_CMR).

The selected clock can be inverted with the CLKI bit in the TC_CMR. 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 TC_CMR defines this signal (none, XC0, XC1, XC2). See Figure 36-3.

Note: 1. In all cases, if an external clock is used, the duration of each of its levels must be longer than the peripheral clock period. The external clock frequency must be at least 2.5 times lower than the peripheral clock.

Figure 36-2. Clock Chaining Selection

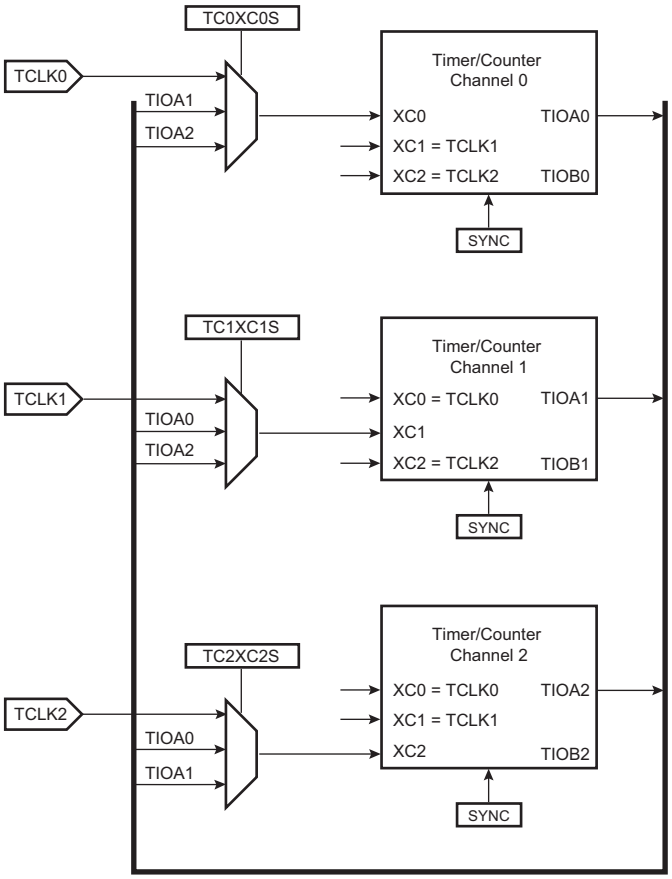
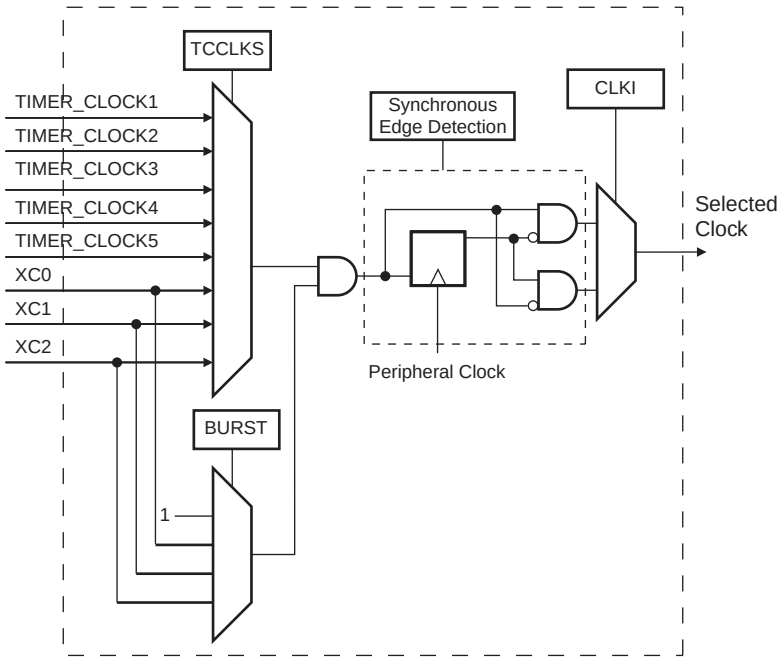


Figure 36-3. Clock Selection

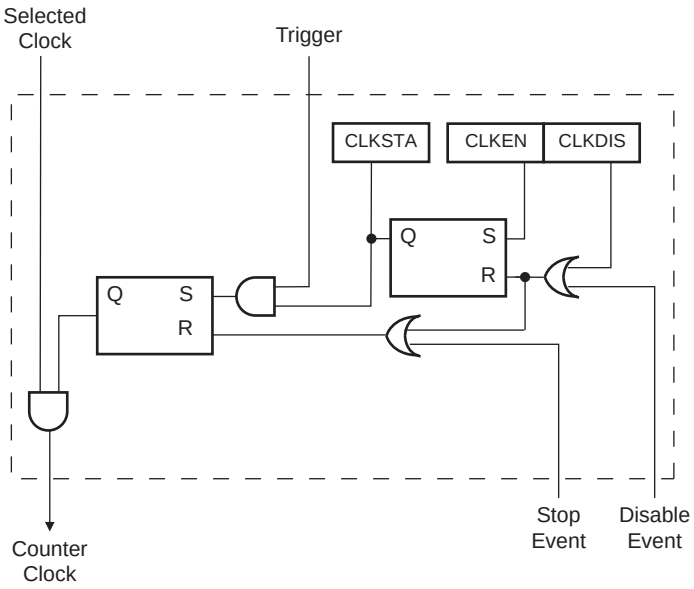


36.6.4 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 36-4.

- The clock can be enabled or disabled by the user with the CLKEN and the CLKDIS commands in the TC Channel Control Register (TC_CCR). In Capture mode it can be disabled by an RB load event if LDBDIS is set to 1 in the TC_CMR. In Waveform mode, it can be disabled by an RC Compare event if CPCDIS is set to 1 in TC_CMR. When disabled, the start or the stop actions have no effect: only a CLKEN command in the TC_CCR can re-enable the clock. When the clock is enabled, the CLKSTA bit is set in the TC_SR.
- 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_CMR) or an RC compare event in Waveform mode (CPCSTOP = 1 in TC_CMR). The start and the stop commands are effective only if the clock is enabled.

Figure 36-4. Clock Control



36.6.5 Operating Modes

Each channel can operate independently 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_CMR.

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.

36.6.6 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 the 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 bit ENETRG in the TC_CMR.

If an external trigger is used, the duration of the pulses must be longer than the peripheral clock period in order to be detected.

36.6.7 Capture Mode

Capture mode is entered by clearing the WAVE bit in the TC_CMR.

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 36-5 shows the configuration of the TC channel when programmed in Capture mode.

36.6.8 Capture Registers A and B

Registers A and B (RA and RB) are used as capture registers. They can be loaded with the counter value when a programmable event occurs on the signal TIOA.

The LDRA field in the TC_CMR defines the TIOA selected edge for the loading of register A, and the LDRB field defines the TIOA selected edge for the loading of Register B.

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.

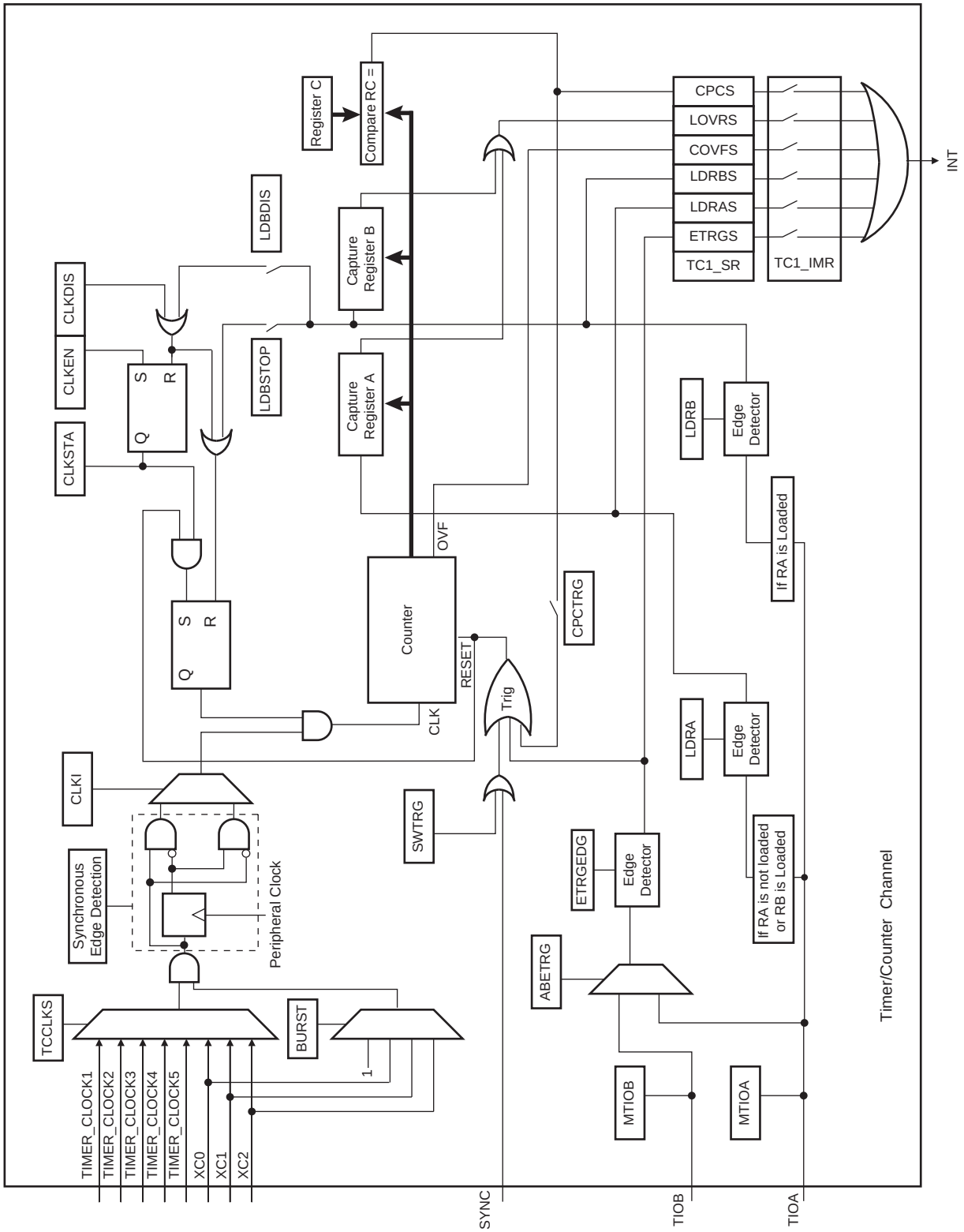
Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS bit) in the TC_SR. In this case, the old value is overwritten.

36.6.9 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 selects TIOA or TIOB input signal as an external trigger . The External Trigger Edge Selection parameter (ETRGEDG field in TC_CMR) defines the edge (rising, falling, or both) detected to generate an external trigger. If ETRGEDG = 0 (none), the external trigger is disabled.

Figure 36-5. Capture Mode



36.6.10 Waveform Mode

Waveform mode is entered by setting the TC_CMRx.WAVE bit.

In Waveform mode, the TC channel generates one or two 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 36-6 shows the configuration of the TC channel when programmed in Waveform operating mode.

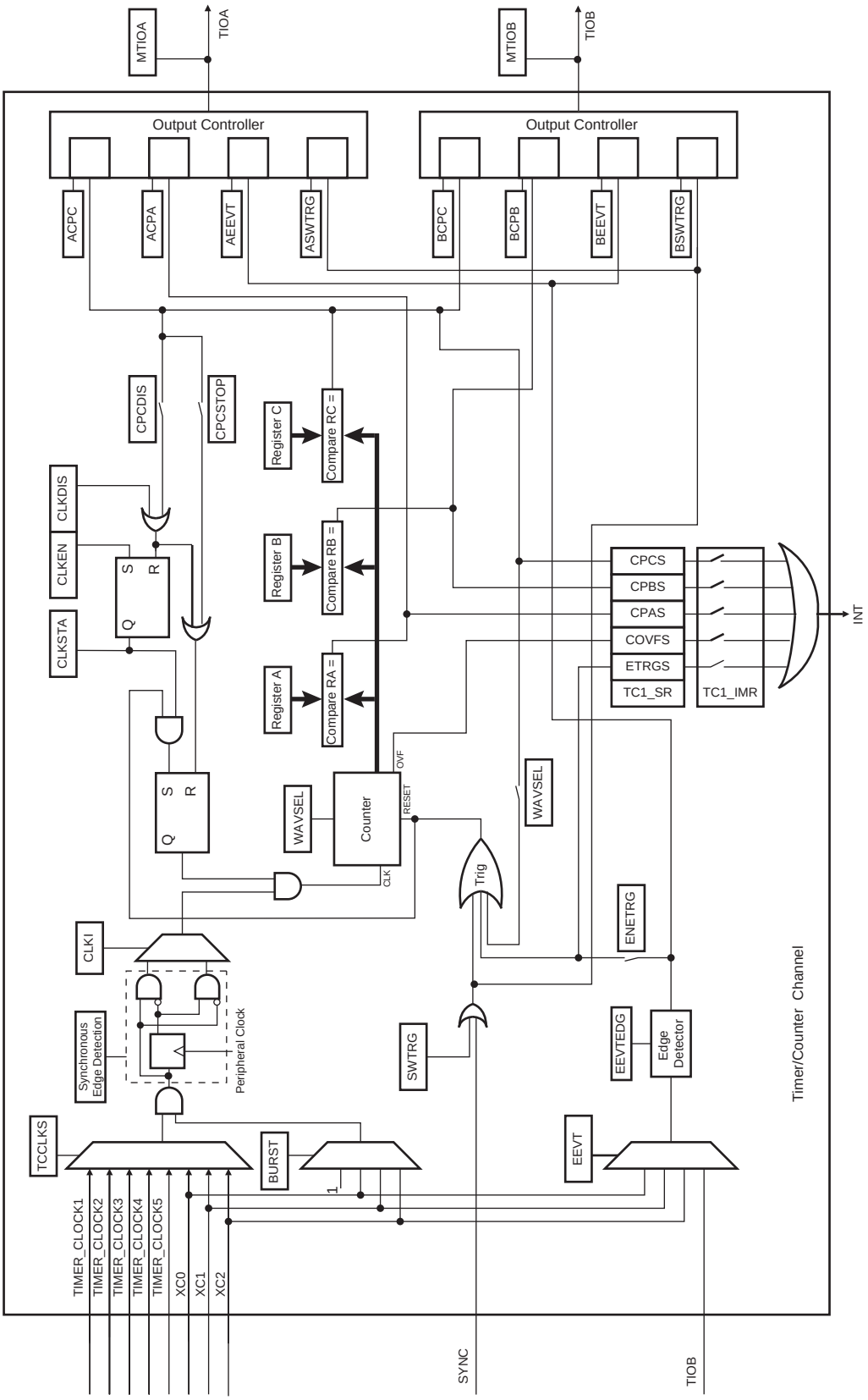
36.6.11 Waveform Selection

Depending on the WAVSEL parameter in TC_CMR, the behavior of TC_CV varies.

With any selection, TC_RA, TC_RB and TC_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 36-6. Waveform Mode



36.6.11.1 WAVSEL = 00

When WAVSEL = 00, the value of TC_CV is incremented from 0 to 2^{32}-1. Once 2^{32}-1 has been reached, the value of TC_CV is reset. Incrementation of TC_CV starts again and the cycle continues. See Figure 36-7.

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 36-8.

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_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 36-7. WAVSEL = 00 without trigger

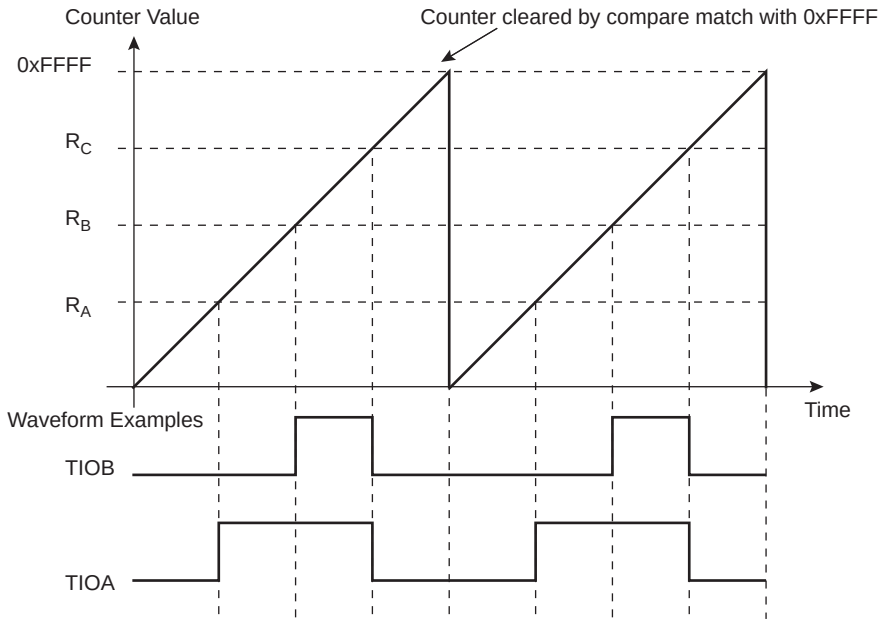
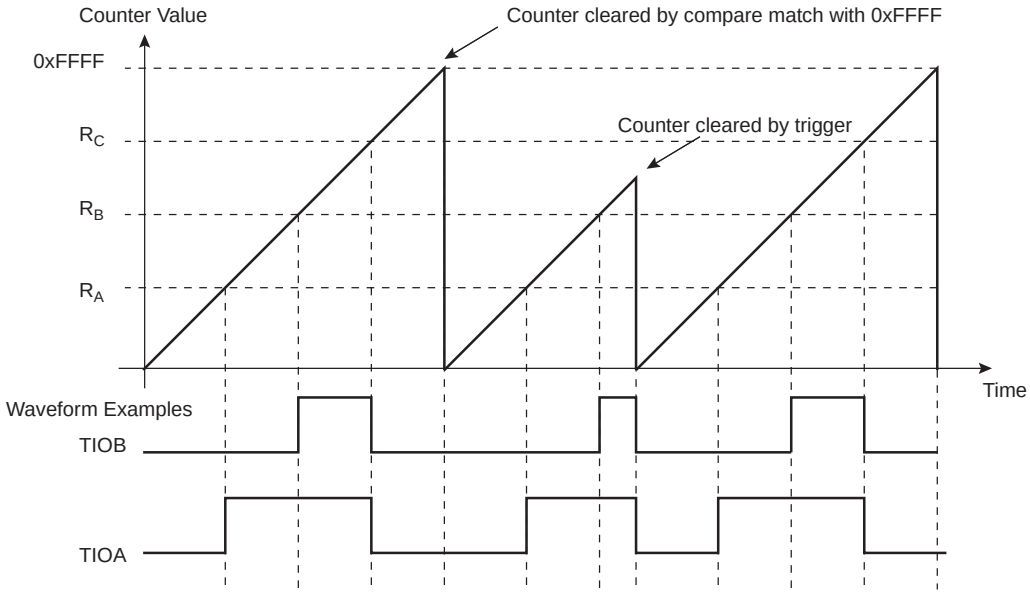


Figure 36-8. WAVSEL = 00 with Trigger



36.6.11.2 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 36-9.

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 36-10.

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 36-9. WAVSEL = 10 without Trigger

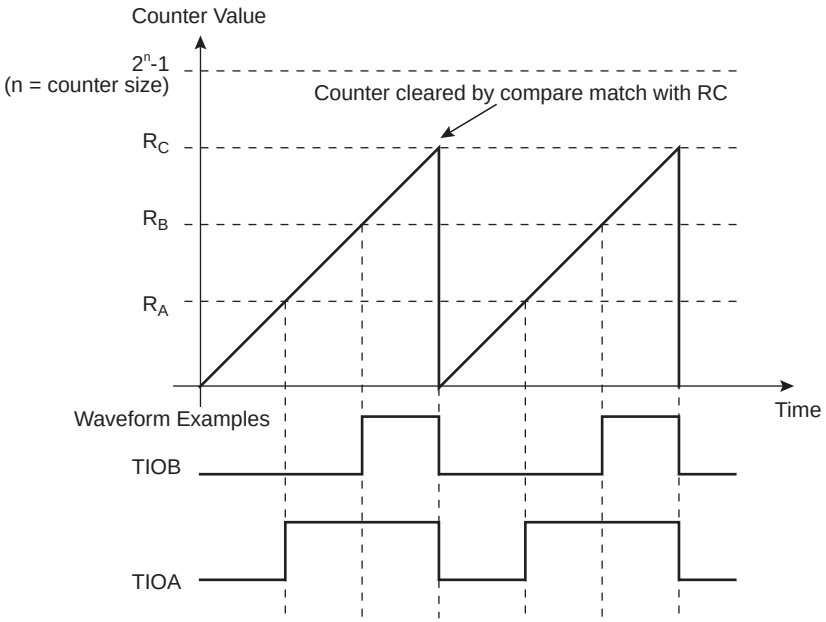
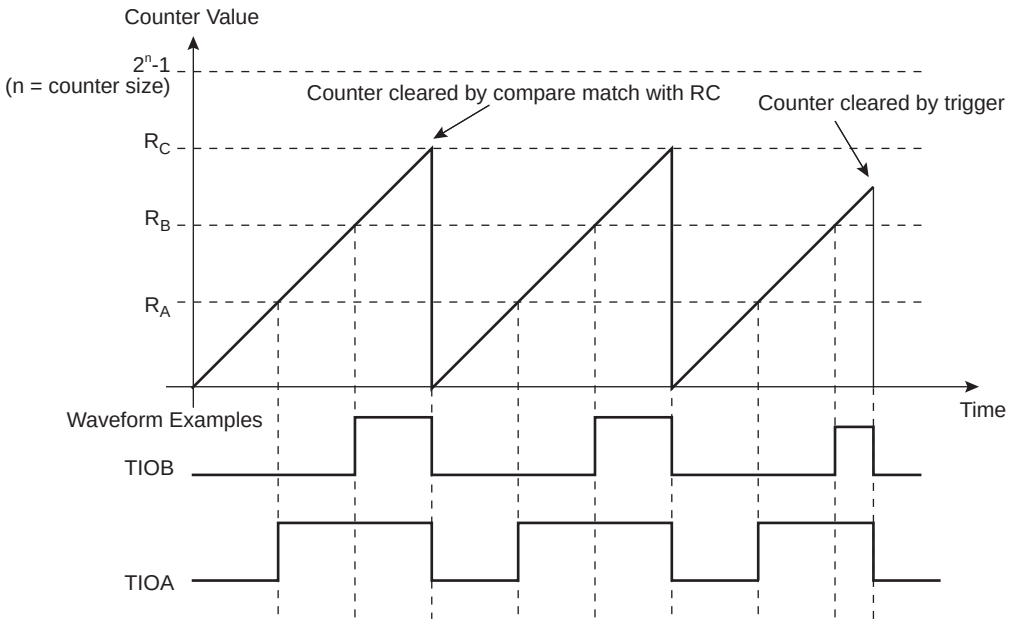


Figure 36-10. WAVSEL = 10 with Trigger



36.6.11.3 WAVSEL = 01

When WAVSEL = 01, the value of TC_CV is incremented from 0 to 2^{32}-1 . Once 2^{32}-1 is reached, the value of TC_CV is decremented to 0, then re-incremented to 2^{32}-1 and so on. See Figure 36-11.

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 36-12.

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 36-11. WAVSEL = 01 without Trigger

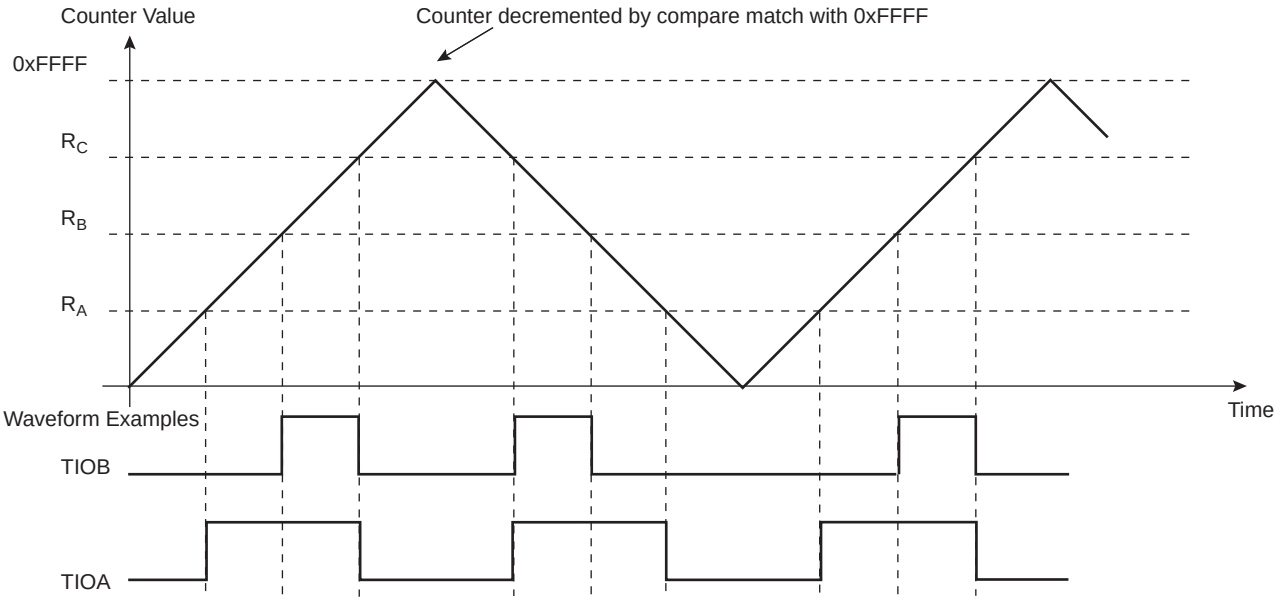
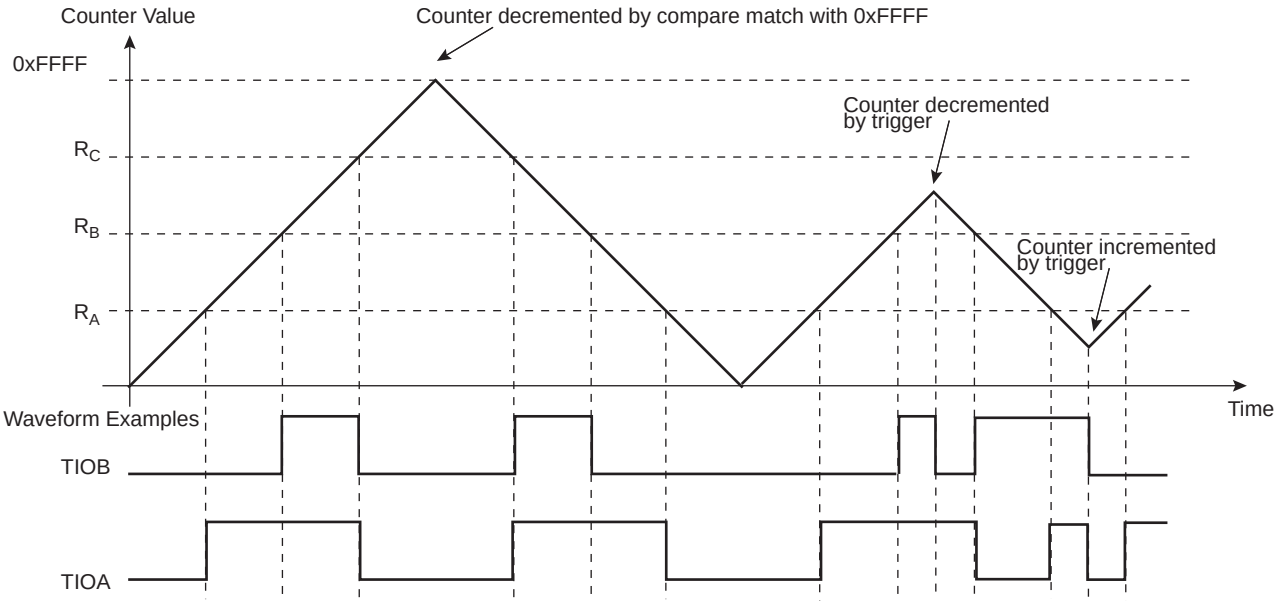


Figure 36-12. WAVSEL = 01 with Trigger



36.6.11.4 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 36-13.

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 36-14.

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 36-13. WAVSEL = 11 without Trigger

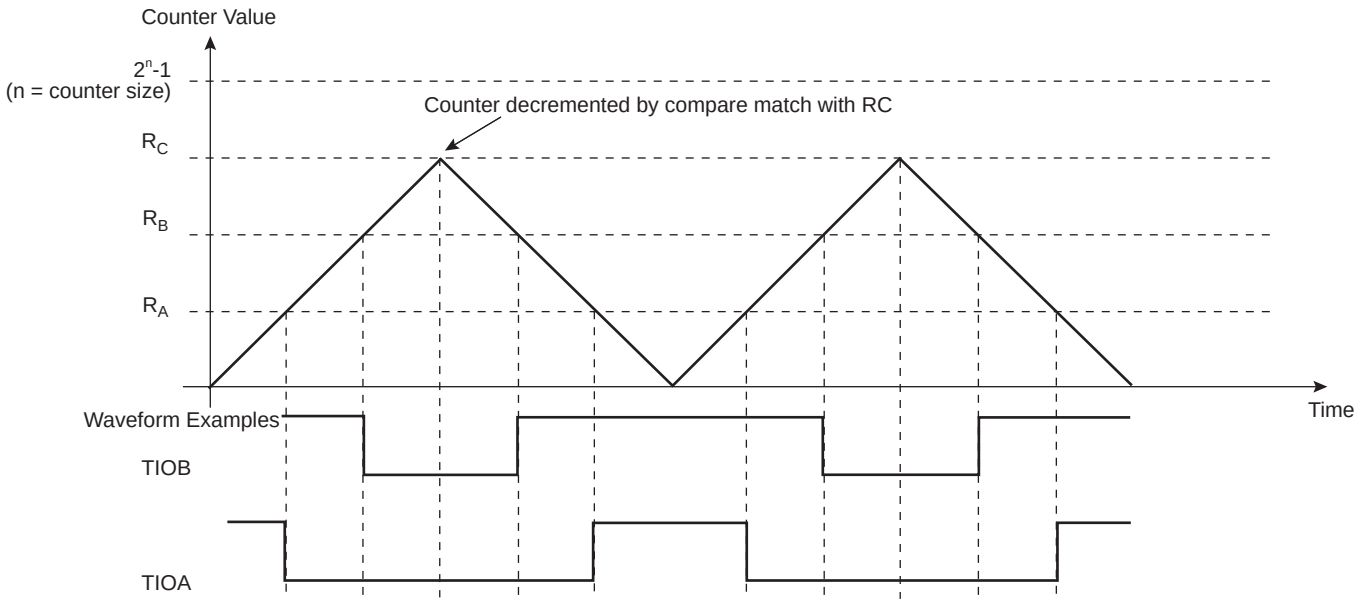
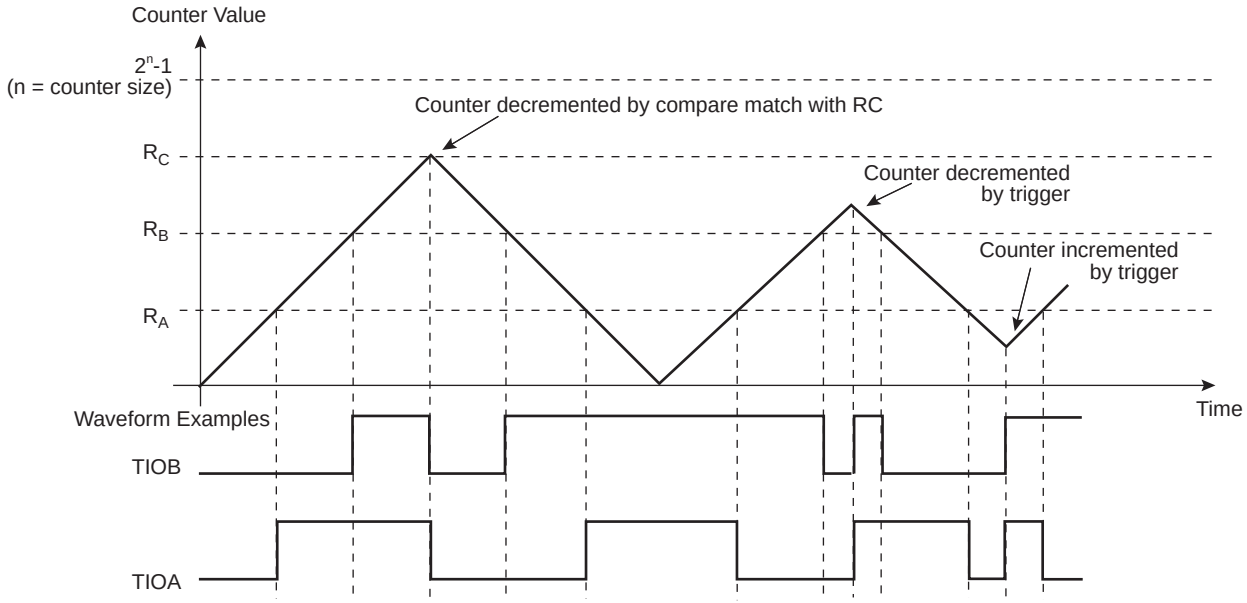


Figure 36-14. WAVSEL = 11 with Trigger



36.6.12 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 the 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.

36.6.13 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.

36.6.14 Quadrature Decoder

36.6.14.1 Description

The quadrature decoder (QDEC) 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 36-15).

When writing a 0 to bit QDEN of the TC_BMR, the QDEC is bypassed and the IO pins are directly routed to the timer counter function. See

TIOA0 and TIOB0 are to be driven by the two 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.

Field TCCLKS of TC_CMRx must be configured to select XC0 input (i.e., 0x101). Field TC0XC0S has no effect as soon as the QDEC 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.

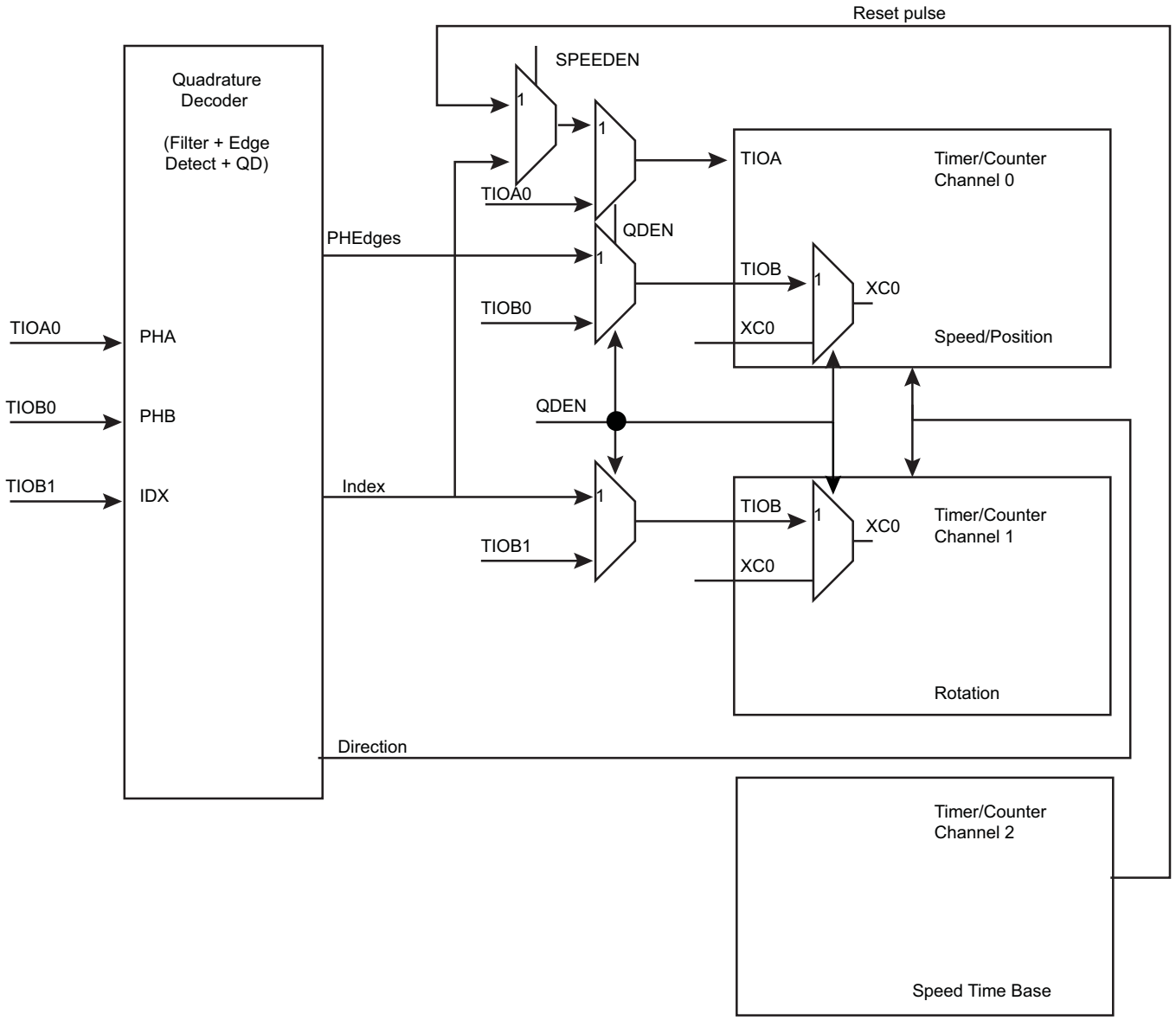
In Speed mode, position cannot be measured but revolution can be measured.

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) 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_SRx.

Figure 36-15. Predefined Connection of the Quadrature Decoder with Timer Counters



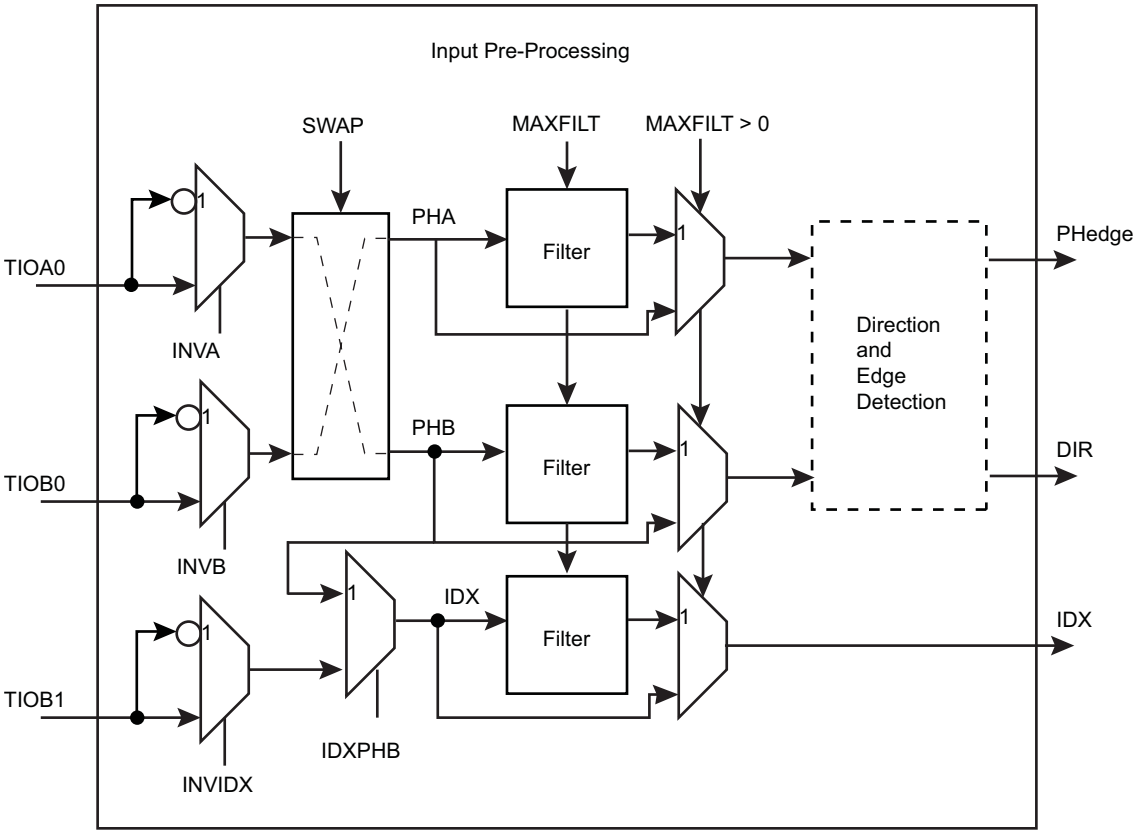
36.6.14.2 Input Pre-processing

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 swapping PHA, PHB is also configurable.

The MAXFILT field in the TC_BMR is used 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 $\times$ $t_{peripheral\ clock}$ ns are not passed to down-stream logic.

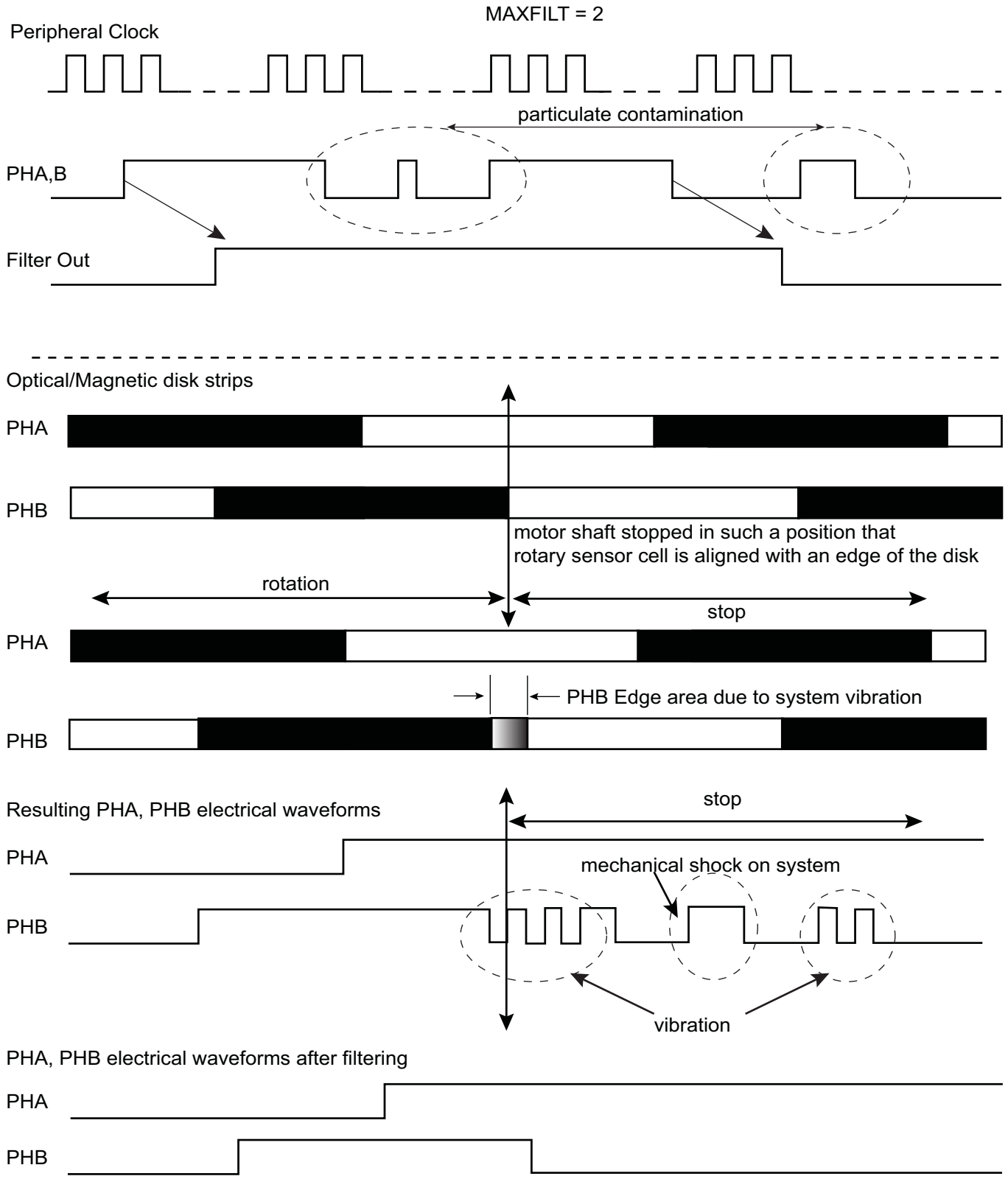
Figure 36-16. Input Stage



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 36-17. Filtering Examples



36.6.14.3 Direction Status and Change Detection

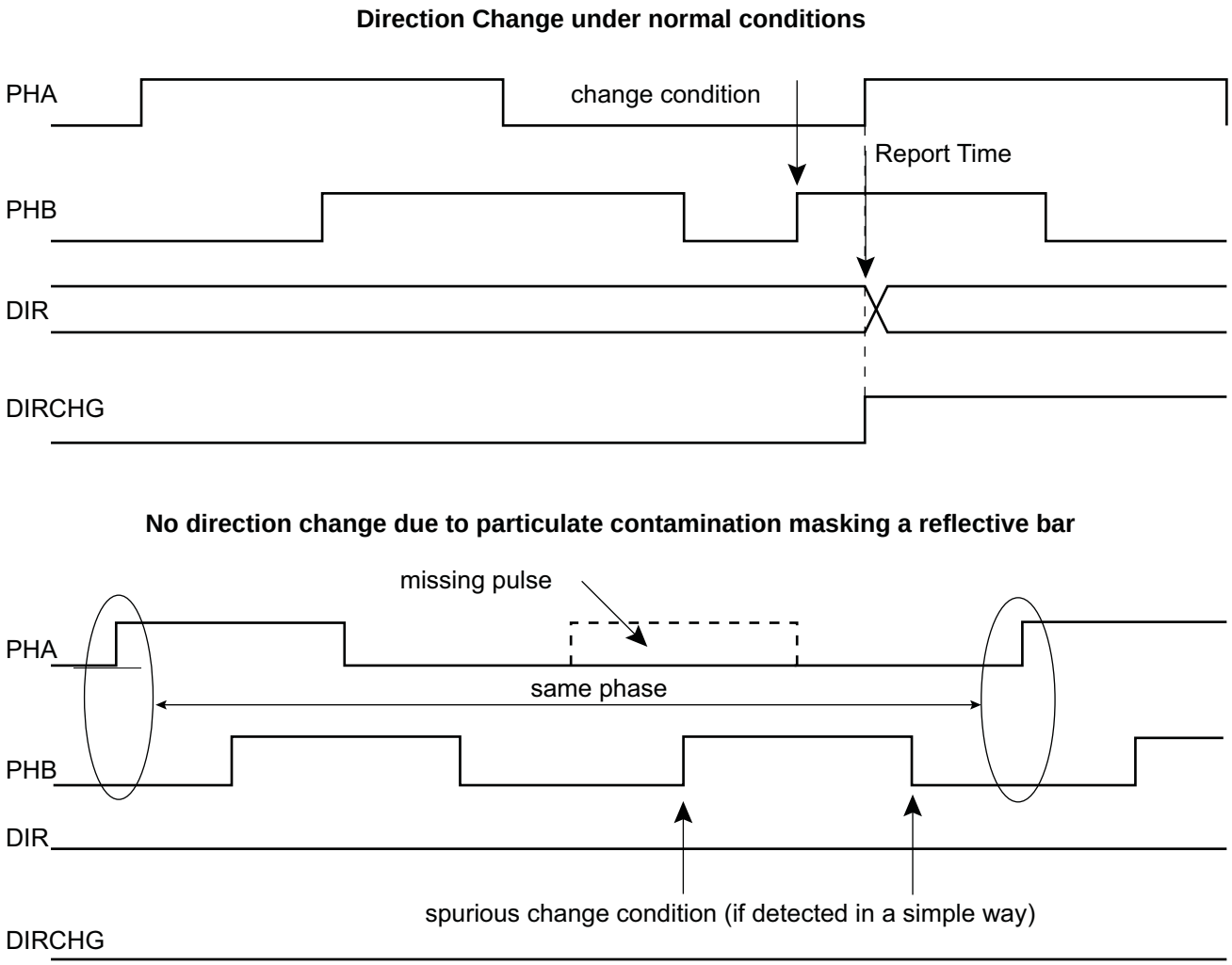
After filtering, the quadrature signals are analyzed to extract the rotation direction and edges of the two quadrature signals detected in order to be counted by timer/counter logic downstream.

The direction status can be directly read at anytime in the TC_QISR. The polarity of the direction flag status depends on the configuration written in TC_BMR. INVA, INVB, INVIDX, SWAP modify the polarity of DIR flag.

Any change in rotation direction is reported in the TC_QISR and can generate an interrupt.

The direction change condition is reported as soon as two 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 two consecutive edges of one 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 bars on the optical or magnetic disk of the sensor. Refer to Figure 36-18 for waveforms.

Figure 36-18. Rotation Change Detection

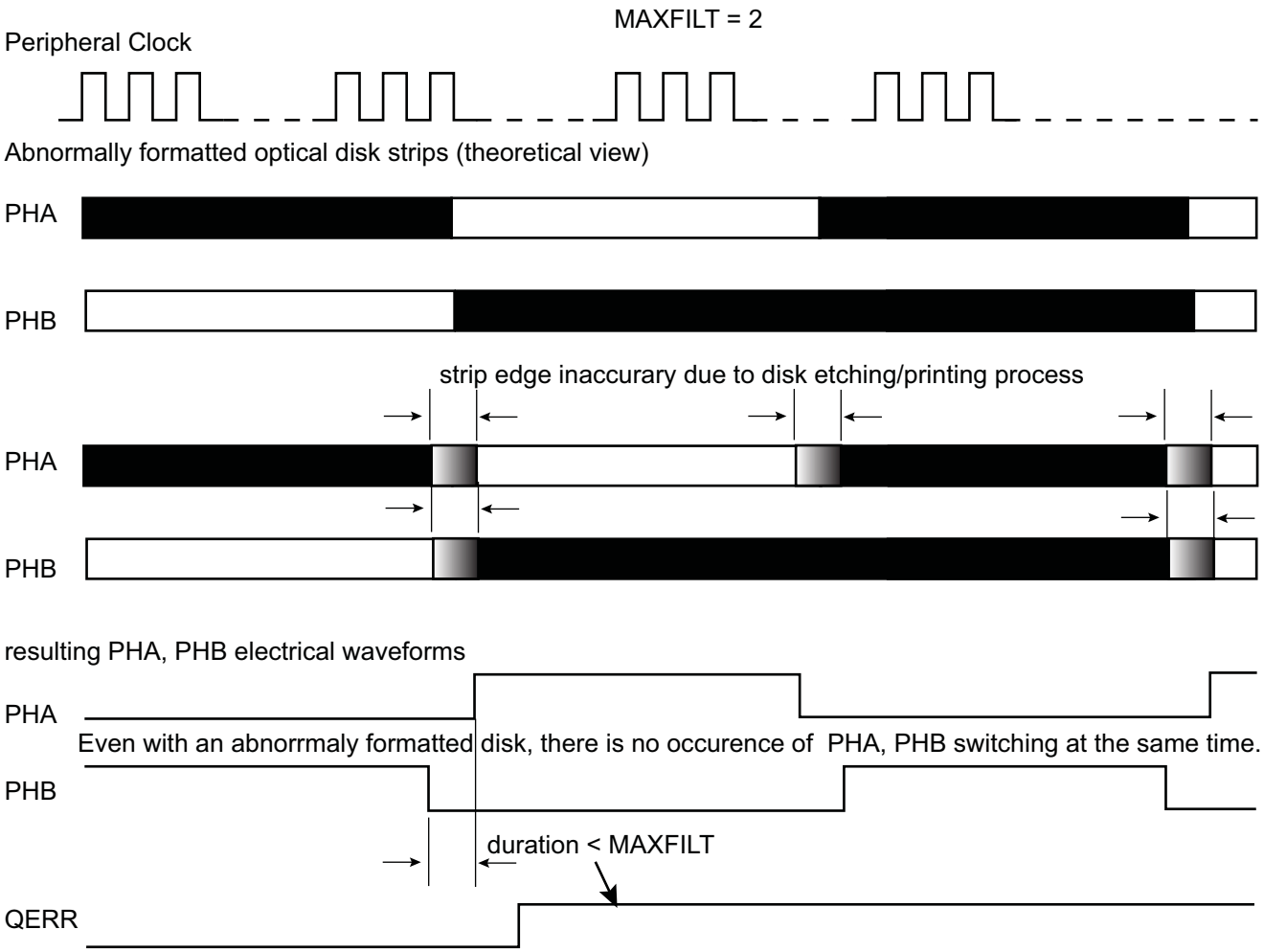


The direction change detection is disabled when QDTRANS is set in the TC_BMR. In this case, the DIR flag report must not be used.

A quadrature error is also reported by the QDEC via the QERR flag in the TC_QISR. This error is reported if the time difference between two edges on PHA, PHB is lower than a predefined value. This predefined value is

configurable and corresponds to (MAXFILT + 1) × $t_{peripheral\ clock}$ ns. After being filtered there is no reason to have two edges closer than (MAXFILT + 1) × $t_{peripheral\ clock}$ ns under normal mode of operation.

Figure 36-19. Quadrature Error Detection



MAXFILT must be tuned according to several factors such as the peripheral clock frequency, type of rotary sensor and rotation speed to be achieved.

36.6.14.4 Position and Rotation Measurement

When the POSEN bit is set in the TC_BMR, the motor axis position is processed on channel 0 (by means of the PHA, PHB edge detections) and the number of motor revolutions are recorded on channel 1 if the IDX signal is provided on the TIOB1 input. The position measurement can be read in the TC_CV0 register and the rotation measurement can be read in the TC_CV1 register.

Channel 0 and 1 must be configured in Capture mode (TC_CMR0.WAVE = 0). ‘Rising edge’ must be selected as the External Trigger Edge (TC_CMR.ETRGEDG = 0x01) and ‘TIOA’ must be selected as the External Trigger (TC_CMR.ABETRG = 0x1).

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.

36.6.14.5 Speed Measurement

When SPEEDEN is set in the TC_BMR, 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 set) in TC_CMR2. The WAVSEL field must be defined with 0x10 to clear the counter by comparison and matching with TC_RC value. Field ACPC 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_CMR0). The ABETRG bit of TC_CMR0 must be configured at 1 to select TIOA as a trigger for this channel.

EDGTRG must be set to 0x01, to clear the counter on a rising edge of the TIOA signal and field LDRA 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 bits CLKEN and SWTRG in the TC_CCR.

The speed can be read on field RA in TC_RA0.

Channel 1 can still be used to count the number of revolutions of the motor.

36.6.15 2-bit Gray Up/Down Counter for Stepper Motor

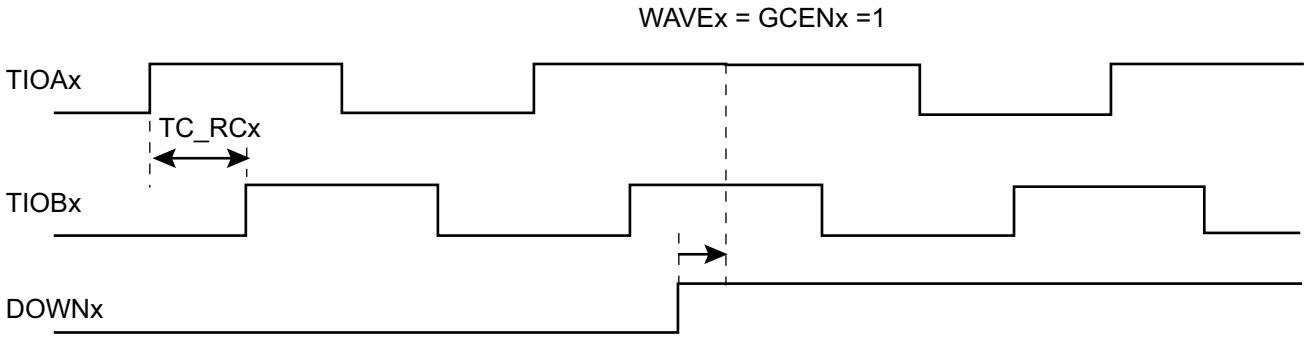
Each channel can be independently configured to generate a 2-bit gray count waveform on corresponding TIOA, TIOB outputs by means of the GCEN bit in TC_SMMRx.

Up or Down count can be defined by writing bit DOWN in TC_SMMRx.

It is mandatory to configure the channel in Waveform mode in the TC_CMR.

The period of the counters can be programmed in TC_RCx.

Figure 36-20. 2-bit Gray Up/Down Counter



36.6.16 Fault Mode

At any time, 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.

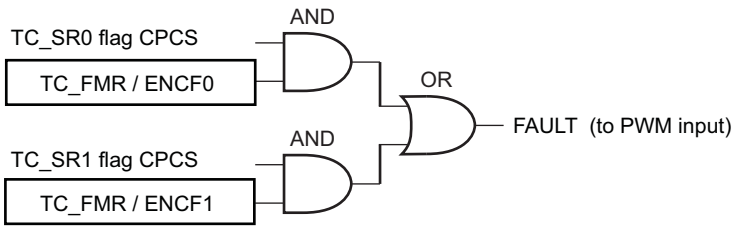
The CPCSx flags can be set accordingly and an interrupt can be generated.

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 and/or CPCS from TC_SR1. Each source can be independently enabled/disabled in the TC_FMR.

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 36-21. Fault Output Generation



36.6.17 Register Write Protection

To prevent any single software error from corrupting TC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the TC Write Protection Mode Register (TC_WPMR).

The Timer Counter clock of the first channel must be enabled to access TC_WPMR.

The following registers can be write-protected:

- 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

36.7 Timer Counter (TC) User Interface

Table 36-6. Register Mapping

Offset[1]	Register	Name	Access	Reset
0x00 + channel * 0x40 + 0x00	Channel Control Register	TC_CCR	Write-only	–
0x00 + channel * 0x40 + 0x04	Channel Mode Register	TC_CMR	Read/Write	0
0x00 + channel * 0x40 + 0x08	Stepper Motor Mode Register	TC_SMMR	Read/Write	0
0x00 + channel * 0x40 + 0x0C	Reserved	–	–	–
0x00 + channel * 0x40 + 0x10	Counter Value	TC_CV	Read-only	0
0x00 + channel * 0x40 + 0x14	Register A	TC_RA	Read/Write[2]	0
0x00 + channel * 0x40 + 0x18	Register B	TC_RB	Read/Write[2]	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
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 Protection Mode Register	TC_WPMR	Read/Write	0
0xE8–0xFC	Reserved	–	–	–

Notes: 1. Channel index ranges from 0 to 2.
2. Read-only if TC_CMRx.WAVE = 0

36.7.1 TC Channel Control Register

Name: TC_CCRx [x=0..2]

Address: 0x40080000 (0)[0], 0x40080040 (0)[1], 0x40080080 (0)[2], 0x40084000 (1)[0], 0x40084040 (1)[1], 0x40084080 (1)[2], 0x40088000 (2)[0], 0x40088040 (2)[1], 0x40088080 (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.

36.7.2 TC Channel Mode Register: Capture Mode

Name: TC_CMRx [x=0..2] (CAPTURE_MODE)

Address: 0x40080004 (0)[0], 0x40080044 (0)[1], 0x40080084 (0)[2], 0x40084004 (1)[0], 0x40084044 (1)[1], 0x40084084 (1)[2], 0x40088004 (2)[0], 0x40088044 (2)[1], 0x40088084 (2)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	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 the TC Write Protection Mode Register.

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal MCK/2 clock signal (from PMC)
1	TIMER_CLOCK2	Clock selected: internal MCK/8 clock signal (from PMC)
2	TIMER_CLOCK3	Clock selected: internal MCK/32 clock signal (from PMC)
3	TIMER_CLOCK4	Clock selected: internal MCK/128 clock signal (from PMC)
4	TIMER_CLOCK5	Clock selected: internal SLCK clock signal (from PMC)
5	XC0	Clock selected: XC0
6	XC1	Clock selected: XC1
7	XC2	Clock selected: XC2

- **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

36.7.3 TC Channel Mode Register: Waveform Mode

Name: TC_CMRx [x=0..2] (WAVEFORM_MODE)

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 the TC Write Protection Mode Register.

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal MCK/2 clock signal (from PMC)
1	TIMER_CLOCK2	Clock selected: internal MCK/8 clock signal (from PMC)
2	TIMER_CLOCK3	Clock selected: internal MCK/32 clock signal (from PMC)
3	TIMER_CLOCK4	Clock selected: internal MCK/128 clock signal (from PMC)
4	TIMER_CLOCK5	Clock selected: internal SLCK clock signal (from PMC)
5	XC0	Clock selected: XC0
6	XC1	Clock selected: XC1
7	XC2	Clock selected: XC2

- **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(1)	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

36.7.4 TC Stepper Motor Mode Register

Name: TC_SMMRx [x=0..2]

Address: 0x40080008 (0)[0], 0x40080048 (0)[1], 0x40080088 (0)[2], 0x40084008 (1)[0], 0x40084048 (1)[1], 0x40084088 (1)[2], 0x40088008 (2)[0], 0x40088048 (2)[1], 0x40088088 (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 the TC Write Protection Mode Register.

- **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.

36.7.5 TC Counter Value Register

Name: TC_CVx [x=0..2]

Address: 0x40080010 (0)[0], 0x40080050 (0)[1], 0x40080090 (0)[2], 0x40084010 (1)[0], 0x40084050 (1)[1], 0x40084090 (1)[2], 0x40088010 (2)[0], 0x40088050 (2)[1], 0x40088090 (2)[2]

Access: Read-only

31	30	29	28	27	26	25	24
CV							

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							

- **CV: Counter Value**

CV contains the counter value in real time.

36.7.6 TC Register A

Name: TC_RAx [x=0..2]

Address: 0x40080014 (0)[0], 0x40080054 (0)[1], 0x40080094 (0)[2], 0x40084014 (1)[0], 0x40084054 (1)[1], 0x40084094 (1)[2], 0x40088014 (2)[0], 0x40088054 (2)[1], 0x40088094 (2)[2]

Access: Read-only if TC_CMRx.WAVE = 0, Read/Write if TC_CMRx.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 the TC Write Protection Mode Register.

- **RA: Register A**

RA contains the Register A value in real time.

36.7.7 TC Register B

Name: TC_RBx [x=0..2]

Address: 0x40080018 (0)[0], 0x40080058 (0)[1], 0x40080098 (0)[2], 0x40084018 (1)[0], 0x40084058 (1)[1], 0x40084098 (1)[2], 0x40088018 (2)[0], 0x40088058 (2)[1], 0x40088098 (2)[2]

Access: Read-only if TC_CMRx.WAVE = 0, Read/Write if TC_CMRx.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 the TC Write Protection Mode Register.

- **RB: Register B**

RB contains the Register B value in real time.

36.7.8 TC Register C

Name: TC_RCx [x=0..2]

Address: 0x4008001C (0)[0], 0x4008005C (0)[1], 0x4008009C (0)[2], 0x4008401C (1)[0], 0x4008405C (1)[1], 0x4008409C (1)[2], 0x4008801C (2)[0], 0x4008805C (2)[1], 0x4008809C (2)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
RC							

23	22	21	20	19	18	17	16
RC							

15	14	13	12	11	10	9	8
RC							

7	6	5	4	3	2	1	0
RC							

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RC: Register C**

RC contains the Register C value in real time.

36.7.9 TC Status Register

Name: TC_SRx [x=0..2]

Address: 0x40080020 (0)[0], 0x40080060 (0)[1], 0x400800A0 (0)[2], 0x40084020 (1)[0], 0x40084060 (1)[1], 0x400840A0 (1)[2], 0x40088020 (2)[0], 0x40088060 (2)[1], 0x400880A0 (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
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow Status (cleared on read)**

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 (cleared on read)**

0: Load overrun has not occurred since the last read of the Status Register or TC_CMRx.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 TC_CMRx.WAVE = 0.

- **CPAS: RA Compare Status (cleared on read)**

0: RA Compare has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 0.

1: RA Compare has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 1.

- **CPBS: RB Compare Status (cleared on read)**

0: RB Compare has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 0.

1: RB Compare has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 1.

- **CPCS: RC Compare Status (cleared on read)**

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 (cleared on read)**

0: RA Load has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1: RA Load has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **LDRBS: RB Loading Status (cleared on read)**

0: RB Load has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1: RB Load has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **ETRGS: External Trigger Status (cleared on read)**

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.

- **CLKSTA: Clock Enabling Status**

0: Clock is disabled.

1: Clock is enabled.

- **MTIOA: TIOA Mirror**

0: TIOA is low. If TC_CMRx.WAVE = 0, this means that TIOA pin is low. If TC_CMRx.WAVE = 1, this means that TIOA is driven low.

1: TIOA is high. If TC_CMRx.WAVE = 0, this means that TIOA pin is high. If TC_CMRx.WAVE = 1, this means that TIOA is driven high.

- **MTIOB: TIOB Mirror**

0: TIOB is low. If TC_CMRx.WAVE = 0, this means that TIOB pin is low. If TC_CMRx.WAVE = 1, this means that TIOB is driven low.

1: TIOB is high. If TC_CMRx.WAVE = 0, this means that TIOB pin is high. If TC_CMRx.WAVE = 1, this means that TIOB is driven high.

36.7.10 TC Interrupt Enable Register

Name: TC_IERx [x=0..2]

Address: 0x40080024 (0)[0], 0x40080064 (0)[1], 0x400800A4 (0)[2], 0x40084024 (1)[0], 0x40084064 (1)[1], 0x400840A4 (1)[2], 0x40088024 (2)[0], 0x40088064 (2)[1], 0x400880A4 (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
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.

36.7.11 TC Interrupt Disable Register

Name: TC_IDRx [x=0..2]

Address: 0x40080028 (0)[0], 0x40080068 (0)[1], 0x400800A8 (0)[2], 0x40084028 (1)[0], 0x40084068 (1)[1], 0x400840A8 (1)[2], 0x40088028 (2)[0], 0x40088068 (2)[1], 0x400880A8 (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
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 TC_CMRx.WAVE = 0).

- **CPAS: RA Compare**

0: No effect.

1: Disables the RA Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPBS: RB Compare**

0: No effect.

1: Disables the RB Compare Interrupt (if TC_CMRx.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 TC_CMRx.WAVE = 0).

- **LDRBS: RB Loading**

0: No effect.

1: Disables the RB Load Interrupt (if TC_CMRx.WAVE = 0).

- **ETRGS: External Trigger**

0: No effect.

1: Disables the External Trigger Interrupt.

36.7.12 TC Interrupt Mask Register

Name: TC_IMRx [x=0..2]

Address: 0x4008002C (0)[0], 0x4008006C (0)[1], 0x400800AC (0)[2], 0x4008402C (1)[0], 0x4008406C (1)[1], 0x400840AC (1)[2], 0x4008802C (2)[0], 0x4008806C (2)[1], 0x400880AC (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
–	–	–	–	–	–	–	–
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.

36.7.13 TC Block Control Register

Name: TC_BCR

Address: 0x400800C0 (0), 0x400840C0 (1), 0x400880C0 (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.

36.7.14 TC Block Mode Register

Name: TC_BMR

Address: 0x400800C4 (0), 0x400840C4 (1), 0x400880C4 (2)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	MAXFILT	

23	22	21	20	19	18	17	16
MAXFILT				–	–	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 the TC Write Protection Mode Register.

- **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	TIOA0	Signal connected to XC2: TIOA0
3	TIOA1	Signal connected to XC2: TIOA1

- **QDEN: Quadrature Decoder Enabled**

0: Disabled.

1: Enables the QDEC (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 the QDEC.

1: PHA is inverted before driving the QDEC.

- **INVB: Inverted PHB**

0: PHB (TIOB0) is directly driving the QDEC.

1: PHB is inverted before driving the QDEC.

- **INVIDX: Inverted Index**

0: IDX (TIOA1) is directly driving the QDEC.

1: IDX is inverted before driving the QDEC.

- **SWAP: Swap PHA and PHB**

0: No swap between PHA and PHB.

1: Swap PHA and PHB internally, prior to driving the QDEC.

- **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.

- **MAXFILT: Maximum Filter**

1–63: Defines the filtering capabilities.

Pulses with a period shorter than MAXFILT+1 peripheral clock cycles are discarded.

36.7.15 TC QDEC Interrupt Enable Register

Name: TC_QIER

Address: 0x400800C8 (0), 0x400840C8 (1), 0x400880C8 (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.

36.7.16 TC QDEC Interrupt Disable Register

Name: TC_QIDR

Address: 0x400800CC (0), 0x400840CC (1), 0x400880CC (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.

36.7.17 TC QDEC Interrupt Mask Register

Name: TC_QIMR

Address: 0x400800D0 (0), 0x400840D0 (1), 0x400880D0 (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.

36.7.18 TC QDEC Interrupt Status Register

Name: TC_QISR

Address: 0x400800D4 (0), 0x400840D4 (1), 0x400880D4 (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.

36.7.19 TC Fault Mode Register

Name: TC_FMR

Address: 0x400800D8 (0), 0x400840D8 (1), 0x400880D8 (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 the TC Write Protection Mode Register.

- **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.

36.7.20 TC Write Protection Mode Register

Name: TC_WPMR

Address: 0x400800E4 (0), 0x400840E4 (1), 0x400880E4 (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 Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x54494D (“TIM” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x54494D (“TIM” in ASCII).

The Timer Counter clock of the first channel must be enabled to access this register.

See Section 36.6.17 ”Register Write Protection”, for a list of registers that can be write-protected and Timer Counter clock conditions.

- **WPKEY: Write Protection 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.

37. High Speed MultiMedia Card Interface (HSMCI)

37.1 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 DMA Controller (DMAC), 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 2 slot(s). Each slot may be used to interface with a High Speed MultiMediaCard 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.

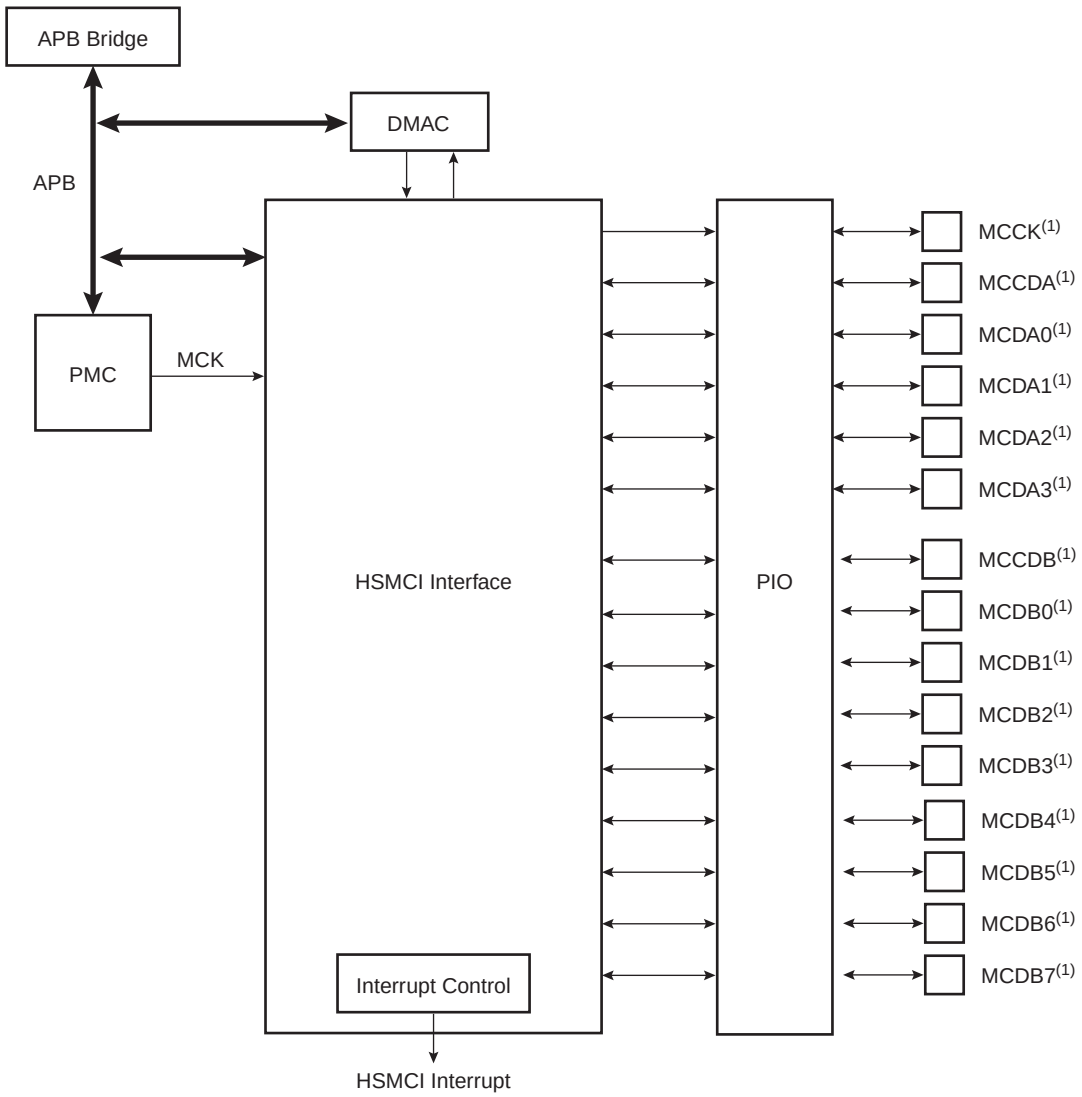
HSMCI fully supports CE-ATA Revision 1.1, built on the MMC System Specification v4.0. The module includes dedicated hardware to issue the command completion signal and capture the host command completion signal disable.

37.2 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 2 Multiplexed Slot(s)
  - Each Slot for either a High Speed MultiMediaCard Bus (Up to 30 Cards) or an SD Memory Card
- Support for Stream, Block and Multi-block Data Read and Write
- Supports Connection to DMA Controller (DMAC)
  - 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

37.3 Block Diagram

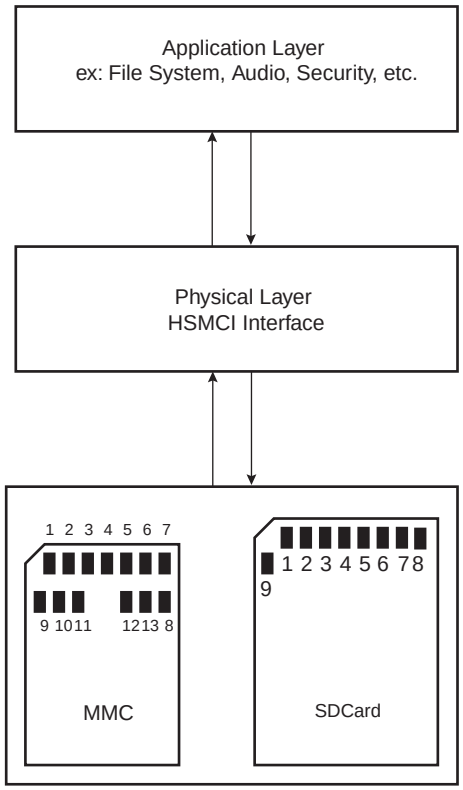
Figure 37-1. Block Diagram



Note: 1. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA, MCCDB to HSMCIx_CDB, MCDAy to HSMCIx_DAy, MCDBy to HSMCIx_DBy.

37.4 Application Block Diagram

Figure 37-2. Application Block Diagram



37.5 Pin Name List

Table 37-1. I/O Lines Description for 8-bit Configuration

Pin Name[2]	Pin Description	Type[1]	Comments
MCCDA/MCCDB	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 - MCDA7	Data 0..7 of Slot A	I/O/PP	DAT[0..7] of an MMC DAT[0..3] of an SD Card/SDIO
MCDB0 - MCDB7	Data 0..7 of Slot B	I/O/PP	DAT[0..7] of an MMC DAT[0..3] of an SD Card/SDIO
MCDC0 - MCDC7	Data 0..7 of Slot C	I/O/PP	DAT[0..7] of an MMC DAT[0..3] of an SD Card/SDIO
MCDD0 - MCDD7	Data 0..7 of Slot D	I/O/PP	DAT[0..7] 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, MCCDB to HSMCIx_CDB, MCCDC to HSMCIx_CDC, MCCDD to HSMCIx_CDD, MCDAy to HSMCIx_DAy, MCDBy to HSMCIx_DBy, MCDCy to HSMCIx_DCy, MCDDy to HSMCIx_DDy.

,.

37.6 Product Dependencies

37.6.1 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 37-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
HSMCI	MCCDA	PA20	A
HSMCI	MCCDB	PE20	B
HSMCI	MCCK	PA19	A
HSMCI	MCDA0	PA21	A
HSMCI	MCDA1	PA22	A
HSMCI	MCDA2	PA23	A
HSMCI	MCDA3	PA24	A
HSMCI	MCDA4	PD0	B
HSMCI	MCDA5	PD1	B
HSMCI	MCDA6	PD2	B
HSMCI	MCDA7	PD3	B
HSMCI	MCDB0	PE22	B
HSMCI	MCDB1	PE24	B
HSMCI	MCDB2	PE26	B
HSMCI	MCDB3	PE27	B

37.6.2 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.

37.6.3 Interrupt

The HSMCI interface has an interrupt line connected to the Nested Vector Interrupt Controller (NVIC).

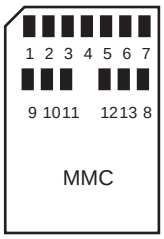
Handling the HSMCI interrupt requires programming the NVIC before configuring the HSMCI.

Table 37-3. Peripheral IDs

Instance	ID
HSMCI	21

37.7 Bus Topology

Figure 37-3. 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.

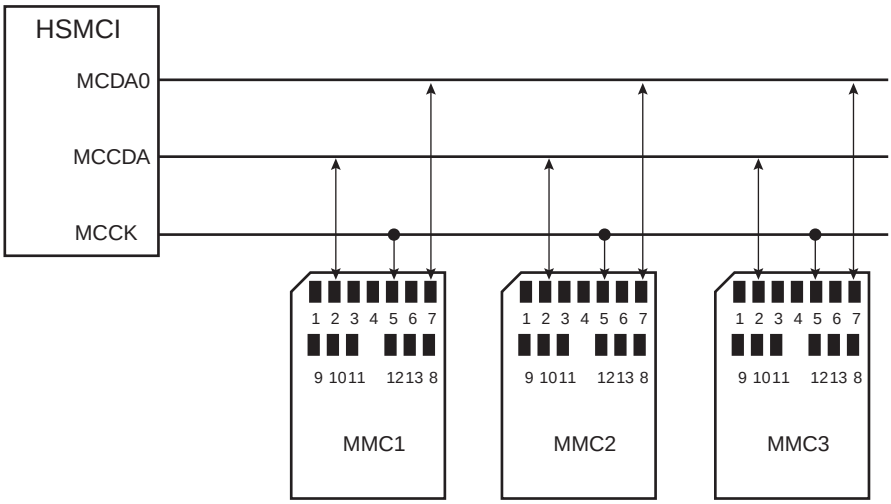
Table 37-4. Bus Topology

Pin Number	Name	Type[1]	Description	HSMCI Pin Name[2] (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.

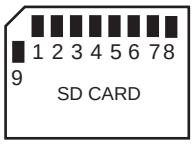
2. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA, MCCDB to HSMCIx_CDB, MCDAy to HSMCIx_DAy, MCDBy to HSMCIx_DBy.

Figure 37-4. 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 37-5. SD Memory Card Bus Topology



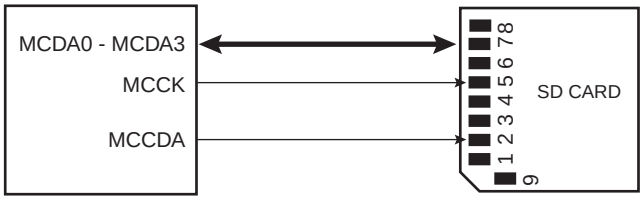
The SD Memory Card bus includes the signals listed in Table 37-5.

Table 37-5. SD Memory Card Bus Signals

Pin Number	Name	Type[1]	Description	HSMCI Pin Name[2] (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

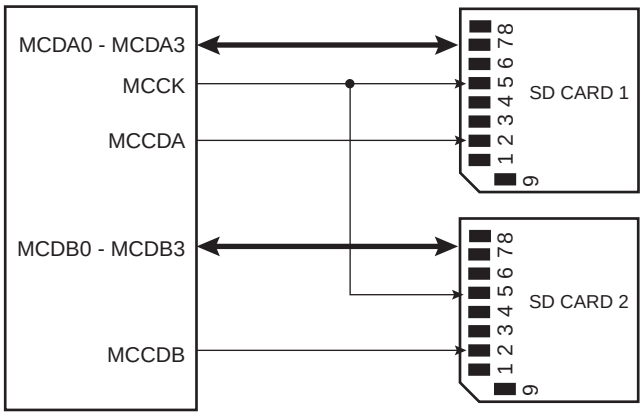
Notes: 1. I: input, O: output, PP: Push Pull, OD: Open Drain.
2. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA, MCCDB to HSMCIx_CDB, MCDAy to HSMCIx_DAy, MCDBy to HSMCIx_DBy.

Figure 37-6. SD Card Bus Connections with One Slot



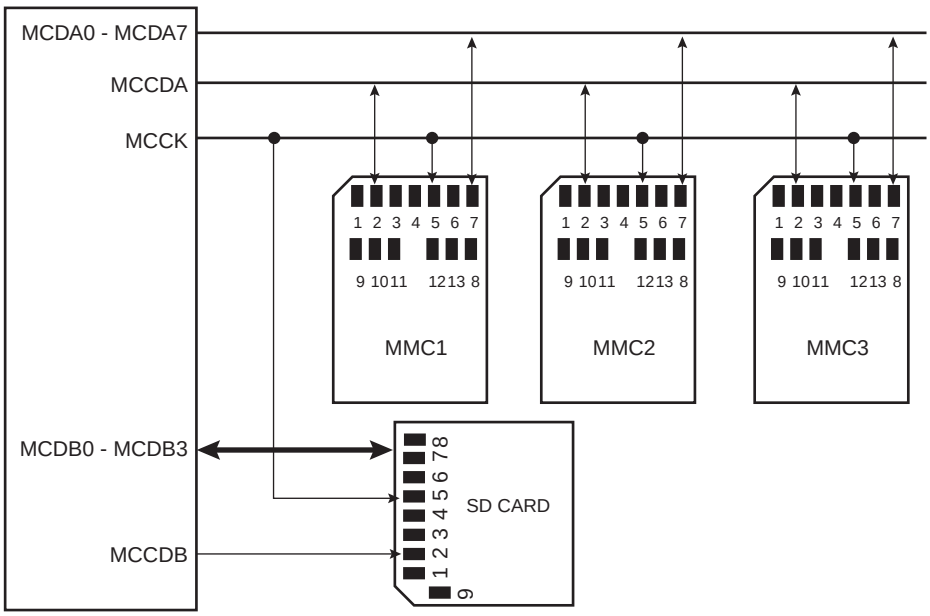
Note: When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA MCDAy to HSMCIx_DAy.

Figure 37-7. SD Card Bus Connections with Two Slots



Note: When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK,MCCDA to HSMCIx_CDA, MCDAy to HSMCIx_DAy, MCCDB to HSMCIx_CDB, MCDBy to HSMCIx_DBy.

Figure 37-8. Mixing High Speed MultiMedia and SD Memory Cards with Two Slots



Note: When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA, MCDAy to HSMCIx_DAy, MCCDB to HSMCIx_CDB, MCDBy to HSMCIx_DBy.

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.

37.8 High Speed MultiMediaCard Operations

After a power-on reset, the cards are initialized by a special message-based High Speed MultiMediaCard 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 37-6 on page 917.

High Speed MultiMediaCard 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 919.).

The HSMCI provides a set of registers to perform the entire range of High Speed MultiMedia Card operations.

37.8.1 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 MultiMediaCard 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:

	Host Command					N_{ID} Cycles					CID			
CMD	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 37-6 and Table 37-7.

Table 37-6. ALL_SEND_CID Command Description

CMD Index	Type	Argument	Resp	Abbreviation	Command Description
CMD2	bcr	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the CMD line

Note: bcr means broadcast command with response.

Table 37-7. 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 37-7).

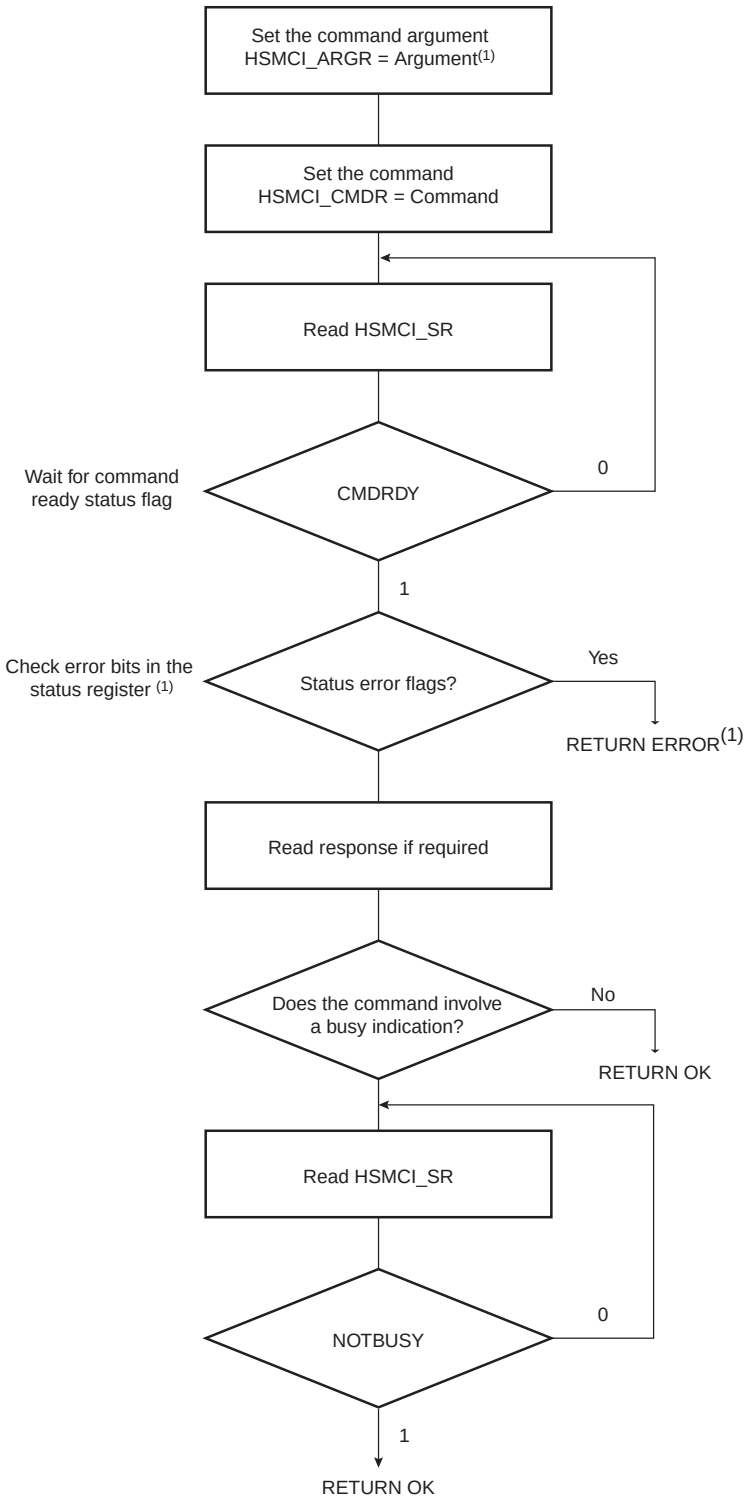
The command is sent immediately after writing the command register.

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.

Table 37-8. 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).

37.8.2 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 DMA Controller.

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.

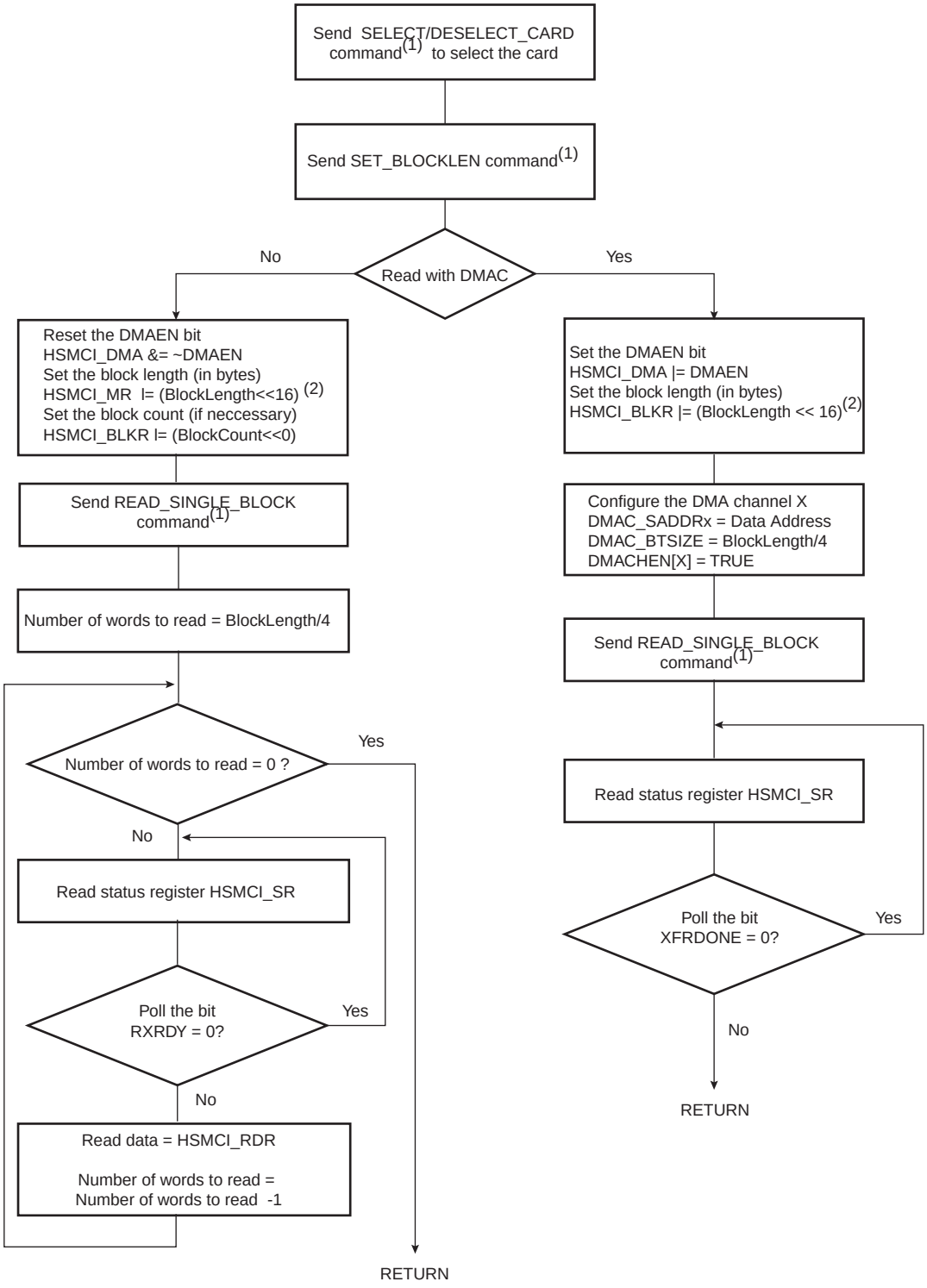
- Multiple block read (or write) with pre-defined block count (since version 3.1 and higher):

  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.

37.8.3 Read Operation

The following flowchart (Figure 37-9) shows how to read a single block with or without use of DMAC facilities. In this example, 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 37-9. Read Functional Flow Diagram



Notes:
1. It is assumed that this command has been correctly sent (see Figure 37-8).
2. This field is also accessible in the HSMCI Block Register (HSMCI_BLKR).

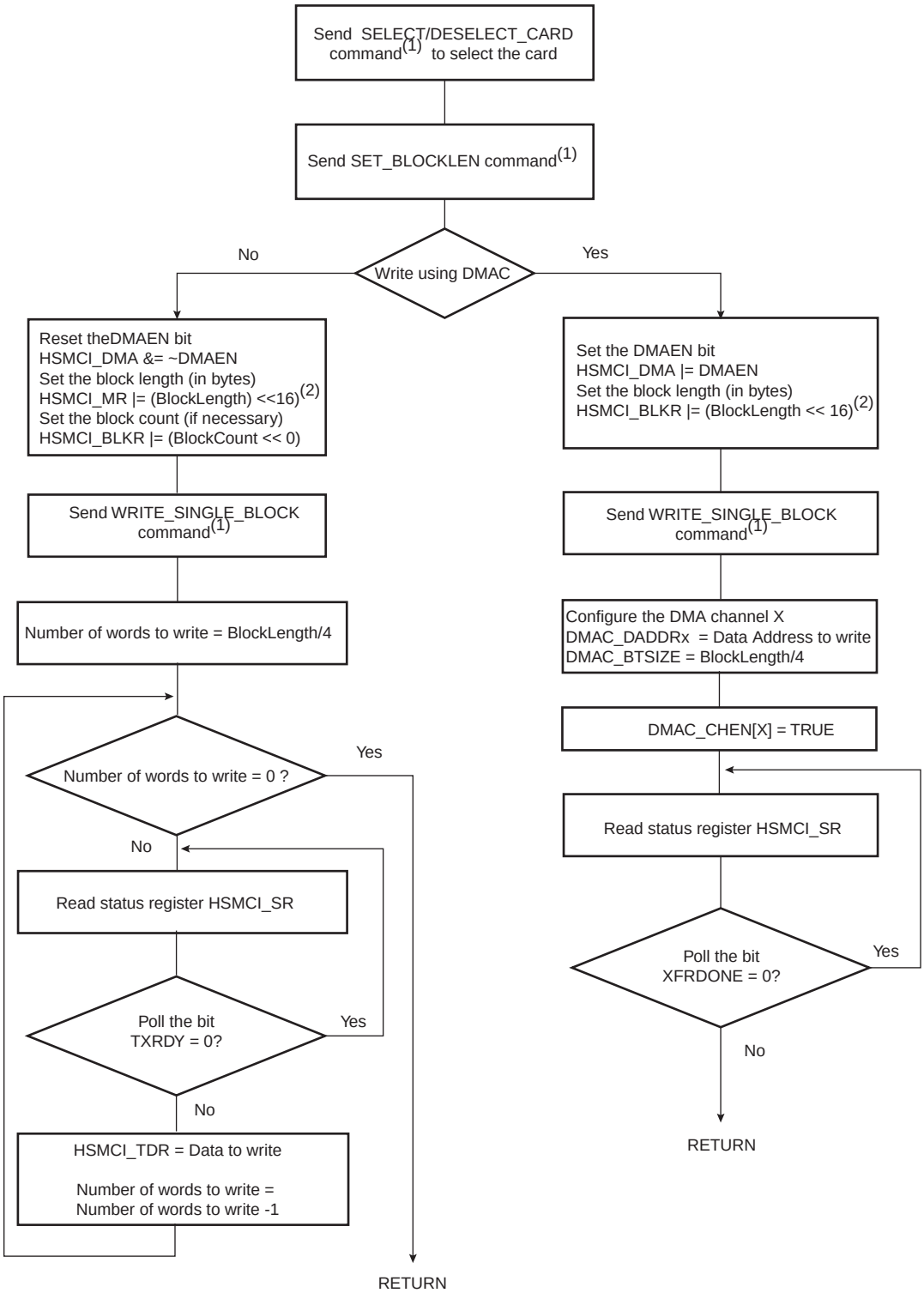
37.8.4 Write Operation

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 PADV is 0, then 0x00 value is used when padding data, otherwise 0xFF is used.

If set, the bit DMAEN in the HSMCI_DMA register enables DMA transfer.

The following flowchart (Figure 37-10) shows how to write a single block with or without use of DMA 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).

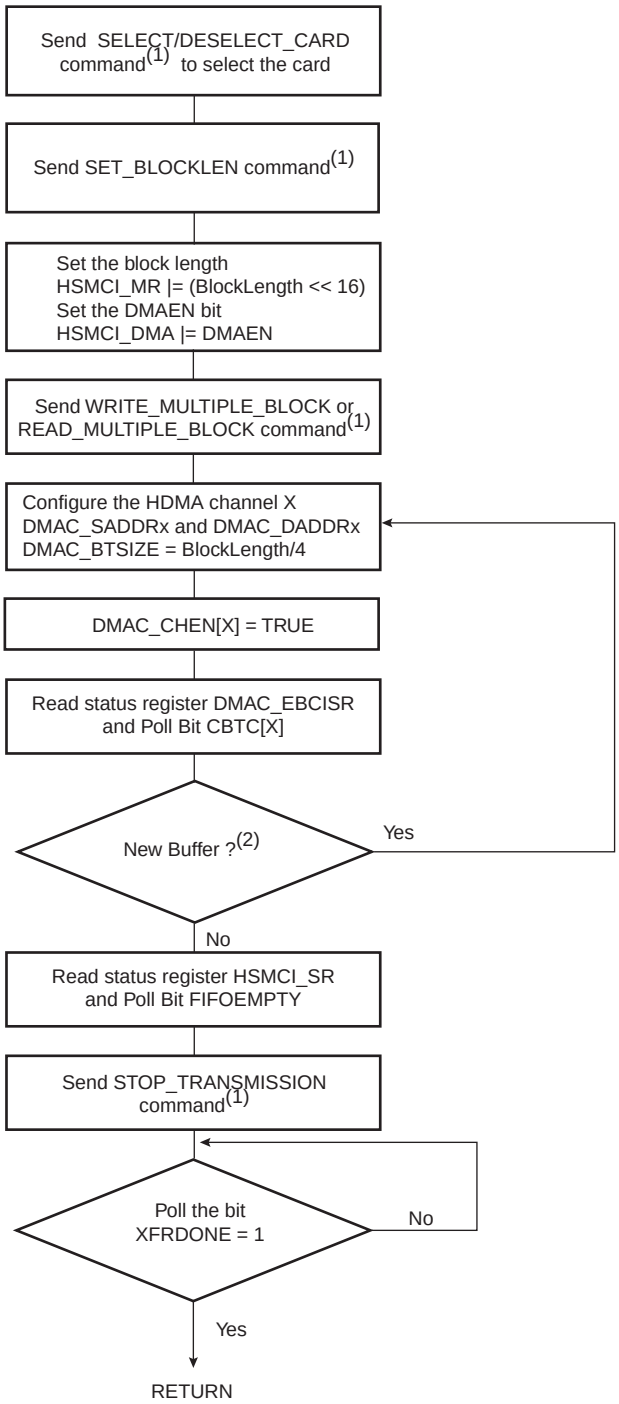
Figure 37-10. Write Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 37-8).
2. This field is also accessible in the HSMCI Block Register (HSMCI_BLKR).

The following flowchart (Figure 37-11) shows how to manage read multiple block and write multiple block transfers with the DMA Controller. 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 37-11. Read Multiple Block and Write Multiple Block



Notes: 1. It is assumed that this command has been correctly sent (see Figure 37-8).
2. Handle errors reported in HSMCI_SR.

37.8.5 WRITE_SINGLE_BLOCK Operation using DMA Controller

1. Wait until the current command execution has successfully terminated.
   c. Check that CMDRDY and NOTBUSY fields are asserted in HSMCI_SR
2. Program the block length in the card. This value defines the value *block_length.*
3. Program the block length in the HSMCI configuration register with *block_length* value.
4. Program HSMCI_DMA register with the following fields:
   - OFFSET field with *dma_offset.*
   - CHKSIZE is user defined and set according to DMAC_DCSIZE.
   - DMAEN is set to true to enable DMA hardware handshaking in the HSMCI. This bit was previously set to false.
5. Issue a WRITE_SINGLE_BLOCK command writing HSMCI_ARG then HSMCI_CMDR.
6. Program the DMA Controller.
   a. Read the channel Register to choose an available (disabled) channel.
   b. Clear any pending interrupts on the channel from the previous DMAC transfer by reading the DMAC_EBCISR register.
   c. Program the channel registers.
   d. The DMAC_SADDRx register for channel x must be set to the location of the source data. When the first data location is not word aligned, the two LSB bits define the temporary value called *dma_offset.* The two LSB bits of DMAC_SADDRx must be set to 0.
   e. The DMAC_DADDRx register for channel x must be set with the starting address of the HSMCI_FIFO address.
   f. Program DMAC_CTRLAx register of channel x with the following field's values:
      –DST_WIDTH is set to WORD.
      –SRC_WIDTH is set to WORD.
      –DCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.
      –BTSIZE is programmed with *CEILING((block_length + dma_offset) / 4),* where the ceiling function is the function that returns the smallest integer not less than x.
   g. Program DMAC_CTRLBx register for channel x with the following field's values:
      –DST_INCR is set to INCR, the *block_length* value must not be larger than the HSMCI_FIFO aperture.
      –SRC_INCR is set to INCR.
      –FC field is programmed with memory to peripheral flow control mode.
      –both DST_DSCR and SRC_DSCR are set to 1 (descriptor fetch is disabled).
      –DIF and SIF are set with their respective layer ID. If SIF is different from DIF, the DMA controller is able to prefetch data and write HSMCI simultaneously.
   h. Program DMAC_CFGx register for channel x with the following field's values:
      –FIFOCFG defines the watermark of the DMAC channel FIFO.
      –DST_H2SEL is set to true to enable hardware handshaking on the destination.
      –DST_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.
   i. Enable Channel x, writing one to DMAC_CHER[x]. The DMAC is ready and waiting for request.
7. Wait for XFRDONE in HSMCI_SR register.

37.8.6 READ_SINGLE_BLOCK Operation using DMA Controller

37.8.6.1 Block Length is Multiple of 4

1. Wait until the current command execution has successfully completed.
   a. Check that CMDRDY and NOTBUSY are asserted in HSMCI_SR.
2. Program the block length in the card. This value defines the value *block_length.*
3. Program the block length in the HSMCI configuration register with *block_length* value.
4. Set RDPROOF bit in HSMCI_MR to avoid overflow.
5. Program HSMCI_DMA register with the following fields:
   - ROPT field is set to 0.
   - OFFSET field is set to 0.
   - CHKSIZE is user defined.
   - DMAEN is set to true to enable DMAC hardware handshaking in the HSMCI. This bit was previously set to false.
6. Issue a READ_SINGLE_BLOCK command.
7. Program the DMA controller.
   a. Read the channel Register to choose an available (disabled) channel.
   b. Clear any pending interrupts on the channel from the previous DMA transfer by reading the DMAC_EBCISR register.
   c. Program the channel registers.
   d. The DMAC_SADDRx register for channel x must be set with the starting address of the HSMCI_FIFO address.
   e. The DMAC_DADDRx register for channel x must be word aligned.
   f. Program DMAC_CTRLAx register of channel x with the following field's values:
      –DST_WIDTH is set to WORD.
      –SRC_WIDTH is set to WORD.
      –SCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.
      –BTSIZE is programmed with *block_length/4.*
   g. Program DMAC_CTRLBx register for channel x with the following field's values:
      - DST_INCR is set to INCR.
      - SRC_INCR is set to INCR.
      - FC field is programmed with peripheral to memory flow control mode.
      - both DST_DSCR and SRC_DSCR are set to 1 (descriptor fetch is disabled).
      - DIF and SIF are set with their respective layer ID. If SIF is different from DIF, the DMA controller is able to prefetch data and write HSMCI simultaneously.
   h. Program DMAC_CFGx register for channel x with the following field's values:
      –FIFOCFG defines the watermark of the DMA channel FIFO.
      –SRC_H2SEL is set to true to enable hardware handshaking on the destination.
      –SRC_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.
      –Enable Channel x, writing one to DMAC_CHER[x]. The DMAC is ready and waiting for request.
8. Wait for XFRDONE in HSMCI_SR register.

37.8.6.2 Block Length is Not Multiple of 4 and Padding Not Used (ROPT field in HSMCI_DMA register set to 0)

In the previous DMA transfer flow (block length multiple of 4), the DMA controller is configured to use only WORD AHB access. When the block length is no longer a multiple of 4 this is no longer true. The DMA controller is programmed to copy exactly the block length number of bytes using 2 transfer descriptors.

1. Use the previous step until READ_SINGLE_BLOCK then
2. Program the DMA controller to use a two descriptors linked list.
   a. Read the channel Register to choose an available (disabled) channel.
   b. Clear any pending interrupts on the channel from the previous DMA transfer by reading the DMAC_EBCISR register.
   c. Program the channel registers in the Memory for the first descriptor. This descriptor will be word oriented. This descriptor is referred to as LLI_W, standing for LLI word oriented transfer.
   d. The LLI_W.DMAC_SADDRx field in memory must be set with the starting address of the HSMCI_FIFO address.
   e. The LLI_W.DMAC_DADDRx field in the memory must be word aligned.
   f. Program LLI_W.DMAC_CTRLAx with the following field's values:
      –DST_WIDTH is set to WORD.
      –SRC_WIDTH is set to WORD.
      –SCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.
      –BTSIZE is programmed with *block_length/4.* If BTSIZE is zero, this descriptor is skipped later.
   g. Program LLI_W.DMAC_CTRLBx with the following field's values:
      –DST_INCR is set to INCR
      –SRC_INCR is set to INCR
      –FC field is programmed with peripheral to memory flow control mode.
      –SRC_DSCR is set to zero. (descriptor fetch is enabled for the SRC)
      –DST_DSCR is set to one. (descriptor fetch is disabled for the DST)
      –DIF and SIF are set with their respective layer ID. If SIF is different from DIF, DMA controller is able to prefetch data and write HSMCI simultaneously.
   h. Program LLI_W.DMAC_CFGx register for channel x with the following field's values:
      –FIFOCFG defines the watermark of the DMA channel FIFO.
      –DST_REP is set to zero meaning that address are contiguous.
      –SRC_H2SEL is set to true to enable hardware handshaking on the destination.
      –SRC_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.
   i. Program LLI_W.DMAC_DSCRx with the address of LLI_B descriptor. And set DSCRx_IF to the AHB Layer ID. This operation actually links the Word oriented descriptor on the second byte oriented descriptor. When *block_length[1:0]* is equal to 0 (multiple of 4) LLI_W.DMAC_DSCRx points to 0, only LLI_W is relevant.
   j. Program the channel registers in the Memory for the second descriptor. This descriptor will be byte oriented. This descriptor is referred to as LLI_B, standing for LLI Byte oriented.
   k. The LLI_B.DMAC_SADDRx field in memory must be set with the starting address of the HSMCI_FIFO address.
   l. The LLI_B.DMAC_DADDRx is not relevant if previous word aligned descriptor was enabled. If 1, 2 or 3 bytes are transferred that address is user defined and not word aligned.
   m. Program LLI_B.DMAC_CTRLAx with the following field's values:
      –DST_WIDTH is set to BYTE.

–SRC_WIDTH is set to BYTE.

–SCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.

–BTSIZE is programmed with *block_length[1:0].* (last 1, 2, or 3 bytes of the buffer).

n. Program LLI_B.DMAC_CTRLBx with the following field's values:

–DST_INCR is set to INCR

–SRC_INCR is set to INCR

–FC field is programmed with peripheral to memory flow control mode.

–Both SRC_DSCR and DST_DSCR are set to 1 (descriptor fetch is disabled) or Next descriptor location points to 0.

–DIF and SIF are set with their respective layer ID. If SIF is different from DIF, DMA Controller is able to prefetch data and write HSMCI simultaneously.

o. Program LLI_B.DMAC_CFGx memory location for channel x with the following field's values:

– FIFOCFG defines the watermark of the DMA channel FIFO.

– SRC_H2SEL is set to true to enable hardware handshaking on the destination.

– SRC_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.

p. Program LLI_B.DMAC_DSCR with 0.

q. Program DMAC_CTRLBx register for channel x with 0. its content is updated with the LLI fetch operation.

r. Program DMAC_DSCRx with the address of LLI_W if *block_length* greater than 4 else with address of LLI_B.

s. Enable Channel x writing one to DMAC_CHER[x]. The DMAC is ready and waiting for request.

3. Wait for XFRDONE in HSMCI_SR register.

37.8.6.3 Block Length is Not Multiple of 4, with Padding Value (ROPT field in HSMCI_DMA register set to 1)

When the ROPT field is set to one, The DMA Controller performs only WORD access on the bus to transfer a non-multiple of 4 block length. Unlike previous flow, in which the transfer size is rounded to the nearest multiple of 4.

1. Program the HSMCI Interface, see previous flow.
   - ROPT field is set to 1.
2. Program the DMA Controller
   a. Read the channel Register to choose an available (disabled) channel.
   b. Clear any pending interrupts on the channel from the previous DMA transfer by reading the DMAC_EBCISR register.
   c. Program the channel registers.
   d. The DMAC_SADDRx register for channel x must be set with the starting address of the HSMCI_FIFO address.
   e. The DMAC_DADDRx register for channel x must be word aligned.
   f. Program DMAC_CTRLAx register of channel x with the following field's values:
      –DST_WIDTH is set to WORD
      –SRC_WIDTH is set to WORD
      –SCSIZE must be set according to the value of HSMCI_DMA.CHKSIZE Field.
      –BTSIZE is programmed with *CEILING(block_length/4).*
   g. Program DMAC_CTRLBx register for channel x with the following field's values:
      –DST_INCR is set to INCR
      –SRC_INCR is set to INCR
      –FC field is programmed with peripheral to memory flow control mode.
      –both DST_DSCR and SRC_DSCR are set to 1. (descriptor fetch is disabled)
      –DIF and SIF are set with their respective layer ID. If SIF is different from DIF, the DMA Controller is able to prefetch data and write HSMCI simultaneously.
   h. Program DMAC_CFGx register for channel x with the following field's values:
      –FIFOCFG defines the watermark of the DMA channel FIFO.
      –SRC_H2SEL is set to true to enable hardware handshaking on the destination.
      –SRC_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.
      –Enable Channel x writing one to DMAC_CHER[x]. The DMAC is ready and waiting for request.
3. Wait for XFRDONE in HSMCI_SR register.

37.8.7 WRITE_MULTIPLE_BLOCK

37.8.7.1 One Block per Descriptor

1. Wait until the current command execution has successfully terminated.
   a. Check that CMDRDY and NOTBUSY are asserted in HSMCI_SR.
2. Program the block length in the card. This value defines the value *block_length.*
3. Program the block length in the HSMCI configuration register with *block_length* value.
4. Program HSMCI_DMA register with the following fields:
   – OFFSET field with *dma_offset.*
   – CHKSIZE is user defined.
   – DMAEN is set to true to enable DMAC hardware handshaking in the HSMCI. This bit was previously set to false.
5. Issue a WRITE_MULTIPLE_BLOCK command.
6. Program the DMA Controller to use a list of descriptors. Each descriptor transfers one block of data. Block *n* of data is transferred with descriptor LLI(n).
   a. Read the channel Register to choose an available (disabled) channel.
   b. Clear any pending interrupts on the channel from the previous DMAC transfer by reading the DMAC_EBCISR register.
   c. Program a List of descriptors.
   d. The LLI(n).DMAC_SADDRx memory location for channel x must be set to the location of the source data. When the first data location is not word aligned, the two LSB bits define the temporary value called *dma_offset.* The two LSB bits of LLI(n).DMAC_SADDRx must be set to 0.
   e. The LLI(n).DMAC_DADDRx register for channel x must be set with the starting address of the HSMCI_FIFO address.
   f. Program LLI(n).DMAC_CTRLAx register of channel x with the following field's values:
      –DST_WIDTH is set to WORD.
      –SRC_WIDTH is set to WORD.
      –DCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.
      –BTSIZE is programmed with *CEILING((block_length + dma_offset)/4).*
   g. Program LLI(n).DMAC_CTRLBx register for channel x with the following field's values:
      –DST_INCR is set to INCR.
      –SRC_INCR is set to INCR.
      –DST_DSCR is set to 0 (fetch operation is enabled for the destination).
      –SRC_DSCR is set to 1 (source address is contiguous).
      –FC field is programmed with memory to peripheral flow control mode.
      –Both DST_DSCR and SRC_DSCR are set to 1 (descriptor fetch is disabled).
      –DIF and SIF are set with their respective layer ID. If SIF is different from DIF, DMA Controller is able to prefetch data and write HSMCI simultaneously.
   h. Program LLI(n).DMAC_CFGx register for channel x with the following field's values:
      –FIFOCFG defines the watermark of the DMA channel FIFO.
      –DST_H2SEL is set to true to enable hardware handshaking on the destination.
      –SRC_REP is set to 0. (contiguous memory access at block boundary)
      –DST_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.

i. If LLI(n) is the last descriptor, then LLI(n).DSCR points to 0 else LLI(n) points to the start address of LLI(n+1).
j. Program DMAC_CTRLBx for channel register x with 0. Its content is updated with the LLI fetch operation.
k. Program DMAC_DSCRx for channel register x with the address of the first descriptor LLI(0).
l. Enable Channel x writing one to DMAC_CHER[x]. The DMA is ready and waiting for request.

7. Poll CBTC[x] bit in the DMAC_EBCISR Register.
8. If a new list of buffers shall be transferred, repeat step 6. Check and handle HSMCI errors.
9. Poll FIFOEMPTY field in the HSMCI_SR.
10. Send The STOP_TRANSMISSION command writing HSMCI_ARG then HSMCI_CMDR.
11. Wait for XFRDONE in HSMCI_SR register.

37.8.8 READ_MULTIPLE_BLOCK

37.8.8.1 Block Length is a Multiple of 4

1. Wait until the current command execution has successfully terminated.
   a. Check that CMDRDY and NOTBUSY are asserted in HSMCI_SR.
2. Program the block length in the card. This value defines the value *block_length.*
3. Program the block length in the HSMCI configuration register with *block_length* value.
4. Set RDPROOF bit in HSMCI_MR to avoid overflow.
5. Program HSMCI_DMA register with the following fields:
   - ROPT field is set to 0.
   - OFFSET field is set to 0.
   - CHKSIZE is user defined.
   - DMAEN is set to true to enable DMAC hardware handshaking in the HSMCI. This bit was previously set to false.
6. Issue a READ_MULTIPLE_BLOCK command.
7. Program the DMA Controller to use a list of descriptors:
   a. Read the channel Register to choose an available (disabled) channel.
   b. Clear any pending interrupts on the channel from the previous DMA transfer by reading the DMAC_EBCISR register.
   c. Program the channel registers in the Memory with the first descriptor. This descriptor will be word oriented. This descriptor is referred to as LLI_W(n), standing for LLI word oriented transfer for block *n*.
   d. The LLI_W(n).DMAC_SADDRx field in memory must be set with the starting address of the HSMCI_FIFO address.
   e. The LLI_W(n).DMAC_DADDRx field in the memory must be word aligned.
   f. Program LLI_W(n).DMAC_CTRLAx with the following field's values:
      –DST_WIDTH is set to WORD
      –SRC_WIDTH is set to WORD
      –SCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.
      –BTSIZE is programmed with *block_length/4.*
   g. Program LLI_W(n).DMAC_CTRLBx with the following field's values:
      –DST_INCR is set to INCR.
      –SRC_INCR is set to INCR.
      –FC field is programmed with peripheral to memory flow control mode.
      –SRC_DSCR is set to 0 (descriptor fetch is enabled for the SRC).
      –DST_DSCR is set to TRUE (descriptor fetch is disabled for the DST).
      –DIF and SIF are set with their respective layer ID. If SIF is different from DIF, the DMA Controller is able to prefetch data and write HSMCI simultaneously.
   h. Program LLI_W(n).DMAC_CFGx register for channel x with the following field's values:
      –FIFOCFG defines the watermark of the DMA channel FIFO.
      –DST_REP is set to zero. Addresses are contiguous.
      –SRC_H2SEL is set to true to enable hardware handshaking on the destination.
      –SRC_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.

i. Program LLI_W(n).DMAC_DSCRx with the address of LLI_W(n+1) descriptor. And set the DSCRx_IF to the AHB Layer ID. This operation actually links descriptors together. If LLI_W(n) is the last descriptor then LLI_W(n).DMAC_DSCRx points to 0.

j. Program DMAC_CTRLBx register for channel x with 0. its content is updated with the LLI Fetch operation.

k. Program DMAC_DSCRx register for channel x with the address of LLI_W(0).

l. Enable Channel x writing one to DMAC_CHER[x]. The DMA is ready and waiting for request.

8. Poll CBTC[x] bit in the DMAC_EBCISR Register.
9. If a new list of buffer shall be transferred repeat step 6. Check and handle HSMCI errors.
10. Poll FIFOEMPTY field in the HSMCI_SR.
11. Send The STOP_TRANSMISSION command writing the HSMCI_ARG then the HSMCI_CMDR.
12. Wait for XFRDONE in HSMCI_SR register.

37.8.8.2 Block Length is Not Multiple of 4. (ROPT field in HSMCI_DMA register set to 0)

Two DMA Transfer descriptors are used to perform the HSMCI block transfer.

1. Use the previous step to configure the HSMCI to perform a READ_MULTIPLE_BLOCK command.
2. Issue a READ_MULTIPLE_BLOCK command.
3. Program the DMA Controller to use a list of descriptors.
   a. Read the channel register to choose an available (disabled) channel.
   b. Clear any pending interrupts on the channel from the previous DMAC transfer by reading the DMAC_EBCISR register.
   c. For every block of data repeat the following procedure:
   d. Program the channel registers in the Memory for the first descriptor. This descriptor will be word oriented. This descriptor is referred to as LLI_W(n) standing for LLI word oriented transfer for block *n*.
   e. The LLI_W(n).DMAC_SADDRx field in memory must be set with the starting address of the HSMCI_FIFO address.
   f. The LLI_W(n).DMAC_DADDRx field in the memory must be word aligned.
   g. Program LLI_W(n).DMAC_CTRLAx with the following field's values:
      –DST_WIDTH is set to WORD.
      –SRC_WIDTH is set to WORD.
      –SCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.
      –BTSIZE is programmed with *block_length/4.* If BTSIZE is zero, this descriptor is skipped later.
   h. Program LLI_W(n).DMAC_CTRLBx with the following field's values:
      –DST_INCR is set to INCR.
      –SRC_INCR is set to INCR.
      –FC field is programmed with peripheral to memory flow control mode.
      –SRC_DSCR is set to 0 (descriptor fetch is enabled for the SRC).
      –DST_DSCR is set to TRUE (descriptor fetch is disabled for the DST).
      –DIF and SIF are set with their respective layer ID. If SIF is different from DIF, the DMA Controller is able to prefetch data and write HSMCI simultaneously.
   i. Program LLI_W(n).DMAC_CFGx register for channel x with the following field's values:
      –FIFOCFG defines the watermark of the DMA channel FIFO.
      –DST_REP is set to zero. Address are contiguous.
      –SRC_H2SEL is set to true to enable hardware handshaking on the destination.
      –SRC_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.
   j. Program LLI_W(n).DMAC_DSCRx with the address of LLI_B(n) descriptor. And set the DSCRx_IF to the AHB Layer ID. This operation actually links the Word oriented descriptor on the second byte oriented descriptor. When *block_length[1:0]* is equal to 0 (multiple of 4) LLI_W(n).DMAC_DSCRx points to 0, only LLI_W(n) is relevant.
   k. Program the channel registers in the Memory for the second descriptor. This descriptor will be byte oriented. This descriptor is referred to as LLI_B(n), standing for LLI Byte oriented.
   l. The LLI_B(n).DMAC_SADDRx field in memory must be set with the starting address of the HSMCI_FIFO address.
   m. The LLI_B(n).DMAC_DADDRx is not relevant if previous word aligned descriptor was enabled. If 1, 2 or 3 bytes are transferred, that address is user defined and not word aligned.
   n. Program LLI_B(n).DMAC_CTRLAx with the following field's values:
      –DST_WIDTH is set to BYTE.

–SRC_WIDTH is set to BYTE.

–SCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.

–BTSIZE is programmed with *block_length[1:0].* (last 1, 2, or 3 bytes of the buffer).

o. Program LLI_B(n).DMAC_CTRLBx with the following field's values:
   - DST_INCR is set to INCR.
   - SRC_INCR is set to INCR.
   - FC field is programmed with peripheral to memory flow control mode.
   - Both SRC_DSCR and DST_DSCR are set to 1 (descriptor fetch is disabled) or Next descriptor location points to 0.
   - DIF and SIF are set with their respective layer ID. If SIF is different from DIF, the DMA Controller is able to prefetch data and write HSMCI simultaneously.

p. Program LLI_B(n).DMAC_CFGx memory location for channel x with the following field's values:
   - FIFOCFG defines the watermark of the DMAC channel FIFO.
   - SRC_H2SEL is set to true to enable hardware handshaking on the destination.
   - SRC_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller

q. Program LLI_B(n).DMAC_DSCR with address of descriptor LLI_W(n+1). If LLI_B(n) is the last descriptor, then program LLI_B(n).DMAC_DSCR with 0.

r. Program DMAC_CTRLBx register for channel x with 0, its content is updated with the LLI Fetch operation.

s. Program DMAC_DSCRx with the address of LLI_W(0) if *block_length* is greater than 4 else with address of LLI_B(0).

t. Enable Channel x writing one to DMAC_CHER[x]. The DMAC is ready and waiting for request.

4. Enable DMADONE interrupt in the HSMCI_IER register.
5. Poll CBTC[x] bit in the DMAC_EBCISR Register.
6. If a new list of buffers shall be transferred, repeat step 7. Check and handle HSMCI errors.
7. Poll FIFOEMPTY field in the HSMCI_SR.
8. Send The STOP_TRANSMISSION command writing HSMCI_ARG then HSMCI_CMDR.
9. Wait for XFRDONE in HSMCI_SR register.

37.8.8.3 Block Length is Not a Multiple of 4. (ROPT field in HSMCI_DMA register set to 1)

One DMA Transfer descriptor is used to perform the HSMCI block transfer, the DMA writes a rounded up value to the nearest multiple of 4.

1. Use the previous step to configure the HSMCI to perform a READ_MULTIPLE_BLOCK.
2. Set the ROPT field to 1 in the HSMCI_DMA register.
3. Issue a READ_MULTIPLE_BLOCK command.
4. Program the DMA controller to use a list of descriptors:
   a. Read the channel Register to choose an available (disabled) channel.
   b. Clear any pending interrupts on the channel from the previous DMAC transfer by reading the DMAC_EBCISR register.
   c. Program the channel registers in the Memory with the first descriptor. This descriptor will be word oriented. This descriptor is referred to as LLI_W(n), standing for LLI word oriented transfer for block *n*.
   d. The LLI_W(n).DMAC_SADDRx field in memory must be set with the starting address of the HSMCI_FIFO address.
   e. The LLI_W(n).DMAC_DADDRx field in the memory must be word aligned.
   f. Program LLI_W(n).DMAC_CTRLAx with the following field's values:
      –DST_WIDTH is set to WORD.
      –SRC_WIDTH is set to WORD.
      –SCSIZE must be set according to the value of HSMCI_DMA, CHKSIZE field.
      –BTSIZE is programmed with *Ceiling(block_length/4).*
   g. Program LLI_W(n).DMAC_CTRLBx with the following field's values:
      –DST_INCR is set to INCR
      –SRC_INCR is set to INCR
      –FC field is programmed with peripheral to memory flow control mode.
      –SRC_DSCR is set to 0. (descriptor fetch is enabled for the SRC)
      –DST_DSCR is set to TRUE. (descriptor fetch is disabled for the DST)
      –DIF and SIF are set with their respective layer ID. If SIF is different from DIF, the DMA Controller is able to prefetch data and write HSMCI simultaneously.
   h. Program LLI_W(n).DMAC_CFGx register for channel x with the following field's values:
      –FIFOCFG defines the watermark of the DMA channel FIFO.
      –DST_REP is set to zero. Address are contiguous.
      –SRC_H2SEL is set to true to enable hardware handshaking on the destination.
      –SRC_PER is programmed with the hardware handshaking ID of the targeted HSMCI Host Controller.
   i. Program LLI_W(n).DMAC_DSCRx with the address of LLI_W(n+1) descriptor. And set the DSCRx_IF to the AHB Layer ID. This operation actually links descriptors together. If LLI_W(n) is the last descriptor then LLI_W(n).DMAC_DSCRx points to 0.
   j. Program DMAC_CTRLBx register for channel x with 0. its content is updated with the LLI Fetch operation.
   k. Program DMAC_DSCRx register for channel x with the address of LLI_W(0).
   l. Enable Channel x writing one to DMAC_CHER[x]. The DMAC is ready and waiting for request.
5. Poll CBTC[x] bit in the DMAC_EBCISR Register.
6. If a new list of buffers shall be transferred repeat step 7. Check and handle HSMCI errors.
7. Poll FIFOEMPTY field in the HSMCI_SR.
8. Send The STOP_TRANSMISSION command writing the HSMCI_ARG then the HSMCI_CMDR.

9. Wait for XFRDONE in HSMCI_SR register.

37.9 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 Multi Media 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).

37.9.1 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.

37.9.2 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.

37.10 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.

37.10.1 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.

37.10.2 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.

37.10.3 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.

37.10.4 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_TRANMISSION (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.

37.11 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.

37.11.1 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.

37.11.2 Boot Procedure DMA Mode

1. Configure the HSMCI data bus width by programming SDCBUS Field in the HSMCI_SDCR register. The BOOT_BUS_WIDTH field 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 by writing BLKLEN and BCNT fields of the HSMCI_BLKR Register.
3. Enable DMA transfer in the HSMCI_DMA register.
4. Configure DMA controller, program the total amount of data to be transferred and enable the relevant channel.
5. Issue the Boot Operation Request command by writing to the HSMCI_CMDR register with SPCND set to BOOTREQ, TRDIR set to READ and TRCMD set to “start data transfer”.
6. DMA controller copies the boot partition to the memory.
7. When DMA transfer is completed, host processor shall terminate the boot stream by writing the HSMCI_CMDR register with SPCMD field set to BOOTEND.

37.12 HSMCI Transfer Done Timings

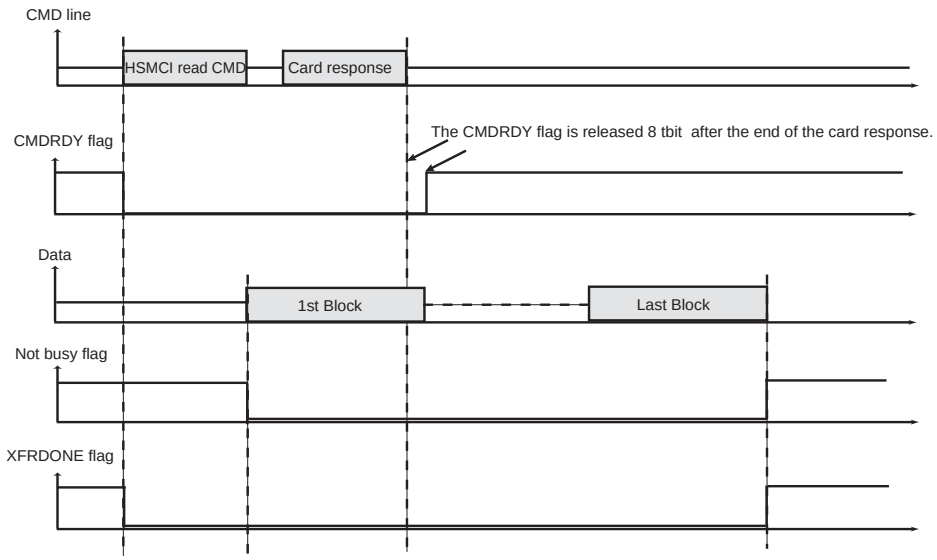
37.12.1 Definition

The XFRDONE flag in the HSMCI_SR indicates exactly when the read or write sequence is finished.

37.12.2 Read Access

During a read access, the XFRDONE flag behaves as shown in Figure 37-12.

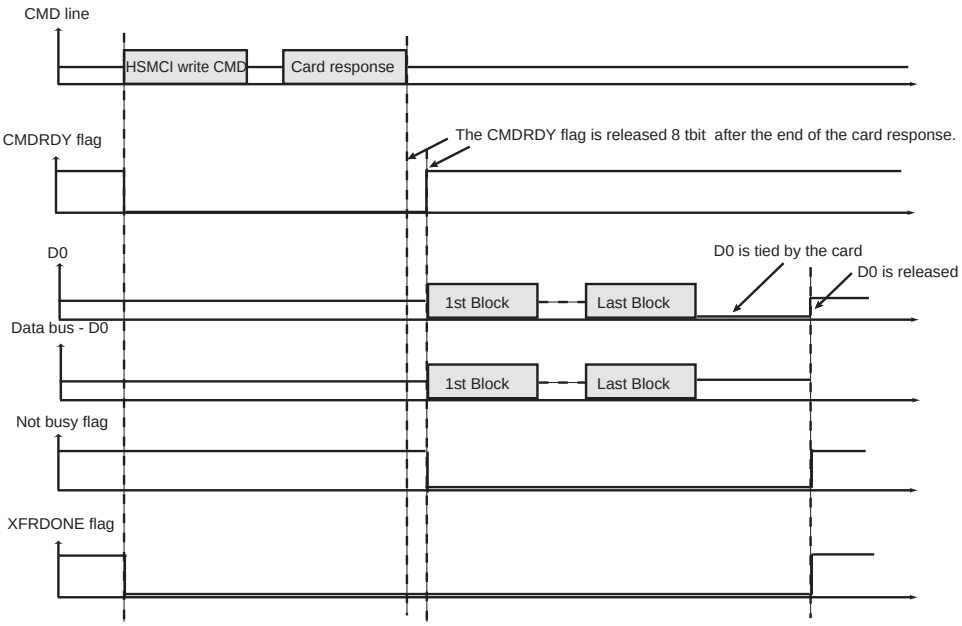
Figure 37-12. XFRDONE During a Read Access



37.12.3 Write Access

During a write access, the XFRDONE flag behaves as shown in Figure 37-13.

Figure 37-13. XFRDONE During a Write Access



37.13 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 WPVSRC 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 943
- "HSMCI Data Timeout Register" on page 945
- "HSMCI SDCard/SDIO Register" on page 946
- "HSMCI Completion Signal Timeout Register" on page 951
- "HSMCI DMA Configuration Register" on page 965
- "HSMCI Configuration Register" on page 966

37.14 High Speed MultiMedia Card Interface (HSMCI) User Interface

Table 37-9. 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(1)	HSMCI_RSPR	Read	0x0
0x24	Response Register(1)	HSMCI_RSPR	Read	0x0
0x28	Response Register(1)	HSMCI_RSPR	Read	0x0
0x2C	Response Register(1)	HSMCI_RSPR	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	DMA Configuration Register	HSMCI_DMA	Read-write	0x00
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	–	–	–
0x100-0x1FC	Reserved	–	–	–
0x200	FIFO Memory Aperture0	HSMCI_FIFO0	Read-write	0x0
...	...	...	...	...
0x5FC	FIFO Memory Aperture255	HSMCI_FIFO255	Read-write	0x0

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.

37.14.1 HSMCI Control Register

Name: HSMCI_CR

Address: 0x40000000

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.

37.14.2 HSMCI Mode Register

Name: HSMCI_MR

Address: 0x40000004

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
–	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 967.

- **CLKDIV: Clock Divider**

High Speed MultiMedia Card Interface clock (MCCK or HSMCI_CK) is Master Clock (MCK) divided by (2*(CLKDIV+1)).

- **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.

- **BLKLEN: Data Block Length**

This field determines the size of the data block.

This field is also accessible in the HSMCI Block Register (HSMCI_BLKR).

Bits 16 and 17 must be set to 0 if FBYTE is disabled.

Note: In SDIO Byte mode, BLKLEN field is not used.

37.14.3 HSMCI Data Timeout Register

Name: HSMCI_DTOR

Address: 0x40000008

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 967.

- **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) raises.

37.14.4 HSMCI SDCard/SDIO Register

Name: HSMCI_SDCR

Address: 0x4000000C

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 967.

- **SDCSEL: SDCard/SDIO Slot**

Value	Name	Description
0	SLOTA	Slot A is selected.
1	SLOTB	SDCARD/SDIO Slot B selected
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

37.14.5 HSMCI Argument Register

Name: HSMCI_ARGR

Address: 0x40000010

Access: Read-write

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

- **ARG: Command Argument**

37.14.6 HSMCI Command Register

Name: HSMCI_CMDR

Address: 0x40000014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	BOOT_ACK	ATACS	IOSPCMD	
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 writeable by an interrupt response (field SPCMD). This means that the current command execution cannot be interrupted or modified.

- **CMDNB: 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/SDCard Single Block
1	MULTIPLE	MMC/SDCard 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.

37.14.7 HSMCI Block Register

Name: HSMCI_BLKR

Address: 0x40000018

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):

Value	Name	Description
0	MULTIPLE	MMC/SDCARD Multiple Block From 1 to 65635: Value 0 corresponds to an infinite block transfer.
4	BYTE	SDIO Byte From 1 to 512 bytes: Value 0 corresponds to a 512-byte transfer. Values from 0x200 to 0xFFFF are forbidden.
5	BLOCK	SDIO Block From 1 to 511 blocks: Value 0 corresponds to an infinite block transfer. Values from 0x200 to 0xFFFF are forbidden.

Warning: In SDIO Byte and Block modes, writing to the 7 last bits of BCNT field is forbidden and may lead to unpredictable results.

- **BLKLEN: Data Block Length**

This field determines the size of the data block.

This field is also accessible in the HSMCI Mode Register (HSMCI_MR).

Bits 16 and 17 must be set to 0 if FBYTE is disabled.

Note: In SDIO Byte mode, BLKLEN field is not used.

37.14.8 HSMCI Completion Signal Timeout Register

Name: HSMCI_CSTOR

Address: 0x4000001C

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 967.

- **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) raises.

37.14.9 HSMCI Response Register

Name: HSMCI_RSPR

Address: 0x40000020

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.

37.14.10 HSMCI Receive Data Register

Name: HSMCI_RDR

Address: 0x40000030

Access: Read-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 Read**

37.14.11 HSMCI Transmit Data Register

Name: HSMCI_TDR

Address: 0x40000034

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 Write**

37.14.12 HSMCI Status Register

Name: HSMCI_SR

Address: 0x40000040

Access: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	DMADONE	BLKOVRE
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
–	–	CSRCV	SDIOWAIT	–	–	SDIO IRQ for Slot B	SDIO IRQ for Slot A
7	6	5	4	3	2	1	0
–	–	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**

This flag must be used only for Write Operations.

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.

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.

Refer to the MMC or SD Specification for more details concerning the busy behavior.

- **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.

- **SDIOIRQB: SDIO Interrupt for Slot B**

0: No interrupt detected on SDIO Slot B.

1: An SDIO Interrupt on Slot B 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.

- **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.

- **BLKOVRE: DMA Block Overrun Error**

0: No error.

1: A new block of data is received and the DMA controller has not started to move the current pending block, a block overrun is raised. Cleared by reading in the HSMCI_SR register.

- **DMADONE: DMA Transfer done**

0: DMA buffer transfer has not completed since the last read of HSMCI_SR register.

1: DMA buffer transfer has completed.

- **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.

37.14.13 HSMCI Interrupt Enable Register

Name: HSMCI_IER

Address: 0x40000044

Access: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	DMADONE	BLKOVRE
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
–	–	CSRCV	SDIOWAIT	–	–	SDIO IRQ for Slot B	SDIO IRQ for Slot A
7	6	5	4	3	2	1	0
–	–	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**
- **SDIOIRQA: SDIO Interrupt for Slot A Interrupt Enable**
- **SDIOIRQB: SDIO Interrupt for Slot B Interrupt Enable**
- **SDIOIRQD: SDIO Interrupt for Slot D Interrupt Enable**
- **SDIOWAIT: SDIO Read Wait Operation Status Interrupt Enable**
- **CSRCV: Completion Signal Received 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**

- **BLKOVRE: DMA Block Overrun Error Interrupt Enable**
- **DMADONE: DMA Transfer completed 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.

37.14.14 HSMCI Interrupt Disable Register

Name: HSMCI_IDR

Address: 0x40000048

Access: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	DMADONE	BLKOVRE
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
–	–	CSRCV	SDIOWAIT	–	–	SDIO IRQ for Slot B	SDIO IRQ for Slot A
7	6	5	4	3	2	1	0
–	–	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**
- **SDIOIRQA: SDIO Interrupt for Slot A Interrupt Disable**
- **SDIOIRQB: SDIO Interrupt for Slot B Interrupt Disable**
- **SDIOWAIT: SDIO Read Wait Operation Status Interrupt Disable**
- **CSRCV: Completion Signal received 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**
- **BLKOVRE: DMA Block Overrun Error Interrupt Disable**

- **DMADONE: DMA Transfer completed 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.

37.14.15 HSMCI Interrupt Mask Register

Name: HSMCI_IMR

Address: 0x4000004C

Access: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	DMADONE	BLKOVRE
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
–	–	CSRCV	SDIOWAIT	–	–	SDIO IRQ for Slot B	SDIO IRQ for Slot A
7	6	5	4	3	2	1	0
–	–	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**
- **SDIOIRQA: SDIO Interrupt for Slot A Interrupt Mask**
- **SDIOIRQB: SDIO Interrupt for Slot B Interrupt Mask**
- **SDIOWAIT: SDIO Read Wait Operation Status Interrupt Mask**
- **CSRCV: Completion Signal Received 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**
- **BLKOVRE: DMA Block Overrun Error Interrupt Mask**

- **DMADONE: DMA Transfer Completed 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.

37.14.16 HSMCI DMA Configuration Register

Name: HSMCI_DMA

Address: 0x40000050

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
–	–	–	ROPT	–	–	–	DMAEN
7	6	5	4	3	2	1	0
–	–	–	CHKSIZE	–	–	OFFSET	

This register can only be written if the WPEN bit is cleared in “HSMCI Write Protect Mode Register” on page 967.

- **OFFSET: DMA Write Buffer Offset**

This field indicates the number of discarded bytes when the DMA writes the first word of the transfer.

- **CHKSIZE: DMA Channel Read and Write Chunk Size**

The CHKSIZE field indicates the number of data available when the DMA chunk transfer request is asserted.

Value	Name	Description
0	1	1 data available
1	4	4 data available

- **DMAEN: DMA Hardware Handshaking Enable**

0: DMA interface is disabled.

1: DMA Interface is enabled.

Note: To avoid unpredictable behavior, DMA hardware handshaking must be disabled when CPU transfers are performed.

- **ROPT: Read Optimization with padding**

0: BLKLEN bytes are moved from the Memory Card to the system memory, two DMA descriptors are used when the transfer size is not a multiple of 4.

1: Ceiling(BLKLEN/4) * 4 bytes are moved from the Memory Card to the system memory, only one DMA descriptor is used.

37.14.17 HSMCI Configuration Register

Name: HSMCI_CFG

Address: 0x40000054

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 967.

- **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)

37.14.18 HSMCI Write Protect Mode Register

Name: HSMCI_WPMR

Address: 0x400000E4

Access: Read-write

31	30	29	28	27	26	25	24
WP_KEY (0x4D => “M”)							
23	22	21	20	19	18	17	16
WP_KEY (0x43 => C”)							
15	14	13	12	11	10	9	8
WP_KEY (0x49 => “I”)							
7	6	5	4	3	2	1	0
							WP_EN

- **WP_EN: Write Protection Enable**

0: Disables the Write Protection if WP_KEY corresponds to 0x4D4349 (“MCI’ in ASCII).

1: Enables the Write Protection if WP_KEY corresponds to 0x4D4349 (“MCI’ in ASCII).

- **WP_KEY: Write Protection Key password**

Should be written at value **0x4D4349** (ASCII code for “MCI”). Writing any other value in this field has no effect.

Protects the registers:

- “HSMCI Mode Register” on page 943
- “HSMCI Data Timeout Register” on page 945
- “HSMCI SDCard/SDIO Register” on page 946
- “HSMCI Completion Signal Timeout Register” on page 951
- “HSMCI DMA Configuration Register” on page 965
- “HSMCI Configuration Register” on page 966

37.14.19 HSMCI Write Protect Status Register

Name: HSMCI_WPSR

Address: 0x400000E8

Access: Read-only

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

- **WP_VS: Write Protection Violation Status**

Value	Name	Description
0	NONE	No Write Protection Violation occurred since the last read of this register (WP_SR)
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.

- **WP_VSRC: 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.

37.14.20 HSMCI FIFOx Memory Aperture

Name: HSMCI_FIFOx[x=0..255]

Address: 0x40000200

Access: Read-write

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 Read or Data to Write**

38. Pulse Width Modulation (PWM)

38.1 Description

The PWM macrocell controls 8 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).

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 duty-cycle 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 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 6 fault inputs**,** capable of detecting a fault condition and to override the PWM outputs asynchronously.

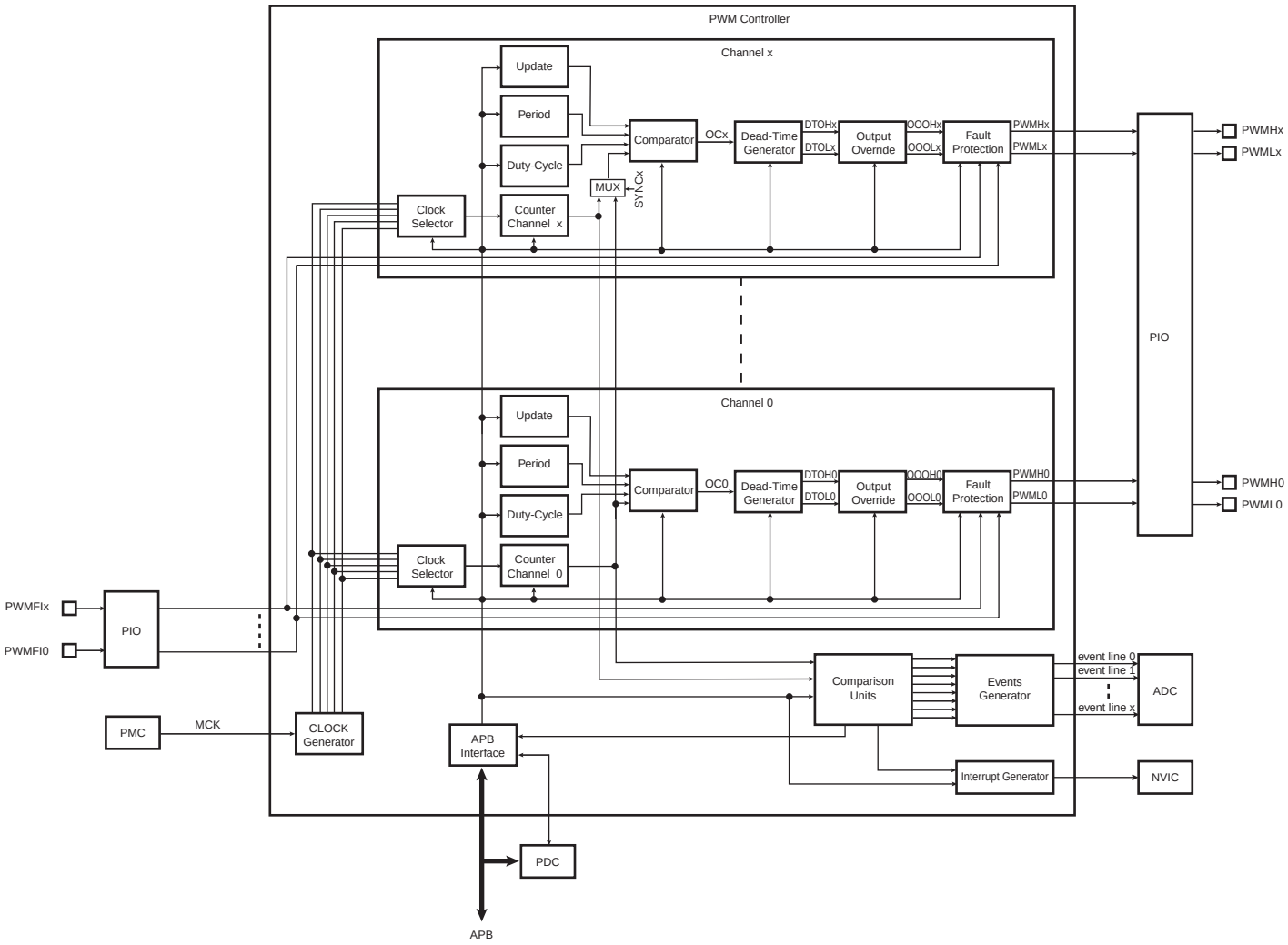
For safety usage, some control registers are write-protected.

38.2 Embedded Characteristics

- 8 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
  - Independent Programmable Center or Left Aligned Output Waveform for Each Channel
  - Independent Output Override for Each Channel
- 4 2-bit Gray Up/Down Channels for Stepper Motor Control
- 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
- 8 Comparison Units Intended to Generate Interrupts, Pulses on Event Lines and PDC Transfer Requests
- 6 Programmable Fault Inputs Providing an Asynchronous Protection of PWM Outputs
  - User Driven through PIO inputs
  - PMC Driven when Crystal Oscillator Clock Fails
  - ADC Controller Driven through Configurable Comparison Function
  - Timer/Counter Driven through Configurable Comparison Function
- Write-Protect Registers

38.3 Block Diagram

Figure 38-1. Pulse Width Modulation Controller Block Diagram



38.4 I/O Lines Description

Each channel outputs two complementary external I/O lines.

Table 38-1. 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

38.5 Product Dependencies

38.5.1 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 38-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
PWM	PWMFI0	PA5	B
PWM	PWMFI1	PA3	B
PWM	PWMFI2	PD6	B
PWM	PWMH0	PA8	B
PWM	PWMH0	PB12	B
PWM	PWMH0	PC3	B
PWM	PWMH0	PE15	A
PWM	PWMH1	PA19	B
PWM	PWMH1	PB13	B
PWM	PWMH1	PC5	B
PWM	PWMH1	PE16	A
PWM	PWMH2	PA13	B
PWM	PWMH2	PB14	B
PWM	PWMH2	PC7	B
PWM	PWMH3	PA9	B
PWM	PWMH3	PB15	B
PWM	PWMH3	PC9	B
PWM	PWMH3	PF3	A
PWM	PWMH4	PC20	B
PWM	PWMH4	PE20	A
PWM	PWMH5	PC19	B
PWM	PWMH5	PE22	A
PWM	PWMH6	PC18	B
PWM	PWMH6	PE24	A
PWM	PWMH7	PE26	A
PWM	PWML0	PA21	B
PWM	PWML0	PB16	B
PWM	PWML0	PC2	B
PWM	PWML0	PE18	A
PWM	PWML1	PA12	B

Table 38-2. I/O Lines (Continued)

PWM	PWML1	PB17	B
PWM	PWML1	PC4	B
PWM	PWML2	PA20	B
PWM	PWML2	PB18	B
PWM	PWML2	PC6	B
PWM	PWML2	PE17	A
PWM	PWML3	PA0	B
PWM	PWML3	PB19	B
PWM	PWML3	PC8	B
PWM	PWML4	PC21	B
PWM	PWML4	PE19	A
PWM	PWML5	PC22	B
PWM	PWML5	PE21	A
PWM	PWML6	PC23	B
PWM	PWML6	PE23	A
PWM	PWML7	PC24	B
PWM	PWML7	PE25	A

38.5.2 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.

38.5.3 Interrupt Sources

The PWM interrupt line is connected on one of the internal sources of the Nested Vectored Interrupt Controller (NVIC). Using the PWM interrupt requires the NVIC to be programmed first.

Table 38-3. Peripheral IDs

Instance	ID
PWM	36

38.5.4 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, and Timer/Counters

Table 38-4. Fault Inputs

Fault Inputs	External PWM Fault Input Number	Polarity Level[1]	Fault Input ID
PA5	PWMFI0	User Defined	0
PA3	PWMFI1	User Defined	1
PD6	PWMFI2	User Defined	2
MAIN OSC	–	1	3
ADC	–	1	4
Timer0	–	1	5

Note: 1. FPOL bit in PWMC_FMR.

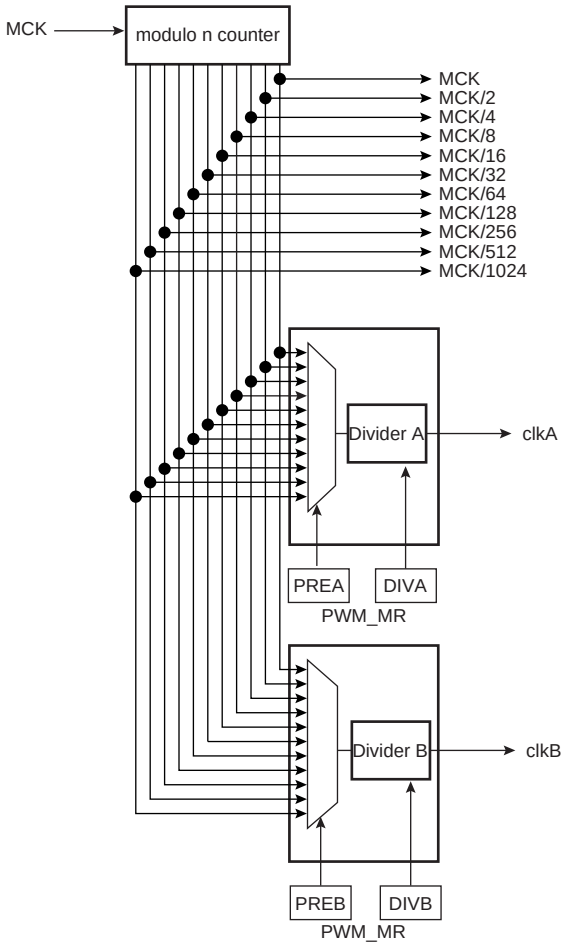
38.6 Functional Description

The PWM macrocell is primarily composed of a clock generator module and 8 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.

38.6.1 PWM Clock Generator

Figure 38-2. 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.

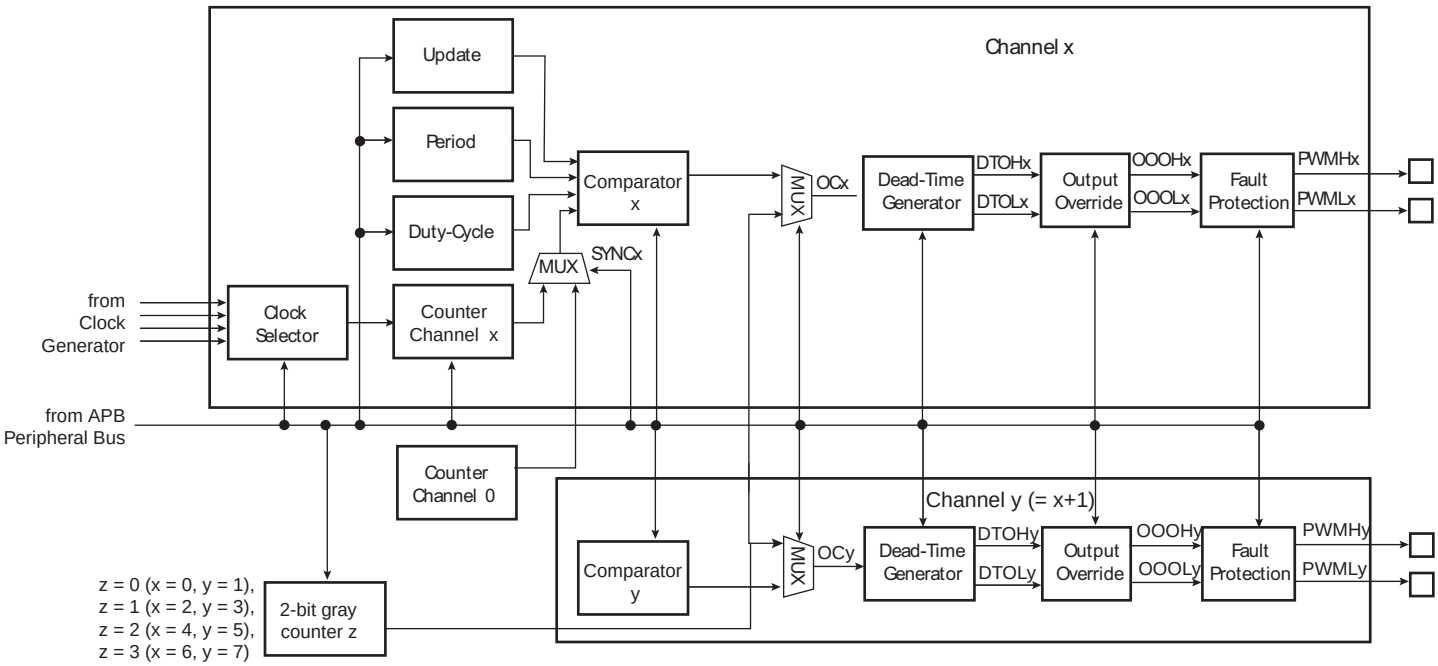
CAUTION:

- Before using the PWM macrocell, the programmer must first enable the PWM clock in the Power Management Controller (PMC).

38.6.2 PWM Channel

38.6.2.1 Block Diagram

Figure 38-3. Functional View of the Channel Block Diagram



Each of the 8 channels is composed of six blocks:

- A clock selector which selects one of the clocks provided by the clock generator (described in Section 38.6.1 on page 976).
- 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 1014 (PWM_SCM).
- 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 in case of fault detection (PWMHx/PWMLx).

38.6.2.2 Comparator

The comparator continuously compares its counter value with the channel period defined by CPRD in the “PWM Channel Period Register” on page 1048 (PWM_CPRDx) and the duty-cycle defined by CDTY in the “PWM Channel Duty Cycle Register” on page 1046 (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 1044 (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:

  $$\frac{(CPRD \times DIVA)}{MCK} \; or \; \frac{(CPRD \times DIVB)}{MCK}$$

  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:

  $$\frac{(2 \times CPRD \times DIVA)}{MCK} \; or \; \frac{(2 \times CPRD \times DIVB)}{MCK}$$

- 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:

  $$\text{duty cycle} = \frac{(period - 1/\text{fchannel_x_clock} \times CDTY)}{period}$$

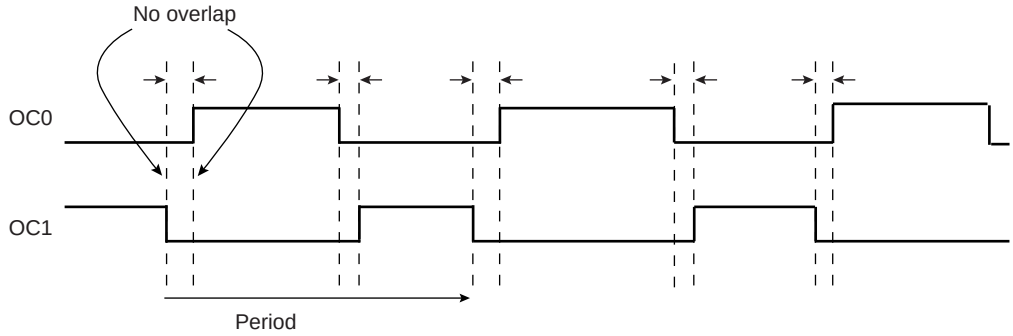
  If the waveform is center aligned, then:

  $$\text{duty cycle} = \frac{((period/2) - 1/\text{fchannel_x_clock} \times CDTY))}{(period/2)}$$

- 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 38-4. Non Overlapped Center Aligned Waveforms



Note: 1. See Figure 38-5 on page 980 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:

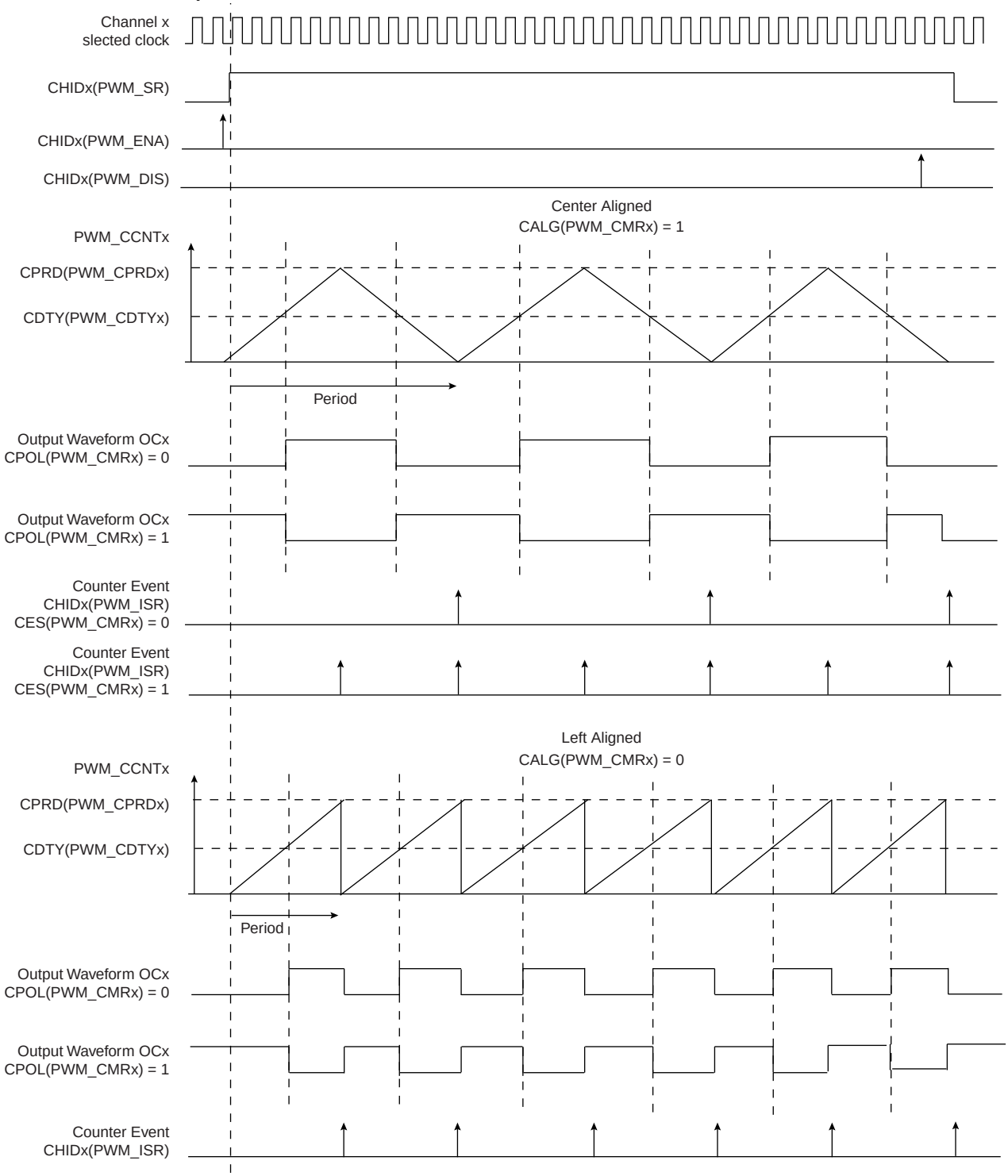
- CDTY = 0 and CPOL = 0
- CDTY = CPRD and CPOL = 1

The waveform polarity must be set before enabling the channel. This immediately affects the channel output level. Changes on channel polarity are not taken into account while the channel is enabled.

Besides generating output signals OCx, 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 38-5 “Waveform Properties” illustrates the counter interrupts in function of the configuration.

Figure 38-5. Waveform Properties



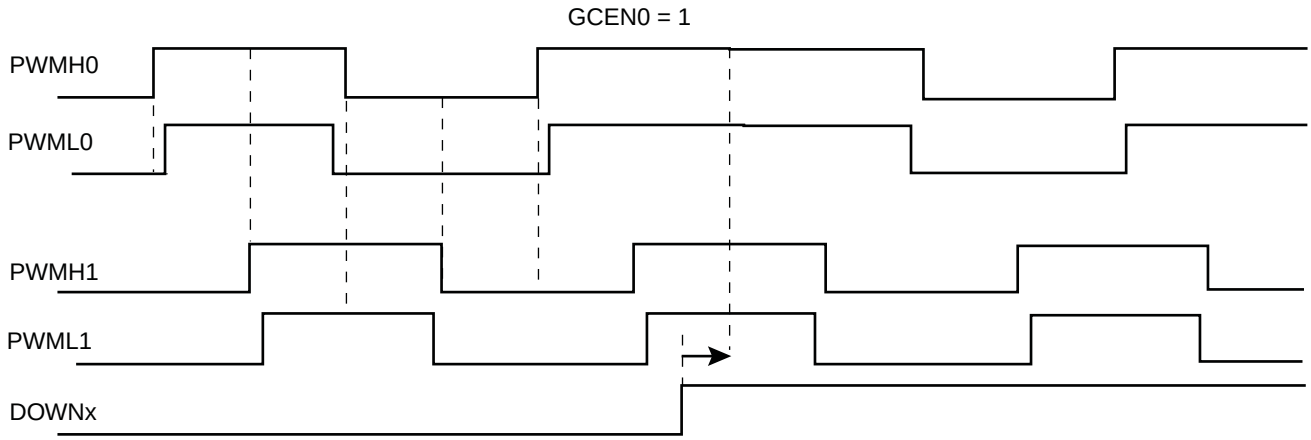
38.6.2.3 2-bit Gray Up/Down Counter for Stepper Motor

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 38-6. 2-bit Gray Up/Down Counter



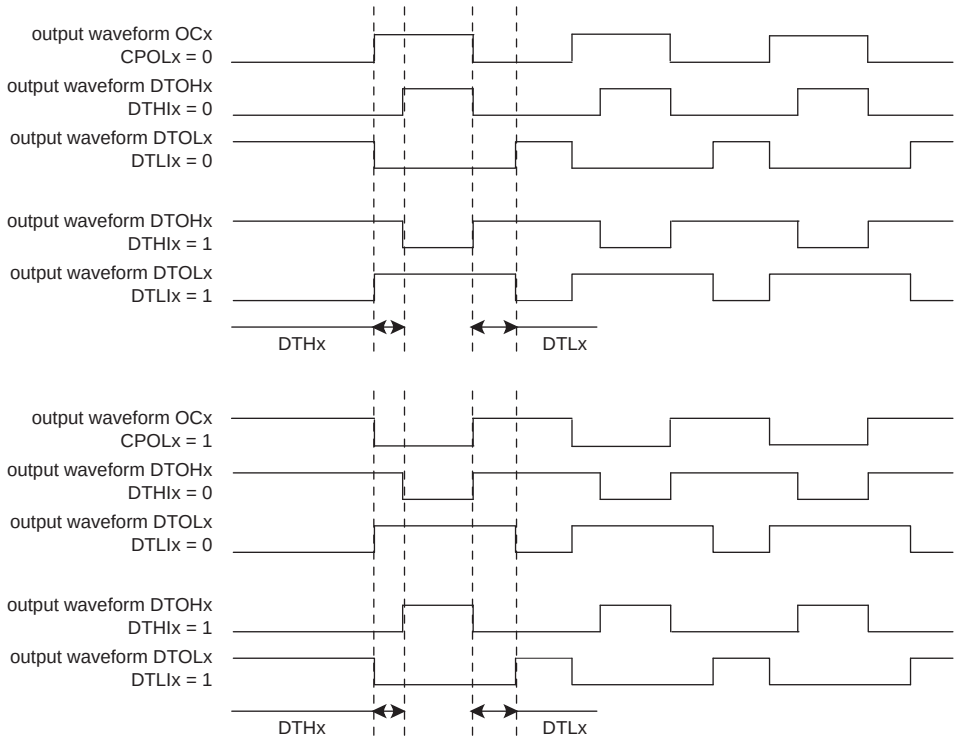
38.6.2.4 Dead-Time Generator

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.

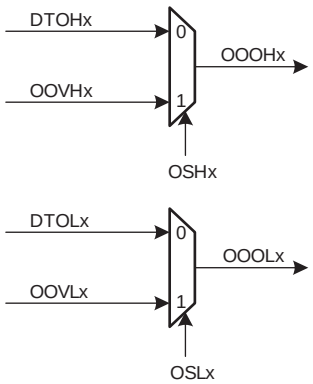
Figure 38-7. Complementary Output Waveforms



38.6.2.5 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 38-8. 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.

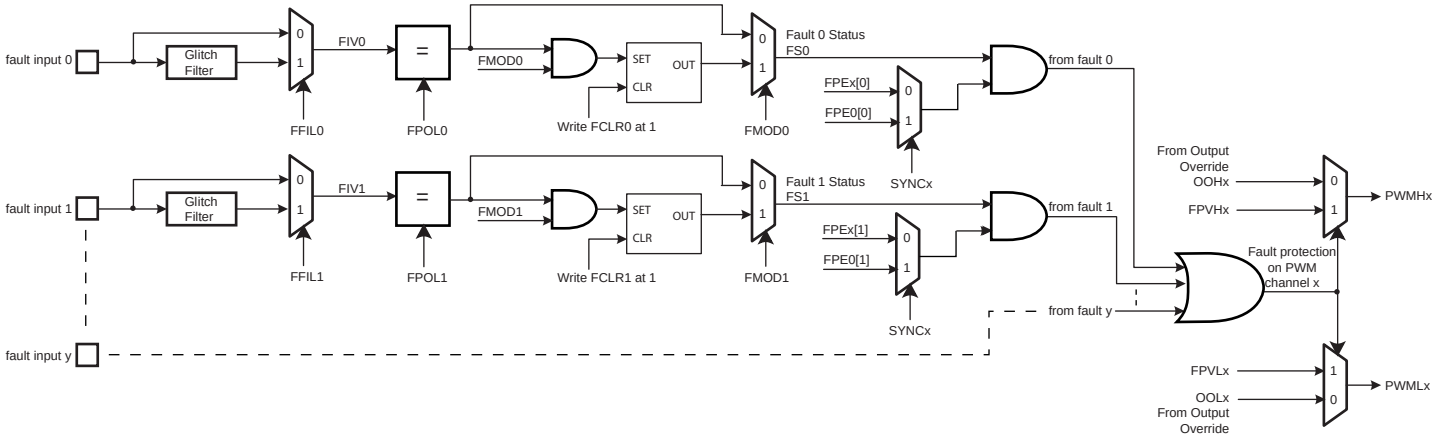
The value of the current output selection can be read in PWM_OS.

While overriding PWM outputs, the channel counters continue to run, only the PWM outputs are forced to user defined values.

38.6.2.6 Fault Protection

6 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 38-9. 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_FPE1/2). However the synchronous channels (see Section 38.6.2.7 “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 forces the channel outputs to the values defined by the fields FPVHx and FPVLx in the "PWM Fault Protection Value Register" (PWM_FPV) and leads to a reset of the counter of this channel. The output forcing is made asynchronously to the channel counter.

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 38.6.3 "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.

38.6.2.7 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_CMR0 register instead of CPREx field in PWM_CMRx register (same source clock)
- CPRD0 field in PWM_CMR0 register instead of CPRDx field in PWM_CMRx register (same period)
- CALG0 field in PWM_CMR0 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 987).
- 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 988).
- 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 990). The user can choose to synchronize the PDC transfer request with a comparison match (see Section 38.6.3 "PWM Comparison Units"), by the fields PTRM and PTRCS in the PWM_SCM register.

Table 38-5. 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:

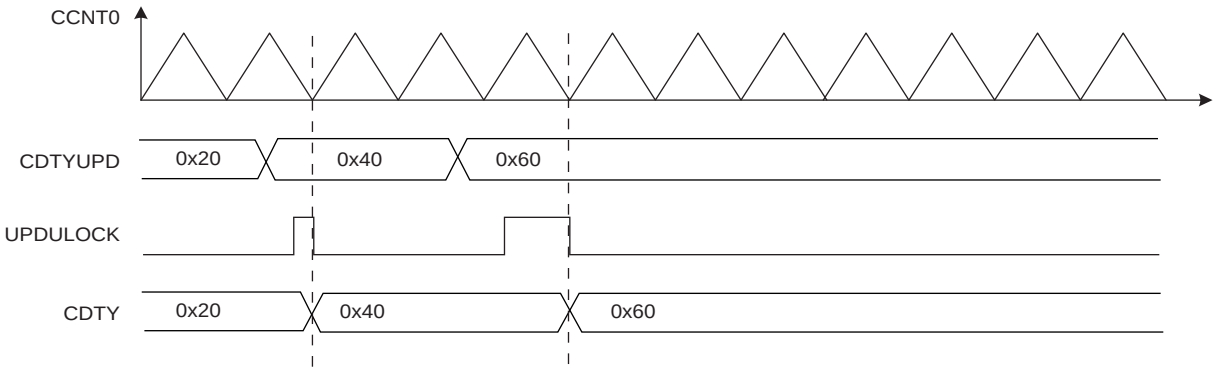
- If the bit UPDULOCK is set to 1, the update is done at the next PWM period of the synchronous channels.
- If the UPDULOCK bit is not set to 1, the update is locked and cannot be performed.

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 38-10. 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:

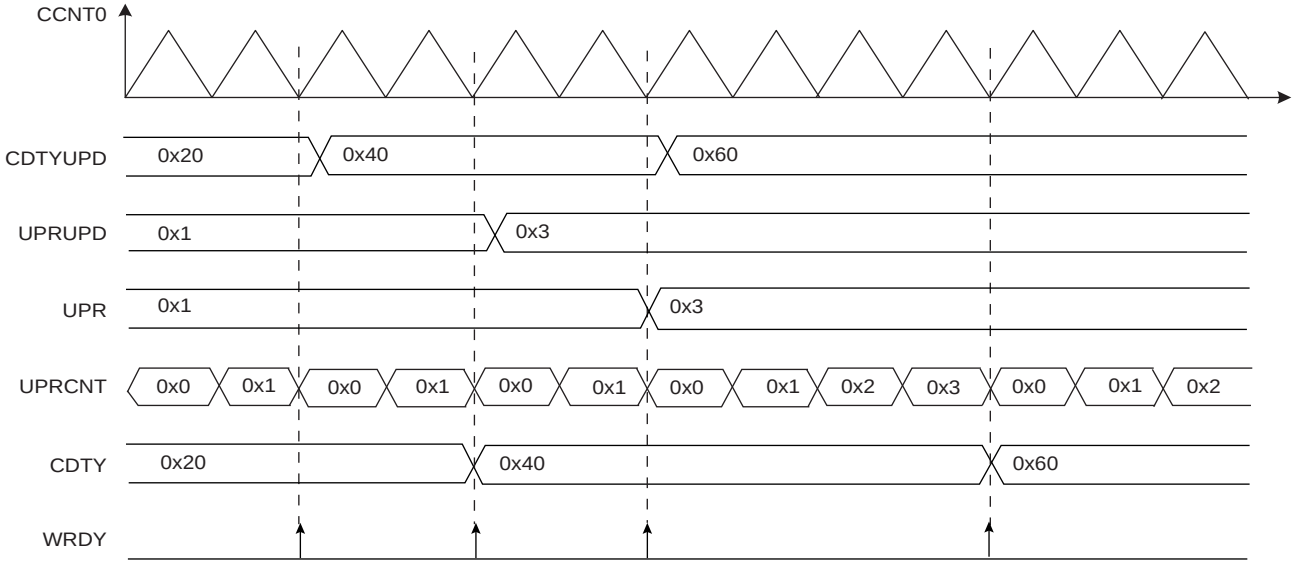
- WRDY: this flag is set to 1 when the PWM Controller is ready to receive new duty-cycle values and a new update period value. It is reset to 0 when the PWM_ISR2 register is read.

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 38-11. 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 38.6.3 "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 38-12. Method 3 (UPDM=2 and PTRM=0)

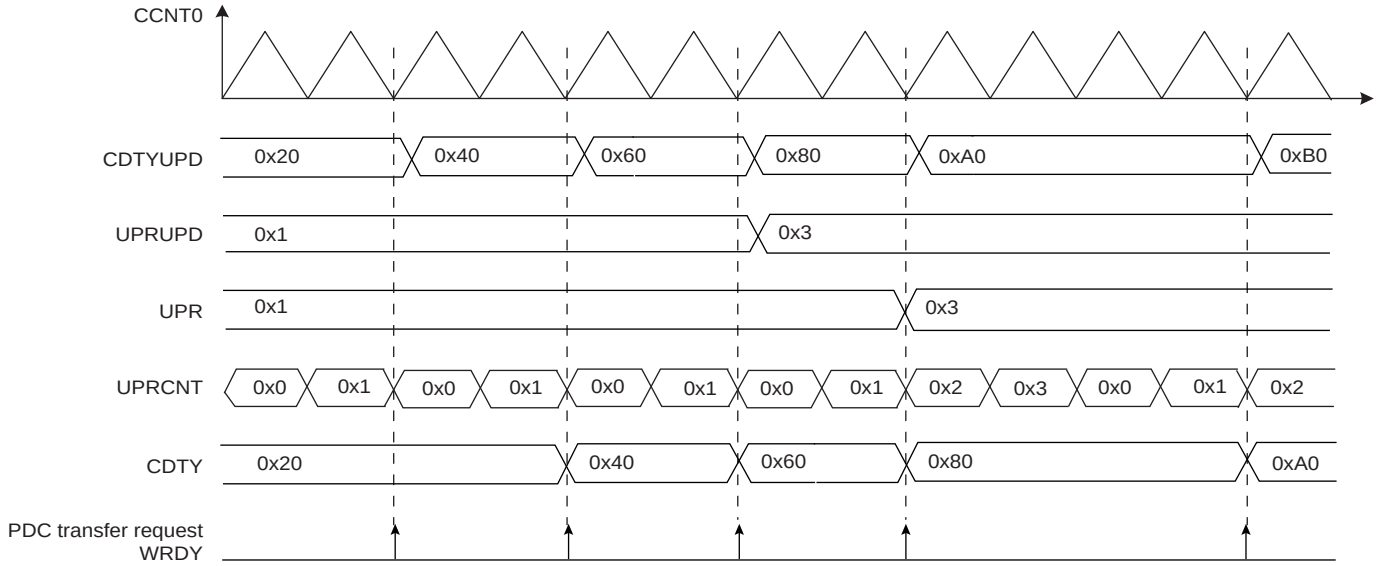
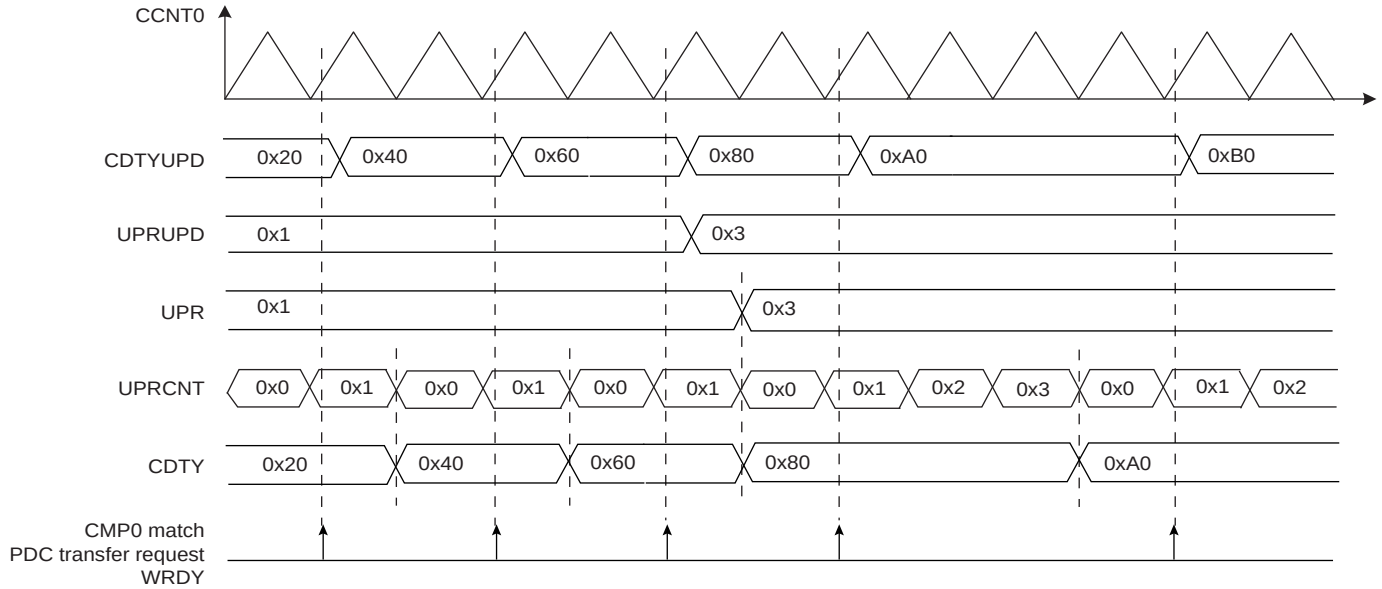


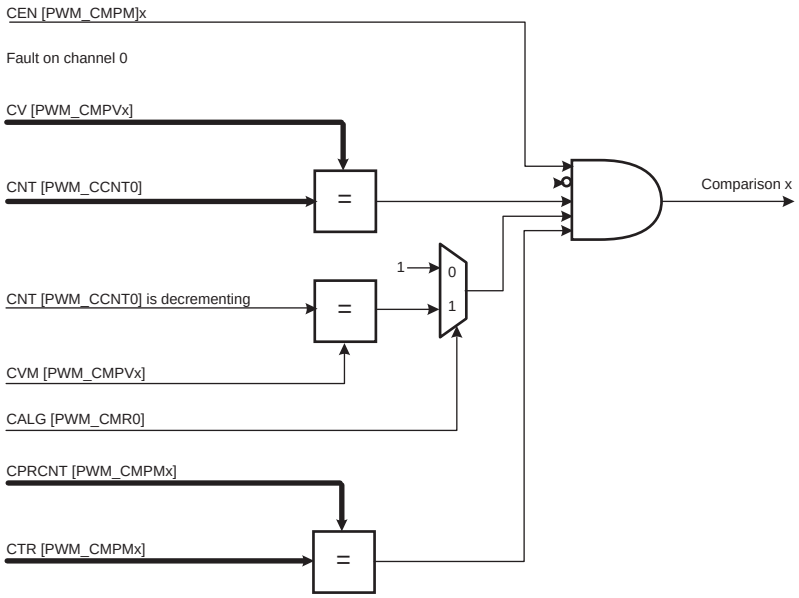
Figure 38-13. Method 3 (UPDM=2 and PTRM=1 and PTRCS=0)



38.6.3 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 38.6.2.7 “Synchronous Channels”). These comparisons are intended to generate pulses on the event lines (used to synchronize ADC, see Section 38.6.4 “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 990).

Figure 38-14. 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 38.6.2.6 “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).

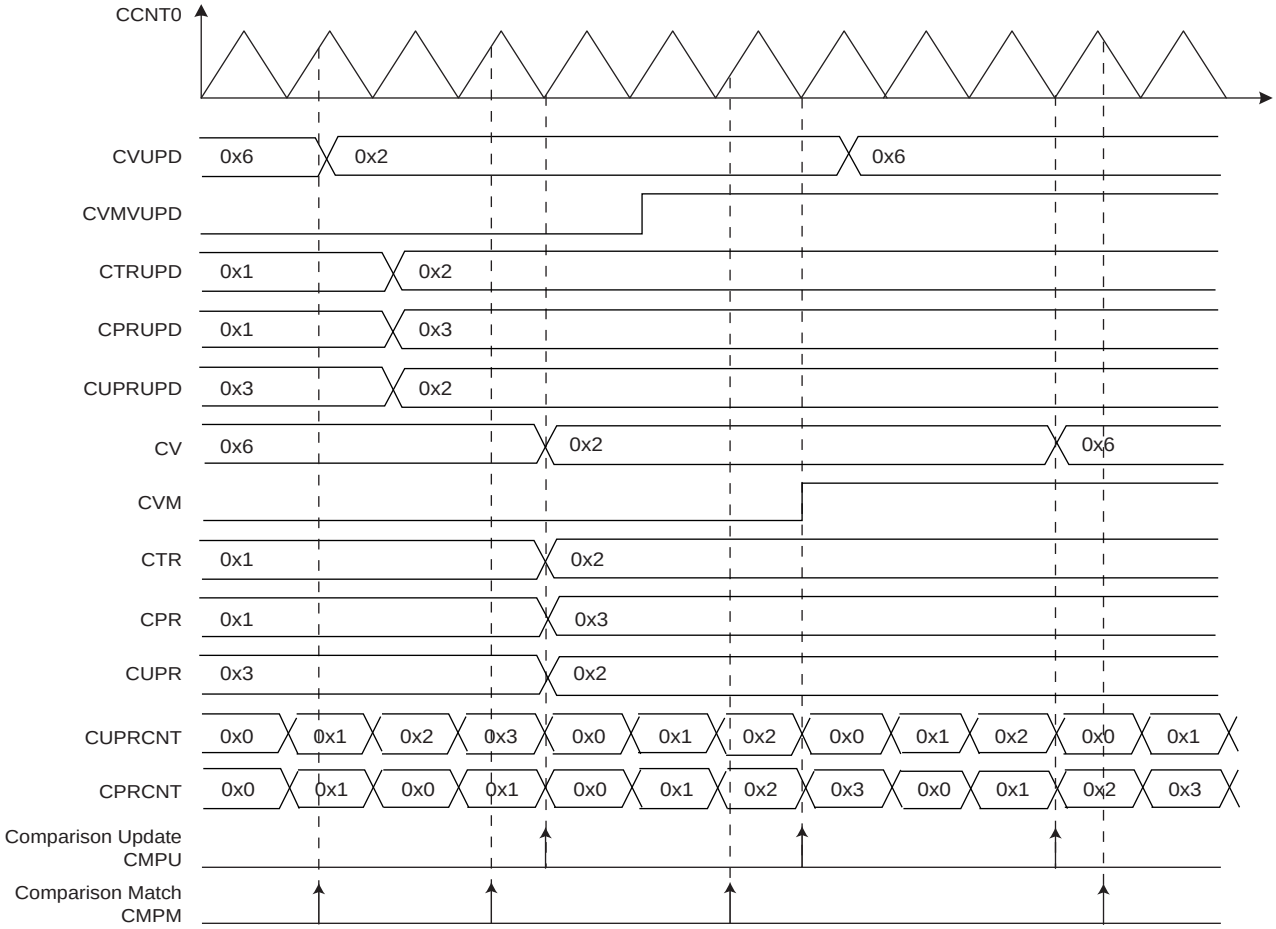
The update of the comparison x configuration and the comparison x value is triggered periodically after the comparison x update period. It is defined by the field CUPR in the PWM_CMPMx. The comparison unit has an update period counter independent from the period counter to trigger this update. When the 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 38-15. Comparison Waveform

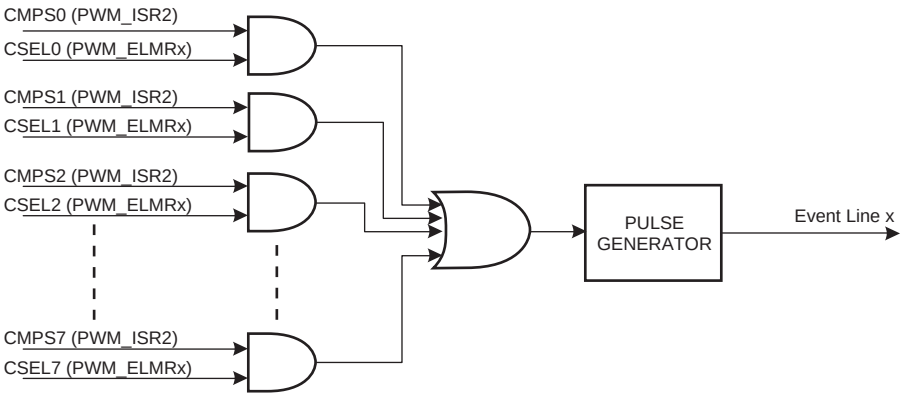


38.6.4 PWM Event Lines

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 38-16. Event Line Block Diagram



38.6.5 PWM Controller Operations

38.6.5.1 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_FPE/2)
- 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)

38.6.5.2 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.

38.6.5.3 Changing the Duty-Cycle, the Period and the Dead-Times

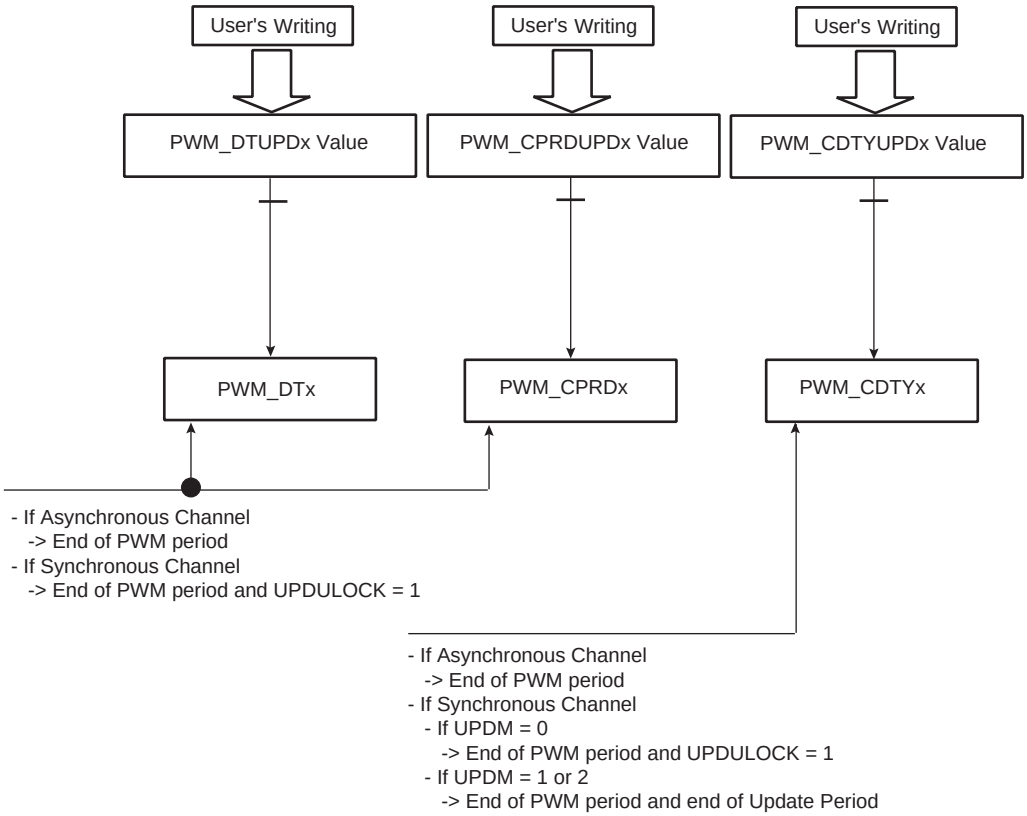
It is possible to modulate the output waveform duty-cycle, period and dead-times.

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 38-17. Synchronized Period, Duty-Cycle and Dead-Times Update



38.6.5.4 Changing the Synchronous Channels Update Period

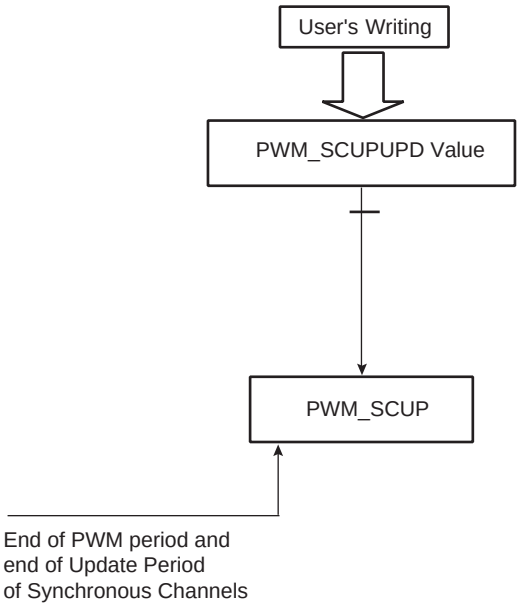
It is possible to change the update period of synchronous channels while they are enabled. (See “Method 2: Manual write of duty-cycle values and automatic trigger of the update” on page 988 and “Method 3: Automatic write of duty-cycle values and automatic trigger of the update” on page 990.)

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.

Note: If the update register PWM_SCUPUPD is written several times between two updates, only the last written value is taken into account.

Note: 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 38-18. Synchronized Update of Update Period Value of Synchronous Channels



38.6.5.5 Changing the Comparison Value and the Comparison Configuration

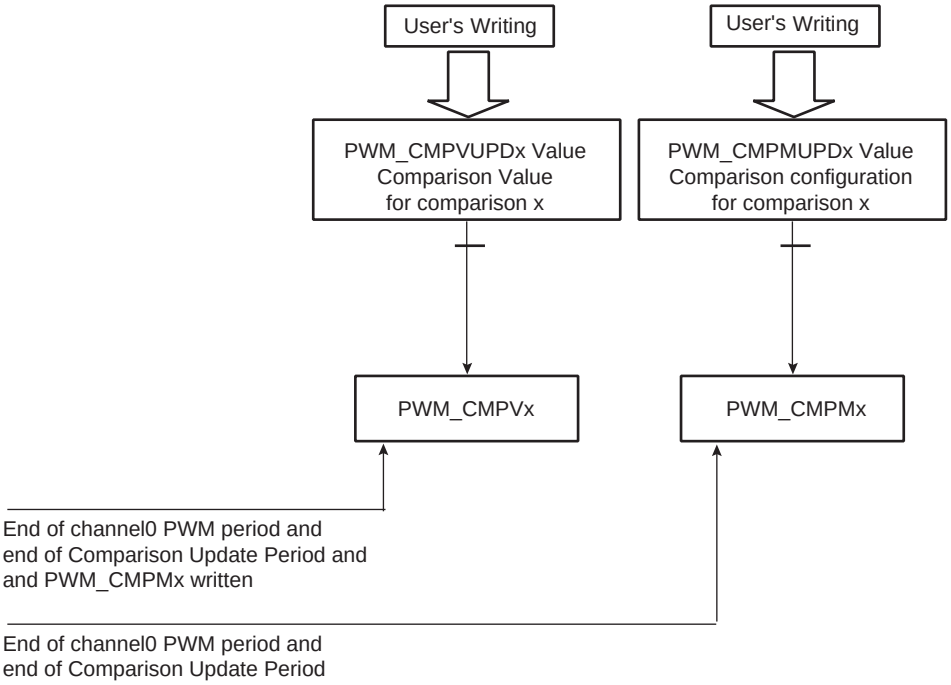
It is possible to change the comparison values and the comparison configurations while the channel 0 is enabled (see Section 38.6.3 “PWM Comparison Units”).

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 38-19. Synchronized Update of Comparison Values and Configurations



38.6.5.6 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.

38.6.5.7 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 1037 (PWM_WPCR). They are divided into 6 groups:

- Register group 0:
  - “PWM Clock Register” on page 1006
- Register group 1:
  - “PWM Disable Register” on page 1008
- Register group 2:
  - “PWM Sync Channels Mode Register” on page 1014
  - “PWM Channel Mode Register” on page 1044
  - “PWM Stepper Motor Mode Register” on page 1036
- Register group 3:
  - “PWM Channel Period Register” on page 1048
  - “PWM Channel Period Update Register” on page 1049
- Register group 4:
  - “PWM Channel Dead Time Register” on page 1051
  - “PWM Channel Dead Time Update Register” on page 1052
- Register group 5:
  - “PWM Fault Mode Register” on page 1029
  - “PWM Fault Protection Value Register” on page 1032
  - “PWM Fault Protection Enable Register 1” on page 1033
  - “PWM Fault Protection Enable Register 2” on page 1034

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 1039 (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 WPVSRC 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.

38.7 Pulse Width Modulation (PWM) User Interface

Table 38-6. 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	PWM_FPV	Read-write	0x0
0x6C	PWM Fault Protection Enable Register 1	PWM_FPE1	Read-write	0x0
0x70	PWM Fault Protection Enable Register 2	PWM_FPE2	Read-write	0x0
0x74-0x78	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-AC	Reserved	–	–	–
0xB0	PWM Stepper Motor Mode Register	PWM_SMMR	Read-write	0x0

Table 38-6. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0xB4-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	–
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

Table 38-6. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x1AC	PWM Comparison 7 Mode Update Register	PWM_CMPMUPD7	Write-only	–
0x1B0 - 0x1FC	Reserved	–	–	–
0x200 + ch_num * 0x20 + 0x00	PWM Channel Mode Register(1)	PWM_CMR	Read-write	0x0
0x200 + ch_num * 0x20 + 0x04	PWM Channel Duty Cycle Register(1)	PWM_CDTY	Read-write	0x0
0x200 + ch_num * 0x20 + 0x08	PWM Channel Duty Cycle Update Register(1)	PWM_CDTYUPD	Write-only	–
0x200 + ch_num * 0x20 + 0x0C	PWM Channel Period Register(1)	PWM_CPRD	Read-write	0x0
0x200 + ch_num * 0x20 + 0x10	PWM Channel Period Update Register(1)	PWM_CPRDUPD	Write-only	–
0x200 + ch_num * 0x20 + 0x14	PWM Channel Counter Register(1)	PWM_CCNT	Read-only	0x0
0x200 + ch_num * 0x20 + 0x18	PWM Channel Dead Time Register(1)	PWM_DT	Read-write	0x0
0x200 + ch_num * 0x20 + 0x1C	PWM Channel Dead Time Update Register(1)	PWM_DTUPD	Write-only	–

Notes: 1. Some registers are indexed with “ch_num” index ranging from 0 to 7.

38.7.1 PWM Clock Register

Name: PWM_CLK

Address: 0x40094000

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 1039.

- **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

38.7.2 PWM Enable Register

Name: PWM_ENA

Address: 0x40094004

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
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = No effect.

1 = Enable PWM output for channel x.

38.7.3 PWM Disable Register

Name: PWM_DIS

Address: 0x40094008

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
CHID7	CHID6	CHID5	CHID4	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 1039.

- **CHIDx: Channel ID**

0 = No effect.

1 = Disable PWM output for channel x.

38.7.4 PWM Status Register

Name: PWM_SR

Address: 0x4009400C

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
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = PWM output for channel x is disabled.

1 = PWM output for channel x is enabled.

38.7.5 PWM Interrupt Enable Register 1

Name: PWM_IER1

Address: 0x40094010

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
FCHID7	FCHID6	FCHID5	FCHID4	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Counter Event on Channel x Interrupt Enable**
- **FCHIDx: Fault Protection Trigger on Channel x Interrupt Enable**

38.7.6 PWM Interrupt Disable Register 1

Name: PWM_IDR1

Address: 0x40094014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
FCHID7	FCHID6	FCHID5	FCHID4	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Counter Event on Channel x Interrupt Disable**
- **FCHIDx: Fault Protection Trigger on Channel x Interrupt Disable**

38.7.7 PWM Interrupt Mask Register 1

Name: PWM_IMR1

Address: 0x40094018

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
FCHID7	FCHID6	FCHID5	FCHID4	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Counter Event on Channel x Interrupt Mask**
- **FCHIDx: Fault Protection Trigger on Channel x Interrupt Mask**

38.7.8 PWM Interrupt Status Register 1

Name: PWM_ISR1

Address: 0x4009401C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
FCHID7	FCHID6	FCHID5	FCHID4	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: 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.

- **FCHIDx: 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 CHIDx and FCHIDx flags.

38.7.9 PWM Sync Channels Mode Register

Name: PWM_SCM

Address: 0x40094020

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
SYNC7	SYNC6	SYNC5	SYNC4	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 1039.

- **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(1)
1	MODE1	Manual write of double buffer registers and automatic update of synchronous channels(2)
2	MODE2	Automatic write of duty-cycle update registers by the PDC and automatic update of synchronous channels(2)
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 “PWM Interrupt Status Register 2” on page 1022 and the PDC transfer request are never set to 1.
1	x	The WRDY flag in “PWM Interrupt Status Register 2” on page 1022 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 “PWM Interrupt Status Register 2” on page 1022 and the PDC transfer request are set to 1 as soon as the update period is elapsed.
	1	The WRDY flag in “PWM Interrupt Status Register 2” on page 1022 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.

38.7.10 PWM Sync Channels Update Control Register

Name: PWM_SCUC

Address: 0x40094028

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 1014, 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.

38.7.11 PWM Sync Channels Update Period Register

Name: PWM_SCUP

Address: 0x4009402C

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 1014). This time is equal to UPR+1 periods of the synchronous channels.

- **UPRCNT: Update Period Counter**

Reports the value of the Update Period Counter.

38.7.12 PWM Sync Channels Update Period Update Register

Name: PWM_SCUPUPD

Address: 0x40094030

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 1014). This time is equal to UPR+1 periods of the synchronous channels.

38.7.13 PWM Interrupt Enable Register 2

Name: PWM_IER2

Address: 0x40094034

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**

38.7.14 PWM Interrupt Disable Register 2

Name: PWM_IDR2

Address: 0x40094038

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**

38.7.15 PWM Interrupt Mask Register 2

Name: PWM_IMR2

Address: 0x4009403C

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**

38.7.16 PWM Interrupt Status Register 2

Name: PWM_ISR2

Address: 0x40094040

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.

38.7.17 PWM Output Override Value Register

Name: PWM_OOV

Address: 0x40094044

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
OOVL7	OOVL6	OOVL5	OOVL4	OOVL3	OOVL2	OOVL1	OOVL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
OOVH7	OOVH6	OOVH5	OOVH4	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.

38.7.18 PWM Output Selection Register

Name: PWM_OS

Address: 0x40094048

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
OSL7	OSL6	OSL5	OSL4	OSL3	OSL2	OSL1	OSL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
OSH7	OSH6	OSH5	OSH4	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.

38.7.19 PWM Output Selection Set Register

Name: PWM_OSS

Address: 0x4009404C

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
OSSL7	OSSL6	OSSL5	OSSL4	OSSL3	OSSL2	OSSL1	OSSL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
OSSH7	OSSH6	OSSH5	OSSH4	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.

38.7.20 PWM Output Selection Clear Register

Name: PWM_OSC

Address: 0x40094050

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
OSCL7	OSCL6	OSCL5	OSCL4	OSCL3	OSCL2	OSCL1	OSCL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
OSCH7	OSCH6	OSCH5	OSCH4	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.

38.7.21 PWM Output Selection Set Update Register

Name: PWM_OSSUPD

Address: 0x40094054

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
OSSUPL7	OSSUPL6	OSSUPL5	OSSUPL4	OSSUPL3	OSSUPL2	OSSUPL1	OSSUPL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
OSSUPH7	OSSUPH6	OSSUPH5	OSSUPH4	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.

38.7.22 PWM Output Selection Clear Update Register

Name: PWM_OSCUPD

Address: 0x40094058

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
OSCUPL7	OSCUPDL6	OSCUPL5	OSCUPL4	OSCUPL3	OSCUPL2	OSCUPL1	OSCUPL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
OSCUPH7	OSCUPH6	OSCUPH5	OSCUPH4	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.

38.7.23 PWM Fault Mode Register

Name: PWM_FMR

Address: 0x4009405C

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 1039.

- **FPOL: Fault Polarity (fault input bit varies from 0 to 5)**

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 5)**

For each field bit y (fault input number):

0 = The fault y is active until the Fault condition is removed at the peripheral(1) level.

1 = The fault y stays active until the Fault condition is removed at the peripheral(1) 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 5)**

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 1030, the bit FMODy can be set to “1” only if the FPOLy bit has been previously configured to its final value.

38.7.24 PWM Fault Status Register

Name: PWM_FSR

Address: 0x40094060

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
FS							

7	6	5	4	3	2	1	0
FIV							

- **FIV: Fault Input Value (fault input bit varies from 0 to 5)**

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 5)**

For each field bit y (fault input number):

0 = The fault y is not currently active.

1 = The fault y is currently active.

38.7.25 PWM Fault Clear Register

Name: PWM_FCR

Address: 0x40094064

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 5)**

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 1029), else writing this bit to 1 has no effect.

38.7.26 PWM Fault Protection Value Register

Name: PWM_FPV

Address: 0x40094068

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
FPVL7	FPVL6	FPVL5	FPVL4	FPVL3	FPVL2	FPVL1	FPVL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
FPVH7	FPVH6	FPVH5	FPVH4	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 1039.

- **FPVHx: Fault Protection Value for PWMH output on channel x**

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**

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.

38.7.27 PWM Fault Protection Enable Register 1

Name: PWM_FPE1

Address: 0x4009406C

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 1039.

Only the first 6 bits (number of fault input pins) of fields FPE0, FPE1, FPE2 and FPE3 are significant.

- **FPEx: Fault Protection Enable for channel x (fault input bit varies from 0 to 5)**

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 FPEx 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 1029.

38.7.28 PWM Fault Protection Enable Register 2

Name: PWM_FPE2

Address: 0x40094070

Access: Read-write

31	30	29	28	27	26	25	24
FPE7							

23	22	21	20	19	18	17	16
FPE6							

15	14	13	12	11	10	9	8
FPE5							

7	6	5	4	3	2	1	0
FPE4							

This register can only be written if the bits WPSWS5 and WPHWS5 are cleared in “PWM Write Protect Status Register” on page 1039.

Only the first 6 bits (number of fault input pins) of fields FPE4, FPE5, FPE6 and FPE7 are significant.

- **FPEx: Fault Protection Enable for channel x (fault input bit varies from 0 to 5)**

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 FPEx 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 1029.

38.7.29 PWM Event Line x Register

Name: PWM_ELMRx

Address: 0x4009407C

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.

38.7.30 PWM Stepper Motor Mode Register

Name: PWM_SMMR

Address: 0x400940B0

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	DOWN3	DOWN2	DOWN1	DOWN0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	GCEN3	GCEN2	GCEN1	GCEN0

- **GCENx: Gray Count ENable**

0 = Disable gray count generation on PWML[2*x], PWMH[2*x], PWML[2*x +1], PWMH[2*x +1]

1 = enable gray count generation on PWML[2*x], PWMH[2*x], PWML[2*x +1], PWMH[2*x +1.

- **DOWNx: DOWN Count**

0 = Up counter.

1 = Down counter.

38.7.31 PWM Write Protect Control Register

Name: PWM_WPCR

Address: 0x400940E4

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 1006
- Register group 1:
  - “PWM Disable Register” on page 1008
- Register group 2:
  - “PWM Sync Channels Mode Register” on page 1014
  - “PWM Channel Mode Register” on page 1044
  - “PWM Stepper Motor Mode Register” on page 1036

- Register group 3:
  - “PWM Channel Period Register” on page 1048
  - “PWM Channel Period Update Register” on page 1049
- Register group 4:
  - “PWM Channel Dead Time Register” on page 1051
  - “PWM Channel Dead Time Update Register” on page 1052
- Register group 5:
  - “PWM Fault Mode Register” on page 1029
  - “PWM Fault Protection Value Register” on page 1032
  - “PWM Fault Protection Enable Register 1” on page 1033
  - “PWM Fault Protection Enable Register 2” on page 1034

38.7.32 PWM Write Protect Status Register

Name: PWM_WPSR

Address: 0x400940E8

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

- **WPSWSx: 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.

- **WPHWSx: 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.

Note: The two LSBs of the address offset of the write-protected register are not reported

Note: Reading PWM_WPSR automatically clears WPVS and WPVSRC fields.

38.7.33 PWM Comparison x Value Register

Name: PWM_CMPVx

Address: 0x40094130 [0], 0x40094140 [1], 0x40094150 [2], 0x40094160 [3], 0x40094170 [4], 0x40094180 [5], 0x40094190 [6], 0x400941A0 [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 1044)

38.7.34 PWM Comparison x Value Update Register

Name: PWM_CMPVUPDx

Address: 0x40094134 [0], 0x40094144 [1], 0x40094154 [2], 0x40094164 [3], 0x40094174 [4], 0x40094184 [5], 0x40094194 [6], 0x400941A4 [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 1044)

CAUTION: to be taken into account, the write of the register PWM_CMPVUPDx must be followed by a write of the register PWM_CMPMUPDx.

38.7.35 PWM Comparison x Mode Register

Name: PWM_CMPMx

Address: 0x40094138, 0x40094148, 0x40094158, 0x40094168, 0x40094178, 0x40094188, 0x40094198,0x400941A8

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

38.7.36 PWM Comparison x Mode Update Register

Name: PWM_CMPMUPDx

Address: 0x4009413C [0], 0x4009414C [1], 0x4009415C [2], 0x4009416C [3], 0x4009417C [4], 0x4009418C [5], 0x4009419C [6], 0x400941AC [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.

38.7.37 PWM Channel Mode Register

Name: PWM_CMRx [x=0..7]

Address: 0x40094200 [0], 0x40094220 [1], 0x40094240 [2], 0x40094260 [3], 0x40094280 [4], 0x400942A0 [5], 0x400942C0 [6], 0x400942E0 [7]

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
–	–	–	–	–	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 1039.

- **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 1013).

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.

- **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.

38.7.38 PWM Channel Duty Cycle Register

Name: PWM_CDTYx [x=0..7]

Address: 0x40094204 [0], 0x40094224 [1], 0x40094244 [2], 0x40094264 [3], 0x40094284 [4], 0x400942A4 [5], 0x400942C4 [6], 0x400942E4 [7]

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).

38.7.39 PWM Channel Duty Cycle Update Register

Name: PWM_CDTYUPDx [x=0..7]

Address: 0x40094208 [0], 0x40094228 [1], 0x40094248 [2], 0x40094268 [3], 0x40094288 [4], 0x400942A8 [5], 0x400942C8 [6], 0x400942E8 [7]

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).

38.7.40 PWM Channel Period Register

Name: PWM_CPRDx [x=0..7]

Address: 0x4009420C [0], 0x4009422C [1], 0x4009424C [2], 0x4009426C [3], 0x4009428C [4], 0x400942AC [5], 0x400942CC [6], 0x400942EC [7]

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 1039.

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:

$$\frac{(CPRD \times DIVA)}{MCK} \text{ or } \frac{(CPRD \times DIVB)}{MCK}$$

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:

$$\frac{(2 \times CPRD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRD \times DIVB)}{MCK}$$

38.7.41 PWM Channel Period Update Register

Name: PWM_CPRDUPDx [x=0..7]

Address: 0x40094210 [0], 0x40094230 [1], 0x40094250 [2], 0x40094270 [3], 0x40094290 [4], 0x400942B0 [5], 0x400942D0 [6], 0x400942F0 [7]

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 1039.

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:

$$\frac{(2 \times CPRDUPD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRDUPD \times DIVB)}{MCK}$$

38.7.42 PWM Channel Counter Register

Name: PWM_CCNTx [x=0..7]

Address: 0x40094214 [0], 0x40094234 [1], 0x40094254 [2], 0x40094274 [3], 0x40094294 [4], 0x400942B4 [5], 0x400942D4 [6], 0x400942F4 [7]

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.

38.7.43 PWM Channel Dead Time Register

Name: PWM_DTx [x=0..7]

Address: 0x40094218 [0], 0x40094238 [1], 0x40094258 [2], 0x40094278 [3], 0x40094298 [4], 0x400942B8 [5], 0x400942D8 [6], 0x400942F8 [7]

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 1039.

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).

38.7.44 PWM Channel Dead Time Update Register

Name: PWM_DTUPDx [x=0..7]

Address: 0x4009421C [0], 0x4009423C [1], 0x4009425C [2], 0x4009427C [3], 0x4009429C [4], 0x400942BC [5], 0x400942DC [6], 0x400942FC [7]

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 1039.

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 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). This value is applied only at the beginning of the next channel x PWM period.

- **DTLUPD: Dead-Time Value Update for PWMLx Output**

Defines the dead-time value for PWMLx output. This value must be defined between 0 and CDTY (PWM_CDTYx). This value is applied only at the beginning of the next channel x PWM period.

39. USB On-The-Go Interface (UOTGHS)

39.1 Description

The Universal Serial Bus (USB) MCU device complies with the Universal Serial Bus (USB) 2.0 specification in all speeds.

Each pipe/endpoint can be configured in one of several USB transfer types. It can be associated with one, two or three banks of a DPRAM used to store the current data payload. If two or three banks are used, then one DPRAM bank is read or written by the CPU or the DMA, while the other is read or written by the UOTGHS core. This feature is mandatory for isochronous pipes/endpoints.

Table 39-1 on page 1053 describes the hardware configuration of the USB MCU device.

Table 39-1. Description of USB Pipes/Endpoints

Pipe/Endpoint	Mnemonic	Max. Nb. Banks	DMA	High Band Width	Max. Pipe/ Endpoint Size	Type
0	PEP_0	1	N	N	64	Control
1	PEP_1	3	Y	1	1024	Isochronous/Bulk/Interrupt/Control
2	PEP_2	3	Y	Y	1024	Isochronous/Bulk/Interrupt/Control
3	PEP_3	2	Y	Y	1024	Isochronous/Bulk/Interrupt/Control
4	PEP_4	2	Y	Y	1024	Isochronous/Bulk/Interrupt/Control
5	PEP_5	2	Y	Y	1024	Isochronous/Bulk/Interrupt/Control
6	PEP_6	2	Y	Y	1024	Isochronous/Bulk/Interrupt/Control
7	PEP_7	2	N	Y	1024	Isochronous/Bulk/Interrupt/Control
8	PEP_8	2	_	Y	1024	Isochronous/Bulk/Interrupt/Control
9	PEP_9	2	_	Y	1024	Isochronous/Bulk/Interrupt/Control

Note: Bit fields are presented throughout the document as follows: ‘REGISTER. FIELD’ (e.g. UOTGHS_CTRL.USBE)

39.2 Embedded Characteristics

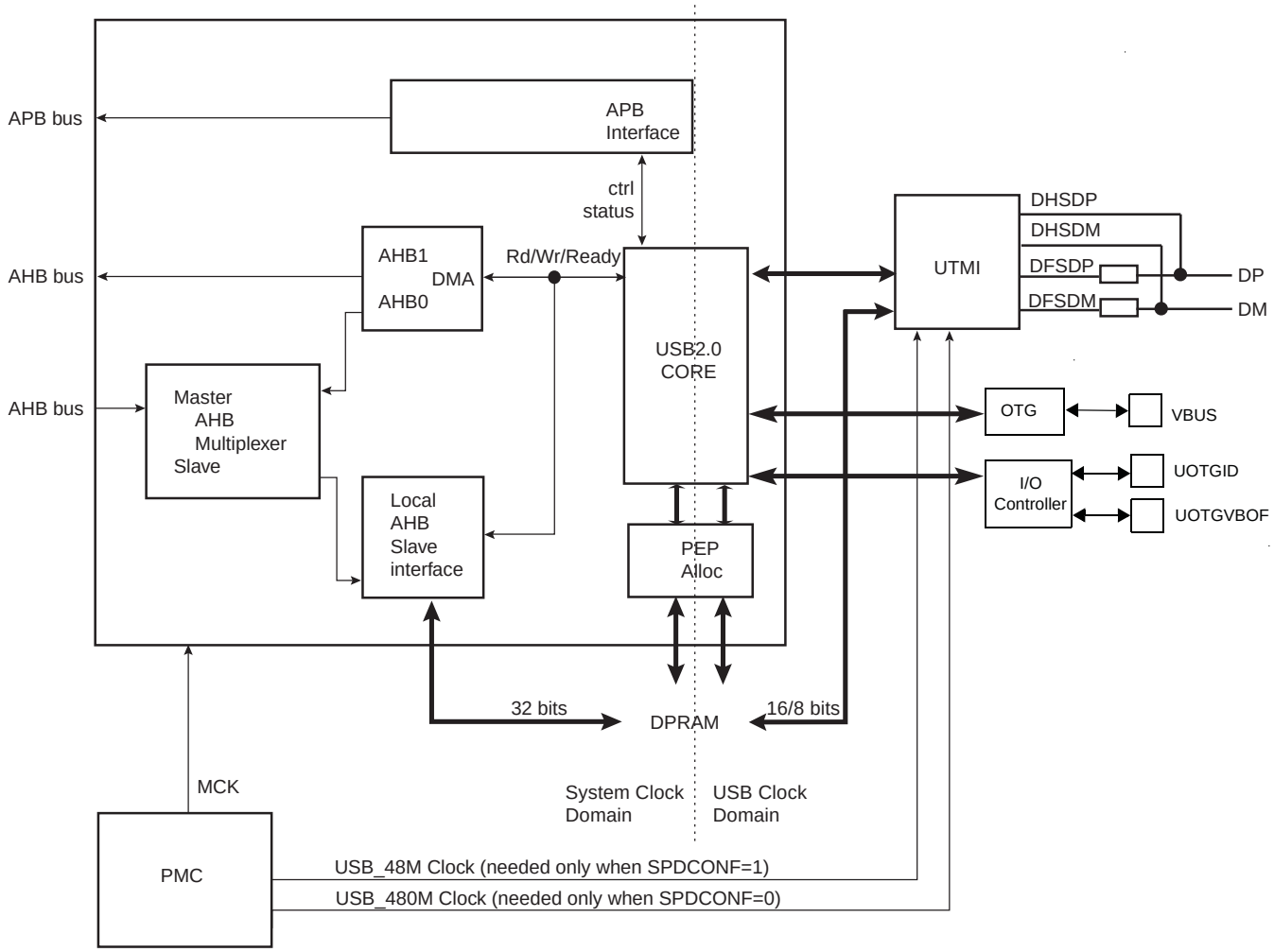
- Compatible with the USB 2.0 specification
- Supports High (480 Mbps), Full (12 Mbps) and Low (1.5 Mbps) speed communication and On-The-Go
- 10 pipes/endpoints
- 4096 bytes of Embedded Dual-Port RAM (DPRAM) for Pipes/Endpoints
- Up to 3 memory banks per Pipe/Endpoint (Not for Control Pipe/Endpoint)
- Flexible Pipe/Endpoint configuration and management with dedicated DMA channels
- On-Chip UTMI transceiver including Pull-Ups/Pull-downs
- On-Chip OTG pad including VBUS analog comparator

39.3 Block Diagram

The UOTGHS provides a hardware device to interface a USB link to a data flow stored in a dual-port RAM (DPRAM).

In normal operation (SPDCONF = 1), the UTMI transceiver requires the UTMI PLL (480 MHz). In case of Full or Low speed only, for a lower consumption (SPDCONF = 0), the UTMI transceiver only requires 48 MHz.

Figure 39-1. UOTGHS Block Diagram



39.3.1 Application Block Diagram

39.3.1.1 Device Mode

Bus-Powered Device

Figure 39-2. Bus-Powered Device Application Block Diagram

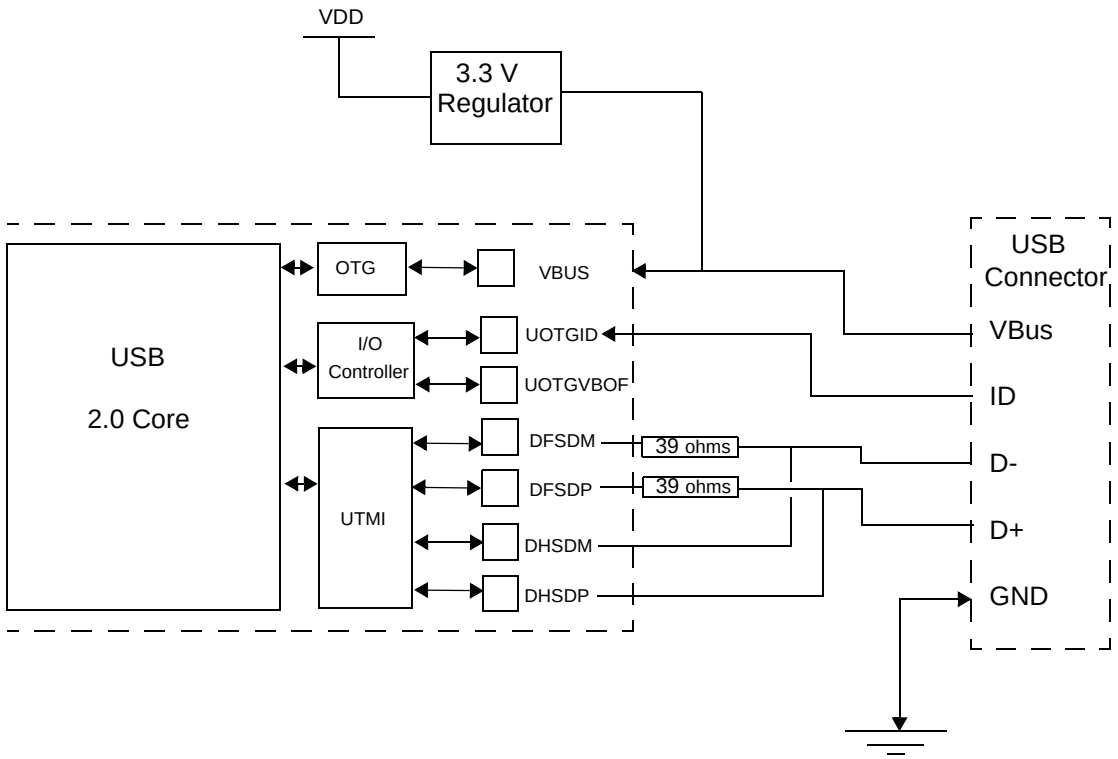
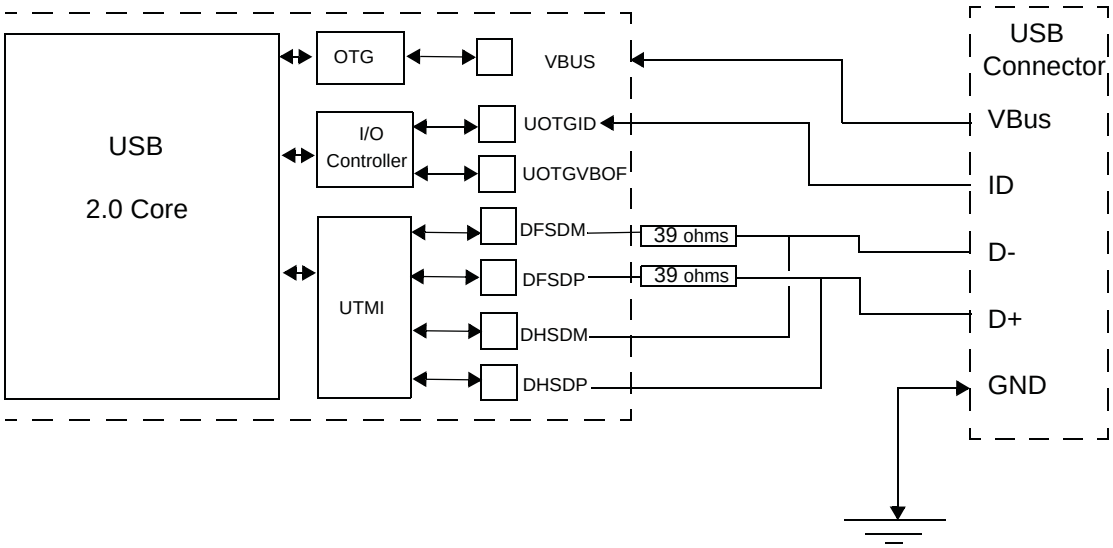
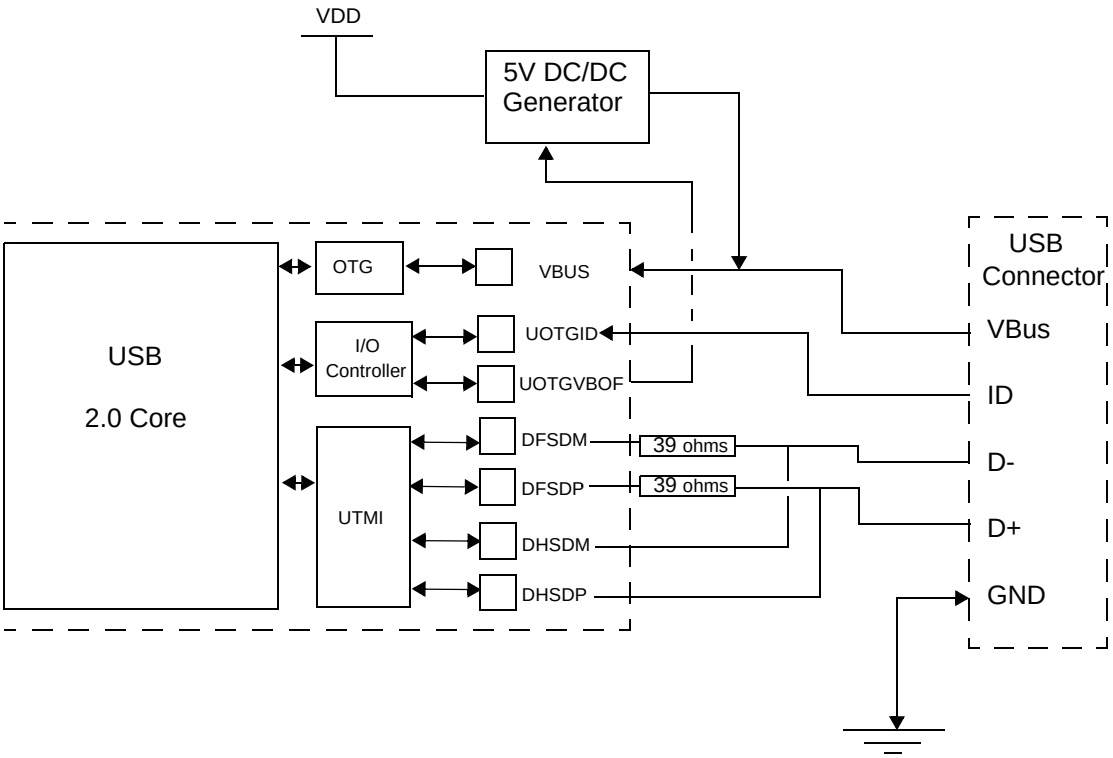


Figure 39-3. Self-powered Device Application Block Diagram



39.3.1.2 Host and OTG Modes

Figure 39-4. Host and OTG Application Block Diagram



39.3.2 I/O Lines Description

Table 39-2. I/O Lines Description

PIn Name	Pin Description	Type	Active Level
UOTGVBOF	USB VBus On/Off: Bus Power Control Port	Output	VBUSPO
VBUS	VBus: Bus Power Measurement Port	Input	
D-	Data -: Differential Data Line - Port	Input/Output	
D+	Data +: Differential Data Line + Port	Input/Output	
DFSDM	FS Data -: Full-Speed Differential Data Line - Port	Input/Output	
DFSDP	FS Data +: Full-Speed Differential Data Line + Port	Input/Output	
DHSDM	HS Data -: Hi-Speed Differential Data Line - Port	Input/Output	
DHSDP	HS Data +: Hi-Speed Differential Data Line + Port	Input/Output	
UOTGID	USB Identification: Mini Connector Identification Port	Input	Low: Mini-A plug High Z: Mini-B plug

39.4 Product Dependencies

In order to use this module, other parts of the system must be configured correctly, as described below.

39.4.1 I/O Lines

The UOTGVBOF and UOTGID pins are multiplexed with I/O Controller lines and may also be multiplexed with lines of other peripherals. In order to use them with the USB, the user must first configure the I/O Controller to assign them to their USB peripheral functions.

- If UOTGID is used, the I/O Controller must be configured to enable the internal pull-up resistor of its pin.
- If UOTGVBOF or UOTGID is not used by the application, the corresponding pin can be used for other purposes by the I/O Controller or by other peripherals.

Table 39-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
UOTGHS	UOTGID	PB11	A
UOTGHS	UOTGVBOF	PB10	A

39.4.2 Clocks

The clock for the UOTGHS bus interface is generated by the Power Management Controller. This clock can be enabled or disabled in the Power Management Controller. It is recommended to disable the UOTGHS before disabling the clock, to avoid freezing the UOTGHS in an undefined state.

The UOTGHS can work in two different modes:

- Normal mode (SPDCONF =1) where High speed, Full speed and Low speed are available.
- Low-power mode (SPDCONF =0) where Full speed and Low speed are available.

For normal mode:

1. Enable the UPLL 480 MHz
2. Wait for the UPLL 480 MHz to be considered as locked by the PMC
3. Enable the UOTGHS Peripheral clock (PMC_PCER)
4. The UOTGHS will use USB_480 M clock (refer to the PMC).

For low-power mode:

1. As USB_48M must be set to 48 MHz (refer to the PMC), select either the PLLA or the UPLL (previously set to ON), and program the PMC_USB register (source selection and divider)
2. Enable the UOTGCK bit (PMC_SCER)
3. Enable the UOTGHS Peripheral clock (PMC_PCER)
4. Put the UOTGHS in Low power mode (SPDCONF)

39.4.3 Interrupts

The UOTGHS interrupt request line is connected to the interrupt controller. Using the UOTGHS interrupt requires the interrupt controller to be programmed first.

39.4.4 USB Pipe/Endpoint x FIFO Data Register (USBFIFOxDATA)

The application has access to each Pipe/Endpoint FIFO through its reserved 32 KB address space. The application can access a 64 KB buffer linearly or fixedly as the DPRAM address increment is fully handled by hardware. Byte, half-word and word access are supported. Data should be accessed in a big-endian way.

Disabling the UOTGHS (by writing a zero to the UOTGHS_CTRL.USBE bit) does not reset the DPRAM.

39.5 Functional Description

39.5.1 USB General Operation

39.5.1.1 Introduction

After a hardware reset, the UOTGHS is disabled. When enabled, the UOTGHS runs either in device mode or in host mode according to the ID detection.

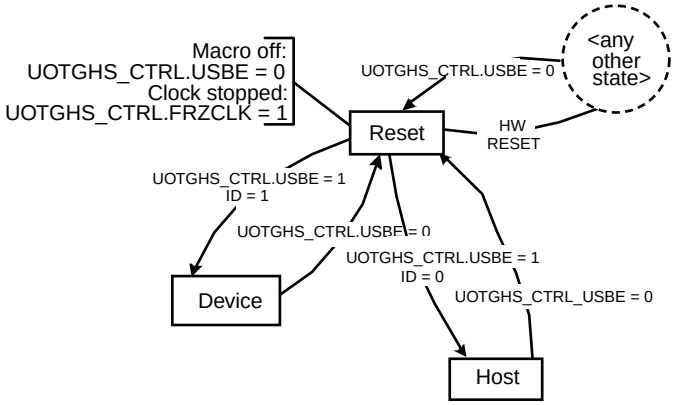
If the UOTGID pin is not connected to the ground, the UOTGID Pin State bit in the General Status register (UOTGHS_SR.ID) is set (the internal pull-up resistor of the UOTGID pin must be enabled by the I/O Controller) and device mode is engaged.

The UOTGHS_SR.ID bit is cleared when a low level has been detected on the UOTGID pin. Host mode is then engaged.

39.5.1.2 Power-On and Reset

Figure 39-5 on page 1058 describes the UOTGHS main states.

Figure 39-5. General States



After a hardware reset, the UOTGHS is in the Reset state. In this state:

- The macro is disabled. The UOTGHS Enable bit in the General Control register (UOTGHS_CTRL.USBE) is zero.
- The macro clock is stopped in order to minimize the power consumption. The Freeze USB Clock bit (UOTGHS_CTRL.FRZCLK) is set.
- The UTMI is in suspend mode.
- The internal states and registers of the device and host modes are reset.
- The DPRAM is not cleared and is accessible.
- The UOTGHS_SR.ID and UOTGHS_SR.VBUS bits reflect the states of the UOTGID and VBUS input pins.
- The OTG Pad Enable (UOTGHS_CTRL.OTGPADE) bit, the VBus Polarity (UOTGHS_CTRL.VBUSPO) bit, the UOTGHS_CTRL.FRZCLK bit, the UOTGHS_CTRL.USBE bit, the UOTGID Pin Enable (UOTGHS_CTRL.UIDE) bit, the UOTGHS Mode (UOTGHS_CTRL.UIMOD) bit, and the Low-Speed Mode Force (UOTGHS_DEVCTRL.LS) bit can be written by software, so that the user can program pads and speed before enabling the macro, but their value is only taken into account once the macro is enabled and unfrozen.

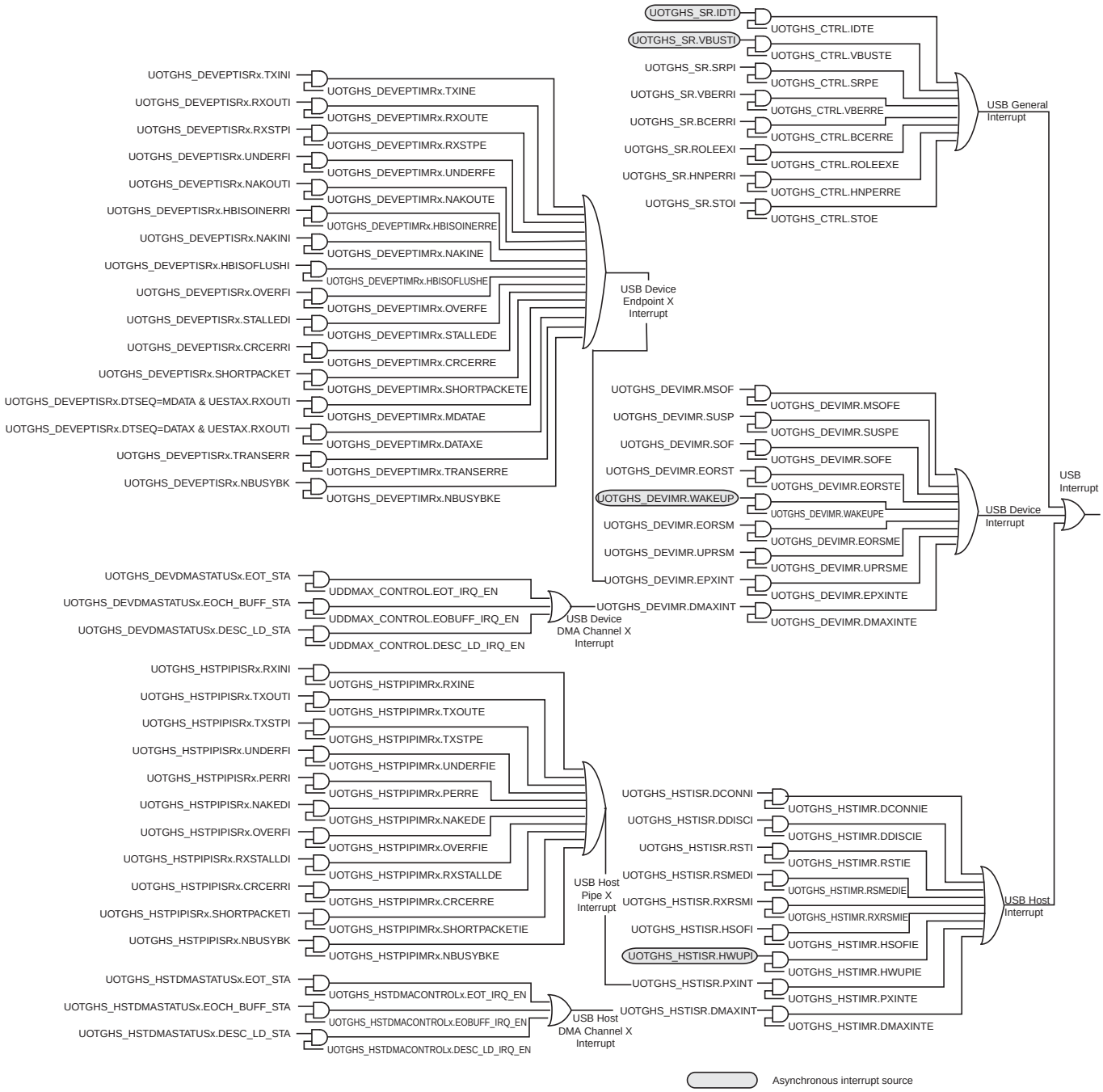
After writing a one to UOTGHS_CTRL.USBE, the UOTGHS enters the Device or the Host mode (according to the ID detection) in idle state.

The UOTGHS can be disabled at any time by writing a zero to UOTGHS_CTRL.USBE. In fact, writing a zero to UOTGHS_CTRL.USBE acts as a hardware reset, except that the UOTGHS_CTRL.OTGPADE, UOTGHS_CTRL.VBUSPO, UOTGHS_CTRL.FRZCLK, UOTGHS_CTRL.UIDE, UOTGHS_CTRL.UIMOD and, UOTGHS_DEVCTRL.LS bits are not reset.

39.5.1.3 Interrupts

One interrupt vector is assigned to the USB interface. Figure 39-6 on page 1059 shows the structure of the USB interrupt system.

Figure 39-6. Interrupt System



See Section 39.5.2.19 and Section 39.5.3.13 for further details about device and host interrupts.

There are two kinds of general interrupts: processing, i.e. their generation is part of the normal processing, and exception, i.e. errors (not related to CPU exceptions).

The processing general interrupts are:

- The ID Transition Interrupt (UOTGHS_SR.IDTI)
- The VBus Transition Interrupt (UOTGHS_SR.VBUSTI)
- The SRP Interrupt (UOTGHS_SR.SRPI)
- The Role Exchange Interrupt (UOTGHS_SR.ROLEEXI)

The exception general interrupts are:

- The VBus Error Interrupt (UOTGHS_SR.VBERRI)
- The B-Connection Error Interrupt (UOTGHS_SR.BCERRI)
- The HNP Error Interrupt (UOTGHS_SR.HNPERRI)
- The Suspend Time-Out Interrupt (UOTGHS_SR.STOI)

39.5.1.4 MCU Power Modes

USB Suspend Mode

In peripheral mode, the Suspend Interrupt bit in the Device Global Interrupt register (UOTGHS_DEVISR.SUSP) indicates that the USB line is in suspend mode. In this case, the transceiver is automatically set in suspend mode to reduce the consumption.

Clock Frozen

The UOTGHS can be frozen when the USB line is in the suspend mode, by writing a one to the UOTGHS_CTRL.FRZCLK bit, which reduces the power consumption.

In this case, it is still possible to access the following elements:

- The UOTGHS_CTRL.OTGPADE, UOTGHS_CTRL.VBUSPO, UOTGHS_CTRL.FRZCLK, UOTGHS_CTRL.USBE, UOTGHS_CTRL.UIDE, UOTGHS_CTRL.UIMOD and UOTGHS_DEVCTRL.LS bits

Moreover, when UOTGHS_CTRL.FRZCLK is written to one, only the asynchronous interrupt sources may trigger the USB interrupt:

- The ID Transition Interrupt (UOTGHS_SR.IDTI)
- The VBus Transition Interrupt (UOTGHS_SR.VBUSTI)
- The Wake-up Interrupt (UOTGHS_DEVISR.WAKEUP)
- The Host Wake-up Interrupt (UOTGHS_HSTISR.HWUPI)

39.5.1.5 Speed Control

Device mode

When the USB interface is in device mode, the speed selection (full-speed or high-speed) is performed automatically by the UOTGHS during the USB reset according to the host speed capability. At the end of the USB reset, the UOTGHS enables or disables high-speed terminations and pull-up.

It is possible to restraint the UOTGHS to full-speed or low-speed mode by handling the UOTGHS_DEVCTRL.LS and the Speed Configuration (UOTGHS_DEVCTRL.SPDCONF) bits in UOTGHS_DEVCTRL.

Host mode

When the USB interface is in host mode, internal pull-down resistors are connected on both D+ and D- and the interface detects the speed of the connected device, which is reflected by the Speed Status (UOTGHS_SR.SPEED) field.

39.5.1.6 DPRAM Management

Pipes and endpoints can only be allocated in ascending order, from the pipe/endpoint 0 to the last pipe/endpoint to be allocated. The user shall therefore configure them in the same order.

The allocation of a pipe/endpoint x starts when the Endpoint Memory Allocate bit in the Endpoint x Configuration register (UOTGHS_DEVEPTCFGx.ALLOC) is written to one. Then, the hardware allocates a memory area in the DPRAM and inserts it between the x-1 and x+1 pipes/endpoints. The x+1 pipe/endpoint memory window slides up and its data is lost. Note that the following pipe/endpoint memory windows (from x+2) do not slide.

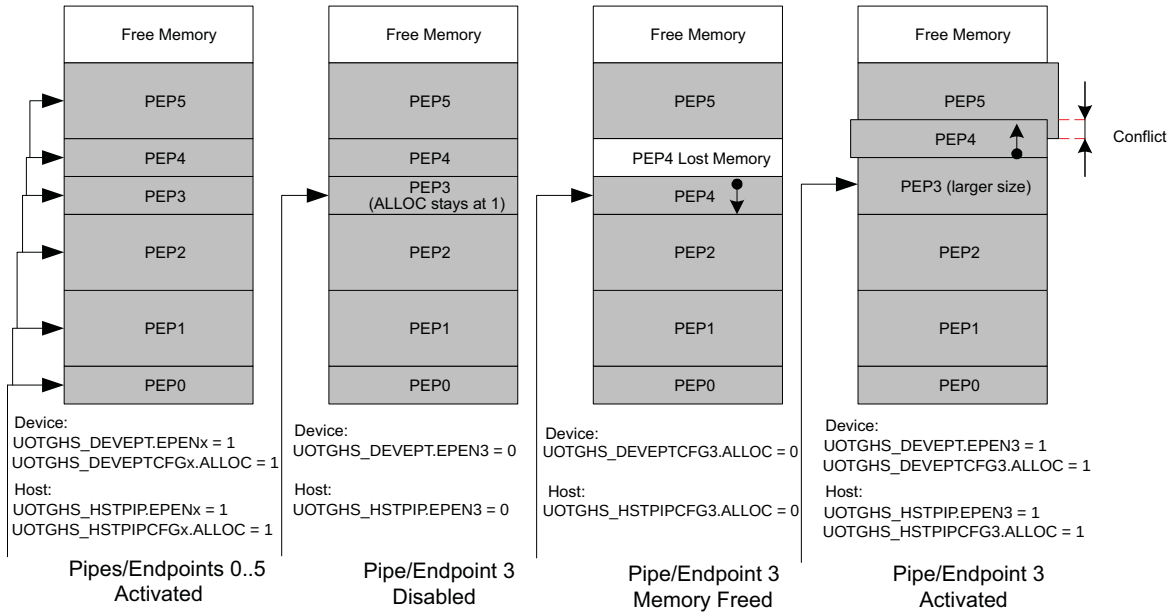
Disabling a pipe, by writing a zero to the Pipe x Enable bit in the Pipe Enable/Reset register (UOTGHS_HSTPIP.PENx), or disabling an endpoint, by writing a zero to the Endpoint x Enable bit in the Device Endpoint register (UOTGHS_DEVEPT.EPENx), resets neither the UOTGHS_DEVEPTCFGx.ALLOC bit nor its configuration:

- the Pipe Banks (UOTGHS_HSTPIPCFGx.PBK) field,
  the Pipe Size (UOTGHS_HSTPIPCFGx.PSIZE) field,
  the Pipe Token (UOTGHS_HSTPIPCFGx.PTOKEN) field,
  the Pipe Type (UOTGHS_HSTPIPCFGx.PTYPE) field,
  the Pipe Endpoint Number (UOTGHS_HSTPIPCFGx.PEPNUM) field,
  and the Pipe Interrupt Request Frequency (UOTGHS_HSTPIPCFGx.INTFRQ) field;
- the Endpoint Banks (UOTGHS_DEVEPTCFGx.EPBK) field,
  the Endpoint Size (UOTGHS_DEVEPTCFGx. EPSIZE) field,
  the Endpoint Direction (UOTGHS_DEVEPTCFGx.EPDIR) field,
  and the Endpoint Type (UOTGHS_DEVEPTCFGx.EPTYPE) field.

To free its memory, the user shall write a zero to the UOTGHS_DEVEPTCFGx.ALLOC bit. The x+1 pipe/endpoint memory window then slides down and its data is lost. Note that the following pipe/endpoint memory windows (from x+2) do not slide.

Figure 39-7 on page 1062 illustrates the allocation and reorganization of the DPRAM in a typical example.

Figure 39-7. Allocation and Reorganization of the DPRAM



1. The pipes/endpoints 0 to 5 are enabled, configured and allocated in ascending order. Each pipe/endpoint then owns a memory area in the DPRAM.
2. The pipe/endpoint 3 is disabled, but its memory is kept allocated by the controller.
3. In order to free its memory, its UOTGHS_DEVEPTCFGx.ALLOC bit is written to zero. The pipe/endpoint 4 memory window slides down, but the pipe/endpoint 5 does not move.
4. If the user chooses to reconfigure the pipe/endpoint 3 with a larger size, the controller allocates a memory area after the pipe/endpoint 2 memory area and automatically slides up the pipe/endpoint 4 memory window. The pipe/endpoint 5 does not move and a memory conflict appears as the memory windows of the pipes/endpoints 4 and 5 overlap. The data of these pipes/endpoints is potentially lost.

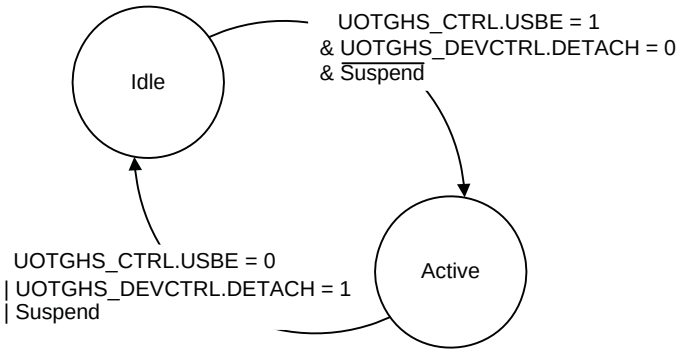
Note:

2. There is no way the data of the pipe/endpoint 0 can be lost (except if it is de-allocated) as the memory allocation and de-allocation may affect only higher pipes/endpoints.
3. Deactivating then reactivating the same pipe/endpoint with the same configuration only modifies temporarily the controller DPRAM pointer and size for this pipe/endpoint. Nothing changes in the DPRAM, higher endpoints seem not to have been moved and their data is preserved as far as nothing has been written or received into them while changing the allocation state of the first pipe/endpoint.
4. When the user writes a one to the UOTGHS_DEVEPTCFGx.ALLOC bit, the Configuration OK Status bit (UOTGHS_DEVEPTISRx.CFGOK) is set only if the configured size and number of banks are correct as compared to the endpoint maximal allowed values and to the maximal FIFO size (i.e. the DPRAM size). The UOTGHS_DEVEPTISRx.CFGOK value does not consider memory allocation conflicts.

39.5.1.7 Pad Suspend

Figure 39-8 on page 1063 shows the pad behavior.

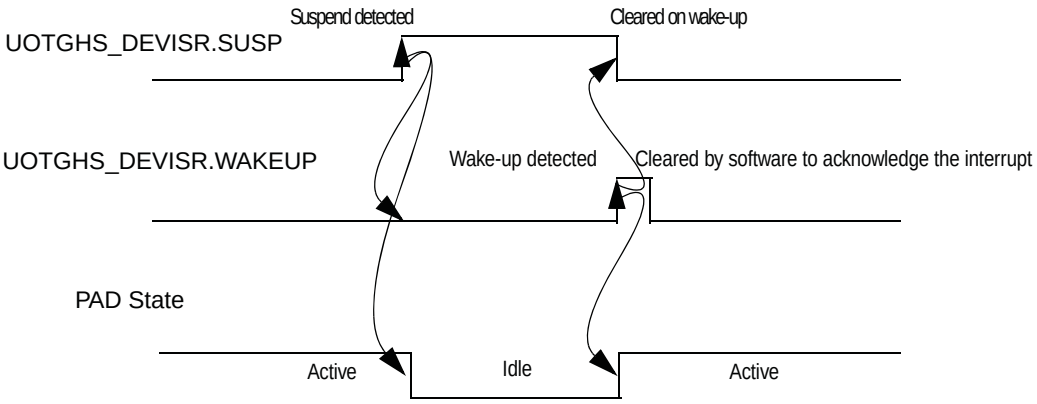
Figure 39-8. Pad Behavior



- In the Idle state, the pad is put in low-power consumption mode, i.e., the differential receiver of the USB pad is off, and internal pull-downs with strong value (15 K) are set in both DP/DM to avoid floating lines.
- In the Active state, the pad is working.

Figure 39-9 on page 1063 illustrates the pad events leading to a PAD state change.

Figure 39-9. Pad Events



The UOTGHS_DEVISR.SUSP bit is set and the Wake-Up Interrupt (UOTGHS_DEVISR.WAKEUP) bit is cleared when a USB “Suspend” state has been detected on the USB bus. This event automatically puts the USB pad in Idle state. The detection of a non-idle event sets UOTGHS_DEVISR.WAKEUP, clears UOTGHS_DEVISR.SUSP and wakes up the USB pad.

The pad goes to the Idle state if the macro is disabled or if the UOTGHS_DEVCTRL.DETACH bit is written to one. It returns to the Active state when UOTGHS_CTRL.USBE is written to one and UOTGHS_DEVCTRL.DETACH is written to zero.

39.5.1.8 Customizing of OTG Timers

It is possible to refine some OTG timers thanks to the Timer Page (UOTGHS_CTRL.TIMPAGE) and Timer Value (UOTGHS_CTRL.TIMVALUE) fields, as shown in Table 39-4 on page 1064.

Table 39-4. Customizing of OTG Timers

		TIMPAGE			
		0b00 AWaitVrise Time-Out (see OTG Standard(1) Section 6.6.5.1)	**0b01 VbBusPulsing Time-Out (see OTG Standard(1) Section5.3.4)**	**0b10 PdTmOutCnt Time-Out (see OTG Standard(1) Section 5.3.2)**	**0b11 SRPDetTmOut Time-Out (see OTG Standard(1) Section 5.3.3)**
TIMVALUE	**00b**	20 ms	15 ms	93 ms	10 µs
	01b	50 ms	23 ms	105 ms	100 µs
	10b	70 ms	31 ms	118 ms	1 ms
	11b	100 ms	40 ms	131 ms	11 ms

Note: 1. “On-The-Go Supplement to the USB 2.0 Specification Revision 1.0a”.

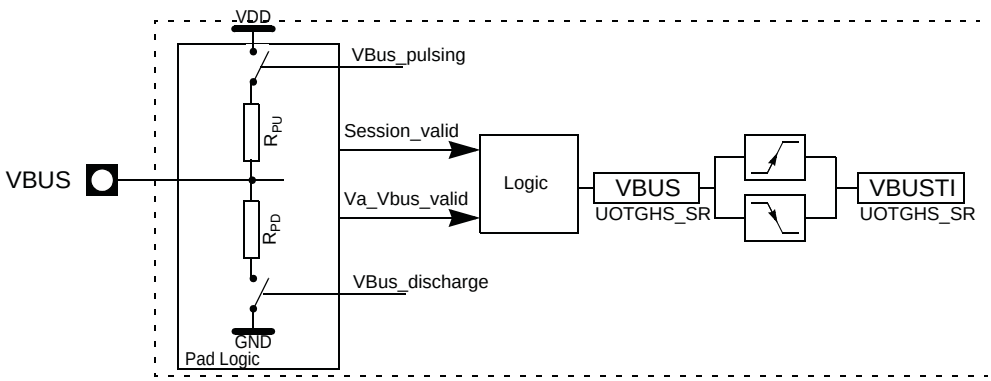
UOTGHS_CTRL.TIMPAGE is used to select the OTG timer to access while UOTGHS_CTRL.TIMVALUE indicates the time-out value of the selected timer.

UOTGHS_CTRL.TIMPAGE and UOTGHS_CTRL.TIMVALUE can be read or written. Before writing them, the user shall unlock write accesses by writing a one to the Timer Access Unlock (UOTGHS_CTRL.UNLOCK) bit. This is not required for read accesses, except before accessing UOTGHS_CTRL.TIMPAGE if it has to be written in order to read the UOTGHS_CTRL.TIMVALUE field of another OTG timer.

39.5.1.9 Plug-In Detection

The USB connection is detected from the VBUS pad. Figure 39-10 on page 1064 shows the architecture of the plug-in detector.

Figure 39-10. Plug-In Detection Input Block Diagram



The control logic of the VBUS pad outputs two signals:

- The Session_valid signal is high when the voltage on the VBUS pad is higher than or equal to 1.4V.
- The Va_Vbus_valid signal is high when the voltage on the VBUS pad is higher than or equal to 4.4V.

In device mode, the UOTGHS_SR.VBUS bit follows the Session_valid comparator output:

- It is set when the voltage on the VBUS pad is higher than or equal to 1.4V.
- It is cleared when the voltage on the VBUS pad is lower than 1.4V.

In host mode, the UOTGHS_SR.VBUS bit follows an hysteresis based on Session_valid and Va_Vbus_valid:

- It is set when the voltage on the VBUS pad is higher than or equal to 4.4V.
- It is cleared when the voltage on the VBUS pad is lower than 1.4V.

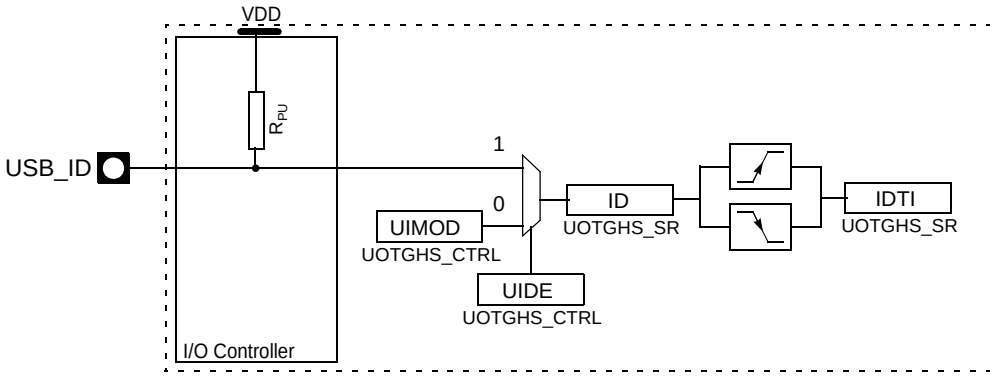
The VBus Transition interrupt (UOTGHS_SR.VBUSTI) bit is set on each transition of the UOTGHS_SR.VBUS bit.

The UOTGHS_SR.VBUS bit is effective whether the UOTGHS is enabled or not.

39.5.1.10 ID Detection

Figure 39-11 on page 1065 shows how the ID transitions are detected.

Figure 39-11. ID Detection Input Block Diagram



The USB mode (device or host) can be either detected from the UOTGID pin or software selected by writing to the UOTGHS_CTRL.UIMOD bit, according to the UOTGHS_CTRL.UIDE bit. This allows the UOTGID pin to be used as a general purpose I/O pin even when the USB interface is enabled.

By default, the UOTGID pin is selected (UOTGHS_CTRL.UIDE is written to one) and the UOTGHS is in device mode (UOTGHS_SR.ID is one), which corresponds to the case where no Mini-A plug is connected, i.e. no plug or a Mini-B plug is connected and the UOTGID pin is kept high by the internal pull-up resistor from the I/O Controller (which must be enabled if UOTGID is used).

The ID Transition Interrupt (UOTGHS_SR.IDTI) bit is set on each transition of the UOTGHS_SR.ID bit, i.e. when a Mini-A plug (host mode) is connected or disconnected. This does not occur when a Mini-B plug (device mode) is connected or disconnected.

The UOTGHS_SR.ID bit is effective whether the UOTGHS is enabled or not.

39.5.2 USB Device Operation

39.5.2.1 Introduction

In device mode, the UOTGHS supports hi-, full- and low-speed data transfers.

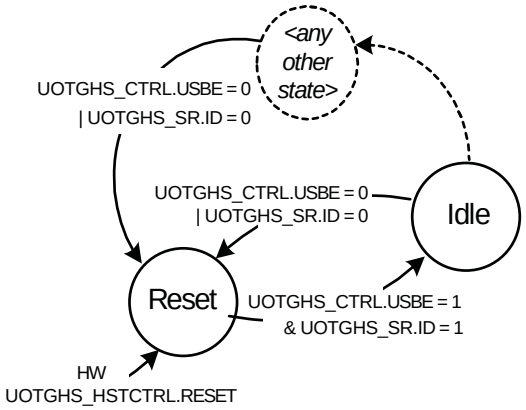
In addition to the default control endpoint, 10 endpoints are provided, which can be configured with an isochronous, bulk or interrupt type, as described in Table 39-1 on page 1053.

As the device mode starts in Idle state, the pad consumption is reduced to the minimum.

39.5.2.2 Power-On and Reset

Figure 39-12 on page 1066 describes the UOTGHS device mode main states.

Figure 39-12. Device Mode States



After a hardware reset, the UOTGHS device mode is in Reset state. In this state:

- the macro clock is stopped to minimize the power consumption (UOTGHS_CTRL.FRZCLK is written to one),
- the internal registers of the device mode are reset,
- the endpoint banks are de-allocated,
- neither D+ nor D- is pulled up (UOTGHS_DEVCTRL.DETACH is written to one).

D+ or D- will be pulled up according to the selected speed as soon as the UOTGHS_DEVCTRL.DETACH bit is written to zero and VBus is present. See "Device mode" for further details.

When the UOTGHS is enabled (UOTGHS_CTRL.USBE is written to one) in device mode (UOTGHS_SR.ID is one), its device mode state goes to the Idle state with minimal power consumption. This does not require the USB clock to be activated.

The UOTGHS device mode can be disabled and reset at any time by disabling the UOTGHS (by writing a zero to UOTGHS_CTRL.USBE) or when the host mode is engaged (UOTGHS_SR.ID is zero).

39.5.2.3 USB Reset

The USB bus reset is managed by hardware. It is initiated by a connected host.

When a USB reset is detected on the USB line, the following operations are performed by the controller:

- All endpoints are disabled, except the default control endpoint.
- The default control endpoint is reset (see Section 39.5.2.4 for more details).
- The data toggle sequence of the default control endpoint is cleared.
- At the end of the reset process, the End of Reset (UOTGHS_DEVISR.EORST) bit is set.
- During a reset, the UOTGHS automatically switches to the Hi-Speed mode if the host is Hi-Speed capable (the reset is called Hi-Speed reset). The user should observe the UOTGHS_SR.SPEED field to know the speed running at the end of the reset (UOTGHS_DEVISR.EORST is one).

39.5.2.4 Endpoint Reset

An endpoint can be reset at any time by writing a one to the Endpoint x Reset (UOTGHS_DEVEPT.EPRSTx) bit. This is recommended before using an endpoint upon hardware reset or when a USB bus reset has been received. This resets:

- the internal state machine of this endpoint,
- the receive and transmit bank FIFO counters,
- all registers of this endpoint (UOTGHS_DEVEPTCFGx, UOTGHS_DEVEPTISRx, the Endpoint x Control (UOTGHS_DEVEPTIMRx) register), except its configuration (UOTGHS_DEVEPTCFGx.ALLOC, UOTGHS_DEVEPTCFGx.EPBK, UOTGHS_DEVEPTCFGx.EPSIZE, UOTGHS_DEVEPTCFGx.EPDIR, UOTGHS_DEVEPTCFGx.EPTYPE) and the Data Toggle Sequence (UOTGHS_DEVEPTISRx.DTSEQ) field.

Note: The interrupt sources located in the UOTGHS_DEVEPTISRx register are not cleared when a USB bus reset has been received.

The endpoint configuration remains active and the endpoint is still enabled.

The endpoint reset may be associated with a clear of the data toggle sequence as an answer to the CLEAR_FEATURE USB request. This can be achieved by writing a one to the Reset Data Toggle Set bit in the Endpoint x Control Set register (UOTGHS_DEVEPTIERx.RSTDTS) (this will set the Reset Data Toggle (UOTGHS_DEVEPTIMRx.RSTDT) bit).

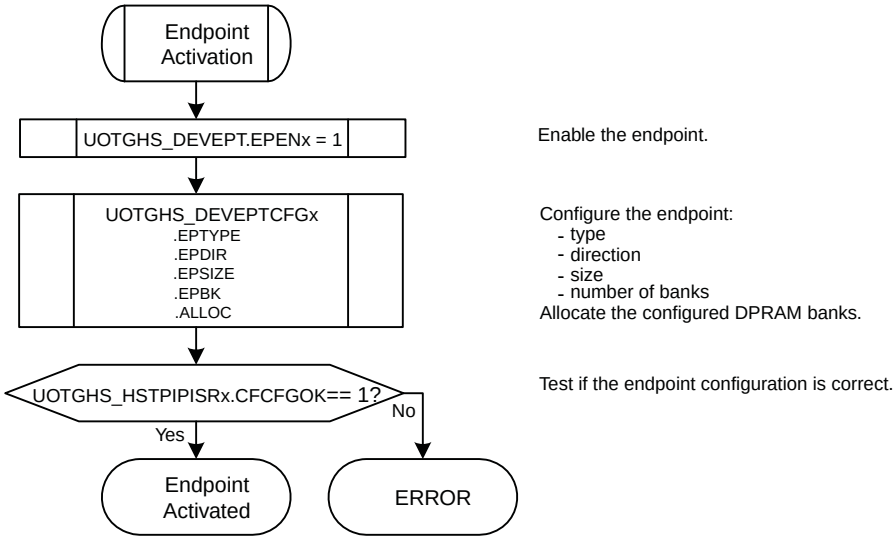
In the end, the user has to write a zero to the UOTGHS_DEVEPT.EPRSTx bit to complete the reset operation and to start using the FIFO.

39.5.2.5 Endpoint Activation

The endpoint is maintained inactive and reset (see Section 39.5.2.4 for more details) as long as it is disabled (UOTGHS_DEVEPT.EPENx is written to zero). UOTGHS_DEVEPTISRx.DTSEQ is also reset.

The algorithm represented on Figure 39-13 on page 1067 must be followed in order to activate an endpoint.

Figure 39-13. Endpoint Activation Algorithm



As long as the endpoint is not correctly configured (UOTGHS_HSTPIPISRx.CFGOK is zero), the controller does not acknowledge the packets sent by the host to this endpoint.

The UOTGHS_HSTPIPISRx.CFGOK bit is set provided that the configured size and number of banks are correct as compared to the endpoint maximal allowed values (see Table 39-1 on page 1053) and to the maximal FIFO size (i.e. the DPRAM size).

See Section 39.5.1.6 for more details about DPRAM management.

39.5.2.6 Address Setup

The USB device address is set up according to the USB protocol.

- After all kinds of resets, the USB device address is 0.
- The host starts a SETUP transaction with a SET_ADDRESS (addr) request.
- The user writes this address to the USB Address (UOTGHS_DEVCTRL.UADD) field, and writes a zero to the Address Enable (UOTGHS_DEVCTRL.ADDEN) bit, so the actual address is still 0.
- The user sends a zero-length IN packet from the control endpoint.
- The user enables the recorded USB device address by writing a one to UOTGHS_DEVCTRL.ADDEN.

Once the USB device address is configured, the controller filters the packets to only accept those targeting the address stored in UOTGHS_DEVCTRL.UADD.

UOTGHS_DEVCTRL.UADD and UOTGHS_DEVCTRL.ADDEN shall not be written all at once.

UOTGHS_DEVCTRL.UADD and UOTGHS_DEVCTRL.ADDEN are cleared:

- on a hardware reset,
- when the UOTGHS is disabled (UOTGHS_CTRL.USBE written to zero),
- when a USB reset is detected.

When UOTGHS_DEVCTRL.UADD or UOTGHS_DEVCTRL.ADDEN is cleared, the default device address 0 is used.

39.5.2.7 Suspend and Wake-up

When an idle USB bus state has been detected for 3 ms, the controller sets the Suspend (UOTGHS_DEVISR.SUSP) interrupt bit. The user may then write a one to the UOTGHS_CTRL.FRZCLK bit to reduce the power consumption.

To recover from the Suspend mode, the user shall wait for the Wake-Up (UOTGHS_DEVISR.WAKEUP) interrupt bit, which is set when a non-idle event is detected, then write a zero to UOTGHS_CTRL.FRZCLK.

As the UOTGHS_DEVISR.WAKEUP interrupt bit is set when a non-idle event is detected, it can occur whether the controller is in the Suspend mode or not. The UOTGHS_DEVISR.SUSP and UOTGHS_DEVISR.WAKEUP interrupts are thus independent, except that one bit is cleared when the other is set.

39.5.2.8 Detach

The reset value of the UOTGHS_DEVCTRL.DETACH bit is one.

It is possible to initiate a device re-enumeration by simply writing a one then a zero to UOTGHS_DEVCTRL.DETACH.

UOTGHS_DEVCTRL.DETACH acts on the pull-up connections of the D+ and D- pads. See “Device mode” for further details.

39.5.2.9 Remote Wake-up

The Remote Wake-Up request (also known as Upstream Resume) is the only one the device may send on its own initiative, but the device should have beforehand been allowed to by a DEVICE_REMOTE_WAKEUP request from the host.

- First, the UOTGHS must have detected a “Suspend” state on the bus, i.e. the Remote Wake-Up request can only be sent after a UOTGHS_DEVISR.SUSP interrupt has been set.
- The user may then write a one to the Remote Wake-Up (UOTGHS_DEVCTRL.RMWKUP) bit to send an upstream resume to the host for a remote wake-up. This will automatically be done by the controller after 5 ms of inactivity on the USB bus.

- When the controller sends the upstream resume, the Upstream Resume (UOTGHS_DEVISR.UPRSM) interrupt is set and UOTGHS_DEVISR.SUSP is cleared.
- UOTGHS_DEVCTRL.RMWKUP is cleared at the end of the upstream resume.
- If the controller detects a valid “End of Resume” signal from the host, the End of Resume (UOTGHS_DEVISR.EORSM) interrupt is set.

39.5.2.10 STALL Request

For each endpoint, the STALL management is performed using:

- The STALL Request (UOTGHS_DEVEPTIMRx.STALLRQ) bit to initiate a STALL request.
- The STALLed Interrupt (UOTGHS_DEVEPTISRx.STALLEDI) bit, which is set when a STALL handshake has been sent.

To answer the next request with a STALL handshake, UOTGHS_DEVEPTIMRx.STALLRQ has to be set by writing a one to the STALL Request Set (UOTGHS_DEVEPTIERx.STALLRQS) bit. All following requests will be discarded (UOTGHS_DEVEPTISRx.RXOUTI, etc. will not be set) and handshaked with a STALL until the UOTGHS_DEVEPTIMRx.STALLRQ bit is cleared, which is done when a new SETUP packet is received (for control endpoints) or when the STALL Request Clear (UOTGHS_DEVEPTIMRx.STALLRQC) bit is written to one.

Each time a STALL handshake is sent, the UOTGHS_DEVEPTISRx.STALLEDI bit is set by the UOTGHS and the PEP_x interrupt is set.

Special considerations for control endpoints

If a SETUP packet is received into a control endpoint for which a STALL is requested, the Received SETUP Interrupt (UOTGHS_DEVEPTISRx.RXSTPI) bit is set and UOTGHS_DEVEPTIMRx.STALLRQ and UOTGHS_DEVEPTISRx.STALLEDI are cleared. The SETUP has to be ACKed.

This simplifies the enumeration process management. If a command is not supported or contains an error, the user requests a STALL and can return to the main task, waiting for the next SETUP request.

STALL handshake and retry mechanism

The retry mechanism has priority over the STALL handshake. A STALL handshake is sent if the UOTGHS_DEVEPTIMRx.STALLRQ bit is set and if no retry is required.

39.5.2.11 Management of Control Endpoints

Overview

A SETUP request is always ACKed. When a new SETUP packet is received, the UOTGHS_DEVEPTISRx.RXSTPI is set; the Received OUT Data Interrupt (UOTGHS_DEVEPTISRx.RXOUTI) bit is not.

The FIFO Control (UOTGHS_DEVEPTIMRx.FIFOCON) bit and the Read-write Allowed (UOTGHS_DEVEPTISRx.RWALL) bit are irrelevant for control endpoints. The user shall never use them on these endpoints. When read, their values are always zero.

Control endpoints are managed using:

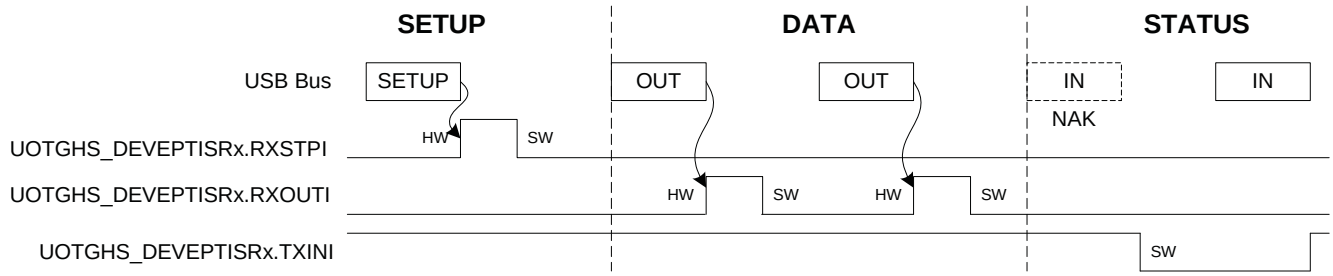
- the UOTGHS_DEVEPTISRx.RXSTPI bit, which is set when a new SETUP packet is received and which shall be cleared by firmware to acknowledge the packet and to free the bank;
- the UOTGHS_DEVEPTISRx.RXOUTI bit, which is set when a new OUT packet is received and which shall be cleared by firmware to acknowledge the packet and to free the bank;
- the Transmitted IN Data Interrupt (UOTGHS_DEVEPTISRx.TXINI) bit, which is set when the current bank is ready to accept a new IN packet and which shall be cleared by firmware to send the packet.

Control write

Figure 39-14 on page 1070 shows a control write transaction. During the status stage, the controller will not necessarily send a NAK on the first IN token:

- if the user knows the exact number of descriptor bytes that must be read, it can then anticipate the status stage and send a zero-length packet after the next IN token, or
- it can read the bytes and wait for the NAKed IN Interrupt (UOTGHS_DEVEPTISRx.NAKINI), which tells that all the bytes have been sent by the host and that the transaction is now in the status stage.

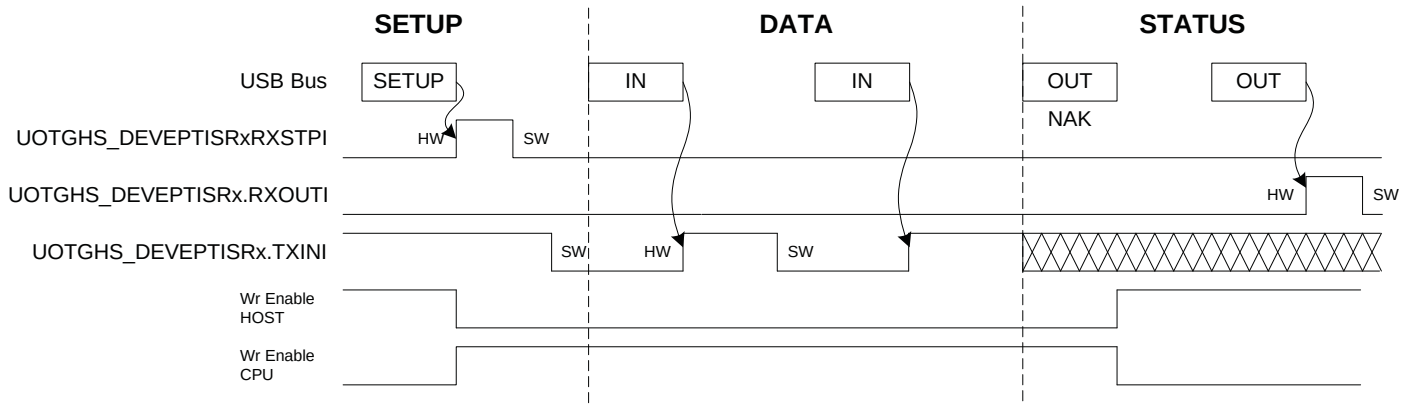
Figure 39-14. Control Write



Control read

Figure 39-15 on page 1070 shows a control read transaction. The UOTGHS has to manage the simultaneous write requests from the CPU and the USB host.

Figure 39-15. Control Read



A NAK handshake is always generated on the first status stage command.

When the controller detects the status stage, all data written by the CPU is lost and clearing UOTGHS_DEVEPTISRx.TXINI has no effect.

The user checks if the transmission or the reception is complete.

The OUT retry is always ACKed. This reception sets UOTGHS_DEVEPTISRx.RXOUTI and UOTGHS_DEVEPTISRx.TXINI. Handle this with the following software algorithm:

```
set TXINI
wait for RXOUTI OR TXINI
if RXOUTI, then clear bit and return
if TXINI, then continue
```

Once the OUT status stage has been received, the UOTGHS waits for a SETUP request. The SETUP request has priority over any other request and has to be ACKed. This means that any other bit should be cleared and the FIFO reset when a SETUP is received.

The user has to consider that the byte counter is reset when a zero-length OUT packet is received.

39.5.2.12 Management of IN Endpoints

Overview

IN packets are sent by the USB device controller upon IN requests from the host. All data which acknowledges or not the bank can be written when it is full.

The endpoint must be configured first.

The UOTGHS_DEVEPTISRx.TXINI bit is set at the same time as UOTGHS_DEVEPTIMRx.FIFOCON when the current bank is free. This triggers a PEP_x interrupt if the Transmitted IN Data Interrupt Enable (UOTGHS_DEVEPTIMRx.TXINE) bit is one.

UOTGHS_DEVEPTISRx.TXINI shall be cleared by software (by writing a one to the Transmitted IN Data Interrupt Clear bit (UOTGHS_DEVEPTIDRx.TXINIC)) to acknowledge the interrupt, what has no effect on the endpoint FIFO.

The user then writes into the FIFO and writes a one to the FIFO Control Clear (UOTGHS_DEVEPTIDRx.FIFOCONC) bit to clear the UOTGHS_DEVEPTIMRx.FIFOCON bit. This allows the UOTGHS to send the data. If the IN endpoint is composed of multiple banks, this also switches to the next bank. The UOTGHS_DEVEPTISRx.TXINI and UOTGHS_DEVEPTIMRx.FIFOCON bits are updated in accordance with the status of the next bank.

UOTGHS_DEVEPTISRx.TXINI shall always be cleared before clearing UOTGHS_DEVEPTIMRx.FIFOCON.

The UOTGHS_DEVEPTISRx.RWALL bit is set when the current bank is not full, i.e. the software can write further data into the FIFO.

Figure 39-16. Example of an IN Endpoint with 1 Data Bank

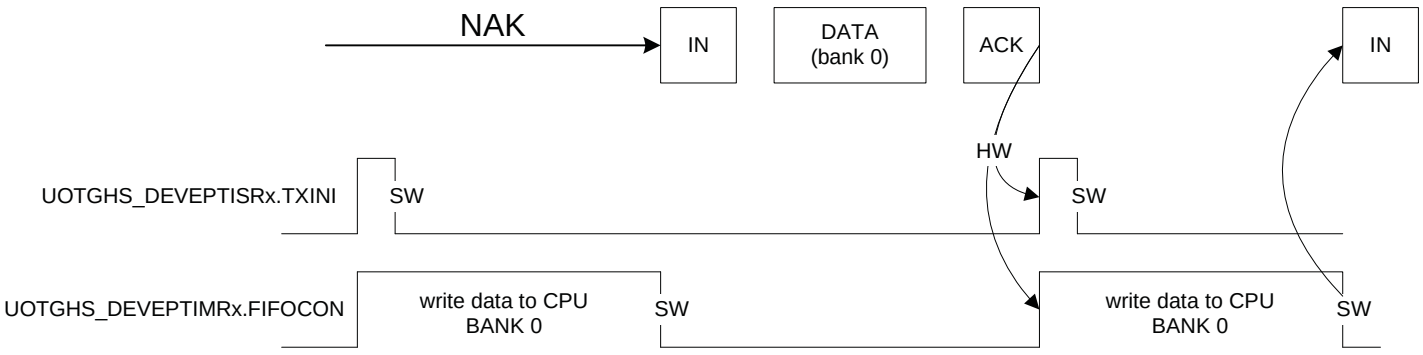
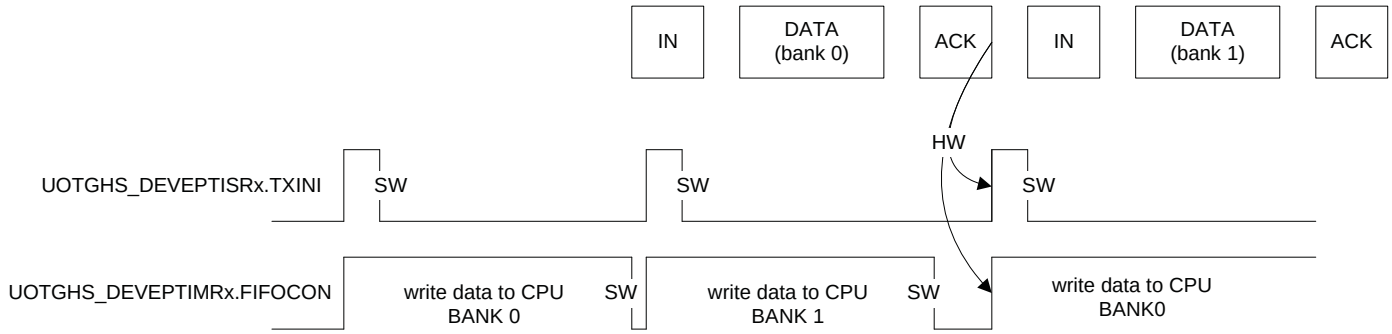


Figure 39-17. Example of an IN Endpoint with 2 Data Banks



Detailed description

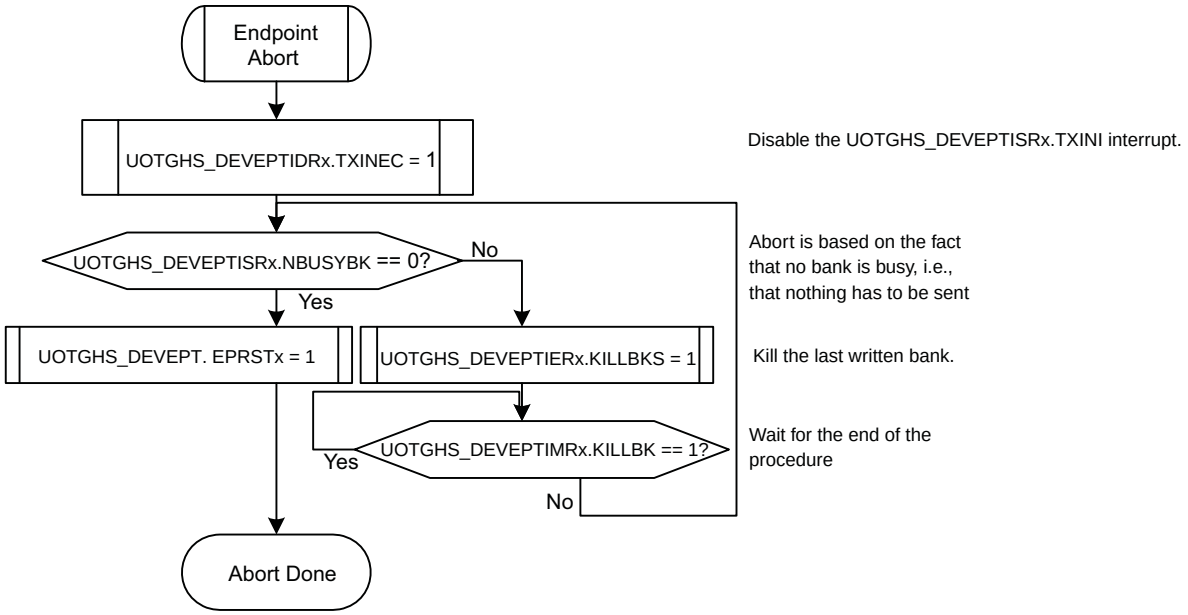
The data is written, following the next flow:

- When the bank is empty, UOTGHS_DEVEPTISRx.TXINI and UOTGHS_DEVEPTIMRx.FIFOCON are set, which triggers a PEP_x interrupt if UOTGHS_DEVEPTIMRx.TXINE is one.
- The user acknowledges the interrupt by clearing UOTGHS_DEVEPTISRx.TXINI.
- The user writes the data into the current bank by using the USB Pipe/Endpoint nFIFO Data (USBFIFOnDATA) register, until all the data frame is written or the bank is full (in which case UOTGHS_DEVEPTISRx.RWALL is cleared and the Byte Count (UOTGHS_DEVEPTISRx.BYCT) field reaches the endpoint size).
- The user allows the controller to send the bank and switches to the next bank (if any) by clearing UOTGHS_DEVEPTIMRx.FIFOCON.

If the endpoint uses several banks, the current one can be written while the previous one is being read by the host. Then, when the user clears UOTGHS_DEVEPTIMRx.FIFOCON, the following bank may already be free and UOTGHS_DEVEPTISRx.TXINI is set immediately.

An “Abort” stage can be produced when a zero-length OUT packet is received during an IN stage of a control or isochronous IN transaction. The Kill IN Bank (UOTGHS_DEVEPTIMRx.KILLBK) bit is used to kill the last written bank. The best way to manage this abort is to apply the algorithm represented on Figure 39-18 on page 1073.

Figure 39-18. Abort Algorithm



39.5.2.13 Management of OUT Endpoints

Overview

OUT packets are sent by the host. All data which acknowledges or not the bank can be read when it is empty.

The endpoint must be configured first.

The UOTGHS_DEVEPTISRx.RXOUTI bit is set at the same time as UOTGHS_DEVEPTIMRx.FIFOCON when the current bank is full. This triggers a PEP_x interrupt if the Received OUT Data Interrupt Enable (UOTGHS_DEVEPTIMRx.RXOUTE) bit is one.

UOTGHS_DEVEPTISRx.RXOUTI shall be cleared by software (by writing a one to the Received OUT Data Interrupt Clear (UOTGHS_DEVEPTICRx.RXOUTIC) bit to acknowledge the interrupt, which has no effect on the endpoint FIFO.

The user then reads from the FIFO and clears the UOTGHS_DEVEPTIMRx.FIFOCON bit to free the bank. If the OUT endpoint is composed of multiple banks, this also switches to the next bank. The UOTGHS_DEVEPTISRx.RXOUTI and UOTGHS_DEVEPTIMRx.FIFOCON bits are updated in accordance with the status of the next bank.

UOTGHS_DEVEPTISRx.RXOUTI shall always be cleared before clearing UOTGHS_DEVEPTIMRx.FIFOCON.

The UOTGHS_DEVEPTISRx.RWALL bit is set when the current bank is not empty, i.e. the software can read further data from the FIFO.

Figure 39-19. Example of an OUT Endpoint with one Data Bank

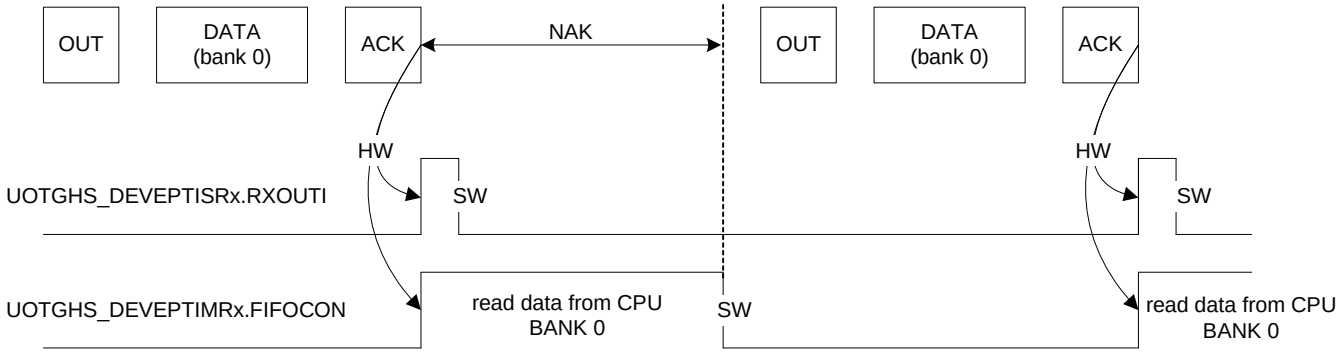
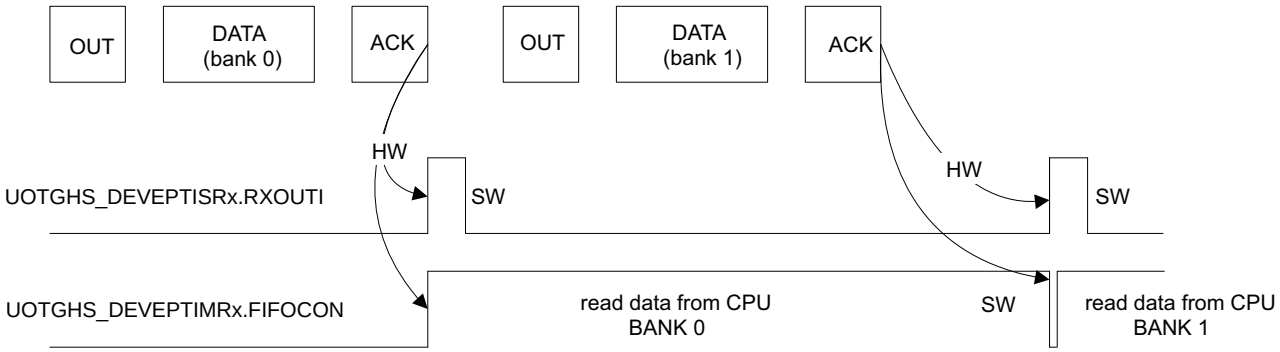


Figure 39-20. Example of an OUT Endpoint with two Data Banks



Detailed description

The data is read, following the next flow:

- When the bank is full, UOTGHS_DEVEPTISRx.RXOUTI and UOTGHS_DEVEPTIMRx.FIFOCON are set, which triggers a PEP_x interrupt if UOTGHS_DEVEPTIMRx.RXOUTE is one.
- The user acknowledges the interrupt by writing a one to UOTGHS_DEVEPTICRx.RXOUTIC in order to clear UOTGHS_DEVEPTISRx.RXOUTI.
- The user can read the byte count of the current bank from UOTGHS_DEVEPTISRx.BYCT to know how many bytes to read, rather than polling UOTGHS_DEVEPTISRx.RWALL.
- The user reads the data from the current bank by using the USBFIFOnDATA register, until all expected data frame is read or the bank is empty (in which case UOTGHS_DEVEPTISRx.RWALL is cleared and UOTGHS_DEVEPTISRx.BYCT reaches zero).
- The user frees the bank and switches to the next bank (if any) by clearing UOTGHS_DEVEPTIMRx.FIFOCON.

If the endpoint uses several banks, the current one can be read while the following one is being written by the host. Then, when the user clears UOTGHS_DEVEPTIMRx.FIFOCON, the following bank may already be read and UOTGHS_DEVEPTISRx.RXOUTI is set immediately.

In Hi-Speed mode, the PING and NYET protocols are handled by the UOTGHS.

- For a single bank, a NYET handshake is always sent to the host (on Bulk-out transaction) to indicate that the current packet is acknowledged but there is no room for the next one.
- For a double bank, the UOTGHS responds to the OUT/DATA transaction with an ACK handshake when the endpoint accepted the data successfully and has room for another data payload (the second bank is free).

39.5.2.14 Underflow

This error only exists for isochronous IN/OUT endpoints. It sets the Underflow Interrupt (UOTGHS_DEVEPTISRx.UNDERFI) bit, which triggers a PEP_x interrupt if the Underflow Interrupt Enable (UOTGHS_DEVEPTIMRx.UNDERFE) bit is one.

- An underflow can occur during the IN stage if the host attempts to read from an empty bank. A zero-length packet is then automatically sent by the UOTGHS.
- An underflow cannot occur during the OUT stage on a CPU action, since the user may only read if the bank is not empty (UOTGHS_DEVEPTISRx.RXOUTI is one or UOTGHS_DEVEPTISRx.RWALL is one).
- An underflow can also occur during the OUT stage if the host sends a packet while the bank is already full. Typically, the CPU is not fast enough. The packet is lost.
- An underflow cannot occur during the IN stage on a CPU action, since the user may only write if the bank is not full (UOTGHS_DEVEPTISRx.TXINI is one or UOTGHS_DEVEPTISRx.RWALL is one).

39.5.2.15 Overflow

This error exists for all endpoint types. It sets the Overflow interrupt (UOTGHS_DEVEPTISRx.OVERFI) bit, which triggers a PEP_x interrupt if the Overflow Interrupt Enable (UOTGHS_DEVEPTIMRx.OVERFE) bit is one.

- An overflow can occur during the OUT stage if the host attempts to write into a bank which is too small for the packet. The packet is acknowledged and the UOTGHS_DEVEPTISRx.RXOUTI bit is set as if no overflow had occurred. The bank is filled with all the first bytes of the packet that fit in.
- An overflow cannot occur during the IN stage on a CPU action, since the user may only write if the bank is not full (UOTGHS_DEVEPTISRx.TXINI is one or UOTGHS_DEVEPTISRx.RWALL is one).

39.5.2.16 HB IsoIn Error

This error only exists for high-bandwidth isochronous IN endpoints.

At the end of the micro-frame, if at least one packet has been sent to the host and less banks than expected have been validated (by clearing the UOTGHS_DEVEPTIMRx.UOTGHS_DEVEPTIMRx.FIFOCON) for this micro-frame, it sets the UOTGHS_DEVEPTISRx.HBISOINERRORI bit, which triggers a PEP_x interrupt if the High Bandwidth Isochronous IN Error Interrupt Enable (HBISOINERRORE) bit is one.

For instance, if the Number of Transaction per MicroFrame for Isochronous Endpoint (NBTRANS field in UOTGHS_DEVEPTCFGx is three (three transactions per micro-frame), only two banks are filled by the CPU (three expected) for the current micro-frame. Then, the HBISOINERRI interrupt is generated at the end of the micro-frame. Note that an UNDERFI interrupt is also generated (with an automatic zero-length-packet), except in the case of a missing IN token.

39.5.2.17 HB IsoFlush

This error only exists for high-bandwidth isochronous IN endpoints.

At the end of the micro-frame, if at least one packet has been sent to the host and there is a missing IN token during this micro-frame, the bank(s) destined to this micro-frame is/are flushed out to ensure a good data synchronization between the host and the device.

For instance, if NBTRANS is three (three transactions per micro-frame), if only the first IN token (among 3) is well received by the UOTGHS, then the two last banks will be discarded.

39.5.2.18 CRC Error

This error only exists for isochronous OUT endpoints. It sets the CRC Error Interrupt (UOTGHS_DEVEPTISRx.CRCERRI) bit, which triggers a PEP_x interrupt if the CRC Error Interrupt Enable (UOTGHS_DEVEPTIMRx.CRCERRE) bit is one.

A CRC error can occur during the OUT stage if the UOTGHS detects a corrupted received packet. The OUT packet is stored in the bank as if no CRC error had occurred (UOTGHS_DEVEPTISRx.RXOUTI is set).

39.5.2.19 Interrupts

See the structure of the USB device interrupt system on Figure 39-6 on page 1059.

There are two kinds of device interrupts: processing, i.e. their generation is part of the normal processing, and exception, i.e. errors (not related to CPU exceptions).

Global interrupts

The processing device global interrupts are:

- The Suspend (UOTGHS_DEVISR.SUSP) interrupt
- The Start of Frame (UOTGHS_DEVISR.SOF) interrupt with no frame number CRC error - the Frame Number CRC Error (UOTGHS_DEVFNUM.FNCERR) bit is zero.
- The Micro Start of Frame (UOTGHS_DEVISR.MSOF) interrupt with no CRC error.
- The End of Reset (UOTGHS_DEVISR.EORST) interrupt
- The Wake-Up (UOTGHS_DEVISR.WAKEUP) interrupt
- The End of Resume (UOTGHS_DEVISR.EORSM) interrupt
- The Upstream Resume (UOTGHS_DEVISR.UPRSM) interrupt
- The Endpoint x (UOTGHS_DEVISR.PEP_x) interrupt
- The DMA Channel x (UOTGHS_DEVISR.DMA_x) interrupt

The exception device global interrupts are:

- The Start of Frame (UOTGHS_DEVISR.SOF) interrupt with a frame number CRC error (UOTGHS_DEVFNUM.FNCERR. is one)
- The Micro Start of Frame (UOTGHS_DEVFNUM.FNCERR.MSOF) interrupt with a CRC error

Endpoint Interrupts

The processing device endpoint interrupts are:

- The Transmitted IN Data Interrupt (UOTGHS_DEVEPTISRx.TXINI)
- The Received OUT Data Interrupt (UOTGHS_DEVEPTISRx.RXOUTI)
- The Received SETUP Interrupt (UOTGHS_DEVEPTISRx.RXSTPI)
- The Short Packet (UOTGHS_DEVEPTISRx.SHORTPACKET) interrupt
- The Number of Busy Banks (UOTGHS_DEVEPTISRx.NBUSYBK) interrupt
- The Received OUT isochronous Multiple Data Interrupt (DTSEQ=MDATA & UOTGHS_DEVEPTISRx.RXOUTI)
- The Received OUT isochronous DataX Interrupt (DTSEQ=DATAX & UOTGHS_DEVEPTISRx.RXOUTI)

The exception device endpoint interrupts are:

- The Underflow Interrupt (UOTGHS_DEVEPTISRx.UNDERFI)
- The NAKed OUT Interrupt (UOTGHS_DEVEPTISRx.NAKOUTI)
- The High-bandwidth isochronous IN error Interrupt (UOTGHS_DEVEPTISRx.HBISOINERRI)
- The NAKed IN Interrupt (UOTGHS_DEVEPTISRx.NAKINI)
- The High-bandwidth isochronous IN Flush error Interrupt (UOTGHS_DEVEPTISRx.HBISOFLUSHI)
- The Overflow Interrupt (UOTGHS_DEVEPTISRx.OVERFI)
- The STALLed Interrupt (UOTGHS_DEVEPTISRx.STALLEDI)
- The CRC Error Interrupt (UOTGHS_DEVEPTISRx.CRCERRI)
- The Transaction error (UOTGHS_DEVEPTISRx.ERRORTRANS) interrupt

DMA interrupts

The processing device DMA interrupts are:

- The End of USB Transfer Status (UOTGHS_DEVDMASTATUSx.END_TR_ST) interrupt
- The End of Channel Buffer Status (UOTGHS_DEVDMASTATUSx.END_BF_ST) interrupt
- The Descriptor Loaded Status (UOTGHS_DEVDMASTATUSx.DESC_LDST) interrupt

There is no exception device DMA interrupt.

39.5.2.20 Test Modes

When written to one, the UOTGHS_DEVCTRL.TSTPCKT bit switches the USB device controller in a “test packet” mode:

The transceiver repeatedly transmits the packet stored in the current bank. UOTGHS_DEVCTRL.TSTPCKT must be written to zero to exit the “test-packet” mode. The endpoint shall be reset by software after a “test-packet” mode.

This enables the testing of rise and falling times, eye patterns, jitter, and any other dynamic waveform specifications.

The flow control used to send the packets is as follows:

- UOTGHS_DEVCTRL.TSTPCKT=1;
- Store data in an endpoint bank
- Write a zero to UOTGHS_DEVEPTIDRx.FIFOCON bit

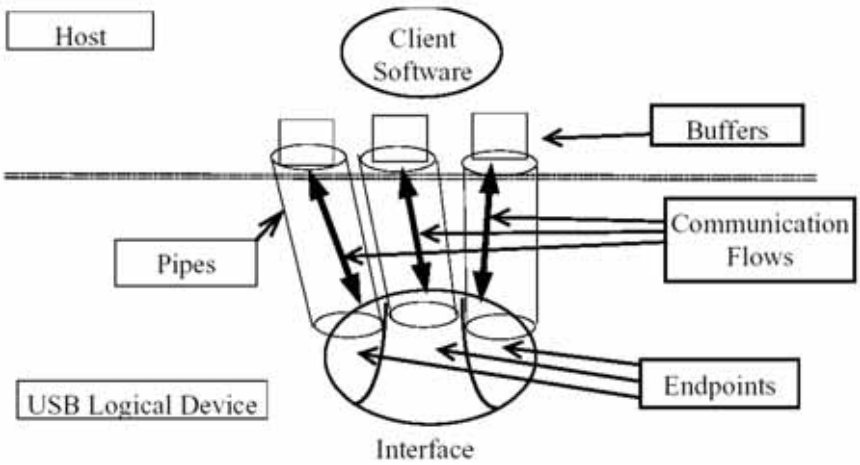
To stop the test-packet mode, just write a zero to the UOTGHS_DEVCTRL.TSTPCKT bit.

39.5.3 USB Host Operation

39.5.3.1 Description of Pipes

For the UOTGHS in host mode, the term “pipe” is used instead of “endpoint” (used in device mode). A host pipe corresponds to a device endpoint, as described in Figure 39-21 on page 1077 from the USB specification.

Figure 39-21. USB Communication Flow

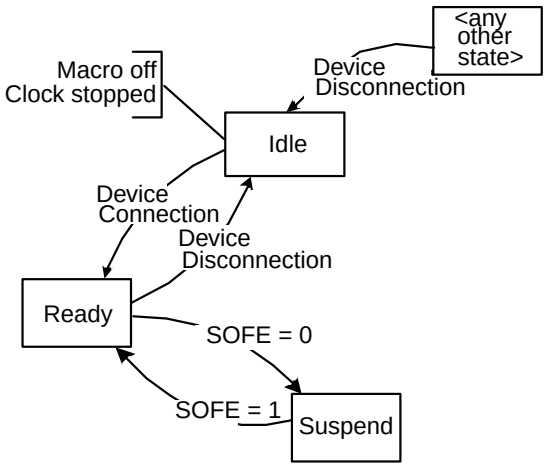


In host mode, the UOTGHS associates a pipe to a device endpoint, considering the device configuration descriptors.

39.5.3.2 Power-On and Reset

Figure 39-22 on page 1078 describes the UOTGHS host mode main states.

Figure 39-22. Host Mode States



After a hardware reset, the UOTGHS host mode is in the Reset state.

When the UOTGHS is enabled (UOTGHS_CTRL.USBE is one) in host mode (UOTGHS_SR.ID is zero), it goes to the Idle state. In this state, the controller waits for a device connection with a minimal power consumption. The USB pad should be in the Idle state. Once a device is connected, the macro enters the Ready state, which does not require the USB clock to be activated.

The controller enters the Suspend state when the USB bus is in a “Suspend” state, i.e., when the host mode does not generate the “Start of Frame (SOF)”. In this state, the USB consumption is minimal. The host mode exits the Suspend state when starting to generate the SOF over the USB line.

39.5.3.3 Device Detection

A device is detected by the UOTGHS host mode when D+ or D- is no longer tied low, i.e., when the device D+ or D- pull-up resistor is connected. To enable this detection, the host controller has to provide the VBus power supply to the device by writing a one to the UOTGHS_SFR.VBUSRQS bit.

The device disconnection is detected by the host controller when both D+ and D- are pulled down.

39.5.3.4 USB Reset

The UOTGHS sends a USB bus reset when the user write a one to the Send USB Reset bit in the Host General Control register (UOTGHS_HSTCTRL.RESET). The USB Reset Sent Interrupt bit in the Host Global Interrupt Status register (UOTGHS_HSTISR.RSTI) is set when the USB reset has been sent. In this case, all pipes are disabled and de-allocated.

If the bus was previously in a “Suspend” state (the Start of Frame Generation Enable (UOTGHS_HSTCTRL.SOFE) bit is zero), the UOTGHS automatically switches to the “Resume” state, the Host Wake-Up Interrupt (UOTGHS_HSTISR.HWUPI) bit is set and the UOTGHS_HSTCTRL.SOFE bit is set in order to generate SOFs or micro SOFs immediately after the USB reset.

At the end of the reset, the user should check the UOTGHS_SR.SPEED field to know the speed running according to the peripheral capability (LS.FS/HS)

39.5.3.5 Pipe Reset

A pipe can be reset at any time by writing a one to the Pipe x Reset (UOTGHS_HSTPIP.PRSTx) bit. This is recommended before using a pipe upon hardware reset or when a USB bus reset has been sent. This resets:

- the internal state machine of this pipe,
- the receive and transmit bank FIFO counters,
- all the registers of this pipe (UOTGHS_HSTPIPCFGx, UOTGHS_HSTPIPISRx, UOTGHS_HSTPIPIMRx), except its configuration (UOTGHS_HSTPIPCFGx.ALLOC, UOTGHS_HSTPIPCFGx.PBK, UOTGHS_HSTPIPCFGx.PSIZE, UOTGHS_HSTPIPCFGx.PTOKEN, UOTGHS_HSTPIPCFGx.PTYPE, UOTGHS_HSTPIPCFGx.PEPNUM, UOTGHS_HSTPIPCFGx.INTFRQ) and its Data Toggle Sequence field (UOTGHS_HSTPIPISRx.DTSEQ).

The pipe configuration remains active and the pipe is still enabled.

The pipe reset may be associated with a clear of the data toggle sequence. This can be achieved by setting the Reset Data Toggle bit in the Pipe x Control register (UOTGHS_HSTPIPIMRx.RSTDT) (by writing a one to the Reset Data Toggle Set bit in the Pipe x Control Set register (UOTGHS_HSTPIPIERx.RSTDTS)).

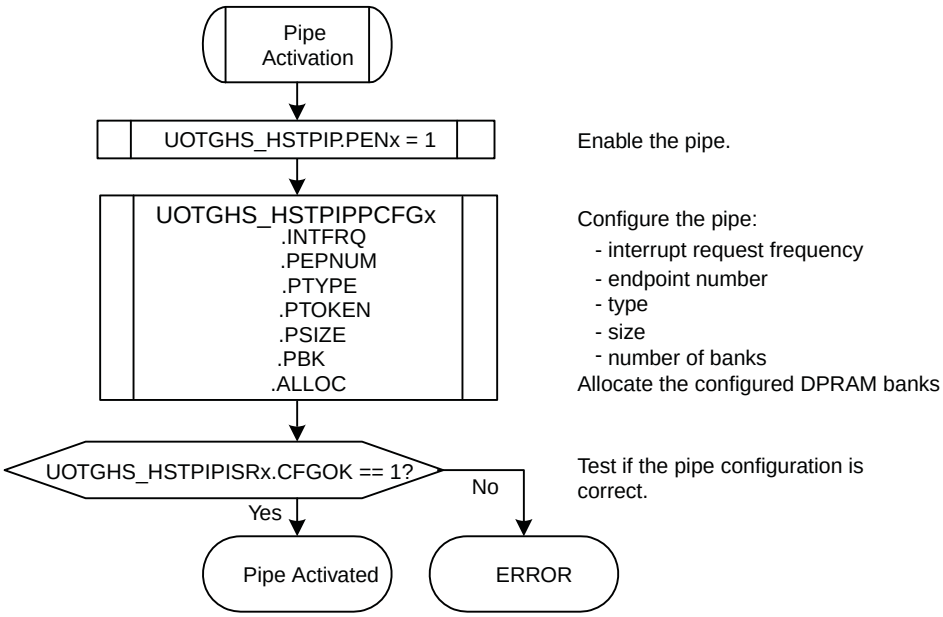
In the end, the user has to write a zero to the UOTGHS_HSTPIP.PRSTx bit to complete the reset operation and to start using the FIFO.

39.5.3.6 Pipe Activation

The pipe is maintained inactive and reset (see Section 39.5.3.5 for more details) as long as it is disabled (UOTGHS_HSTPIP.PENx is zero). The Data Toggle Sequence field (UOTGHS_HSTPIPISRx.DTSEQ) is also reset.

The algorithm represented on Figure 39-23 on page 1079 must be followed in order to activate a pipe.

Figure 39-23. Pipe Activation Algorithm



As long as the pipe is not correctly configured (UOTGHS_HSTPIPISRx.CFGOK is zero), the controller cannot send packets to the device through this pipe.

The UOTGHS_HSTPIPISRx.CFGOK bit is only set if the configured size and number of banks are correct as compared to their maximal allowed values for the pipe (see Table 39-1 on page 1053) and to the maximal FIFO size (i.e. the DPRAM size).

See Section 39.5.1.6 for more details about DPRAM management.

Once the pipe is correctly configured (UOTGHS_HSTPIPISRx.CFGOK is one), only the UOTGHS_HSTPIPCFGx.PTOKEN and UOTGHS_HSTPIPCFGx.INTFRQ fields can be written by software. UOTGHS_HSTPIPCFGx.INTFRQ is meaningless for non-interrupt pipes.

When starting an enumeration, the user gets the device descriptor by sending a GET_DESCRIPTOR USB request. This descriptor contains the maximal packet size of the device default control endpoint (bMaxPacketSize0) and the user re-configures the size of the default control pipe with this size parameter.

39.5.3.7 Address Setup

Once the device has answered the first host requests with default device address 0, the host assigns a new address to the device. The host controller has to send a USB reset to the device and to send a SET_ADDRESS (addr) SETUP request with the new address to be used by the device. Once this SETUP transaction is over, the user writes the new address into the USB Host Address for Pipe x field in the USB Host Device Address register (HSTADDR.HSTADDRPx). All following requests, on all pipes, will be performed using this new address.

When the host controller sends a USB reset, the HSTADDRPx field is reset by hardware and the following host requests will be performed using default device address 0.

39.5.3.8 Remote Wake-up

The controller host mode enters the Suspend state when the UOTGHS_HSTCTRL.SOFE bit is written to zero. No more “Start of Frame” is sent on the USB bus and the USB device enters the Suspend state 3 ms later.

The device awakes the host by sending an Upstream Resume (Remote Wake-Up feature). When the host controller detects a non-idle state on the USB bus, it sets the Host Wake-Up interrupt (UOTGHS_HSTISR.HWUPI) bit. If the non-idle bus state corresponds to an Upstream Resume (K state), the Upstream Resume Received Interrupt (UOTGHS_HSTISR.RXRSMI) bit is set. The user has to generate a Downstream Resume within 1 ms and for at least 20 ms by writing a one to the Send USB Resume (UOTGHS_HSTCTRL.RESUME) bit. It is mandatory to write a one to UOTGHS_HSTCTRL.SOFE before writing a one to UOTGHS_HSTCTRL.RESUME to enter the Ready state, else UOTGHS_HSTCTRL.RESUME will have no effect.

39.5.3.9 Management of Control Pipes

A control transaction is composed of three stages:

- SETUP
- Data (IN or OUT)
- Status (OUT or IN)

The user has to change the pipe token according to each stage.

For the control pipe, and only for it, each token is assigned a specific initial data toggle sequence:

- SETUP: Data0
- IN: Data1
- OUT: Data1

39.5.3.10 Management of IN Pipes

IN packets are sent by the USB device controller upon IN requests from the host. All data which acknowledges or not the bank can be read when it is empty.

The pipe must be configured first.

When the host requires data from the device, the user has to select beforehand the IN request mode with the IN Request Mode bit in the Pipe x IN Request register (UOTGHS_HSTPIPINRQx.INMODE):

- When UOTGHS_HSTPIPINRQx.INMODE is written to zero, the UOTGHS will perform (INRQ + 1) IN requests before freezing the pipe.
- When UOTGHS_HSTPIPINRQx.INMODE is written to one, the UOTGHS will perform IN requests endlessly when the pipe is not frozen by the user.

The generation of IN requests starts when the pipe is unfrozen (the Pipe Freeze (UOTGHS_HSTPIPIMRx.PFREEZE) field in UOTGHS_HSTPIPIMRx is zero).

The Received IN Data Interrupt (UOTGHS_HSTPIPISRx.RXINI) bit is set at the same time as the FIFO Control (UOTGHS_HSTPIPIMRx.FIFOCON) bit when the current bank is full. This triggers a PEP_x interrupt if the Received IN Data Interrupt Enable (UOTGHS_HSTPIPIMRx.RXINE) bit is one.

UOTGHS_HSTPIPISRx.RXINI shall be cleared by software (by writing a one to the Received IN Data Interrupt Clear bit in the Host Pipe x Clear register (UOTGHS_HSTPIPIDRx.RXINIC)) to acknowledge the interrupt, which has no effect on the pipe FIFO.

The user then reads from the FIFO and clears the UOTGHS_HSTPIPIMRx.FIFOCON bit (by writing a one to the FIFO Control Clear (UOTGHS_HSTPIPIDRx.FIFOCONC) bit) to free the bank. If the IN pipe is composed of multiple banks, this also switches to the next bank. The UOTGHS_HSTPIPISRx.RXINI and UOTGHS_HSTPIPIMRx.FIFOCON bits are updated in accordance with the status of the next bank.

UOTGHS_HSTPIPISRx.RXINI shall always be cleared before clearing UOTGHS_HSTPIPIMRx.FIFOCON.

The Read-write Allowed (UOTGHS_HSTPIPISRx.RWALL) bit is set when the current bank is not empty, i.e., the software can read further data from the FIFO.

Figure 39-24. Example of an IN Pipe with 1 Data Bank

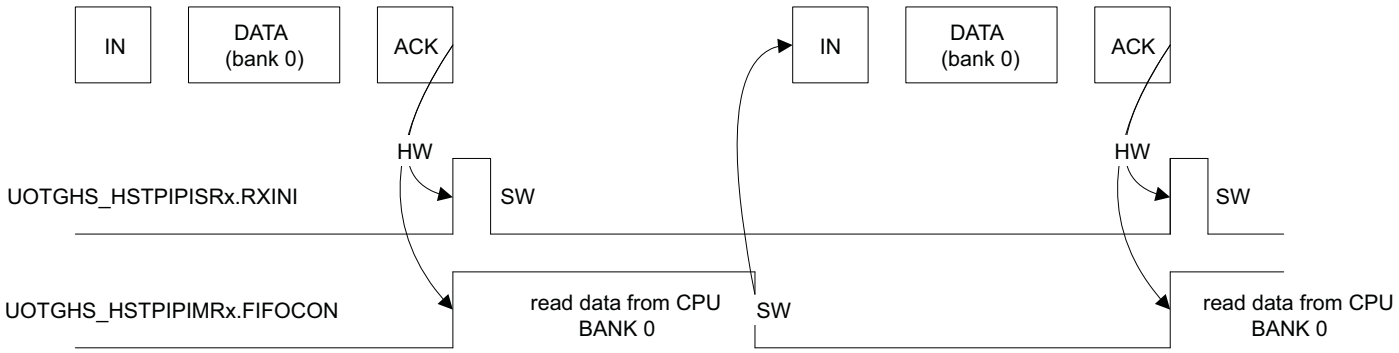
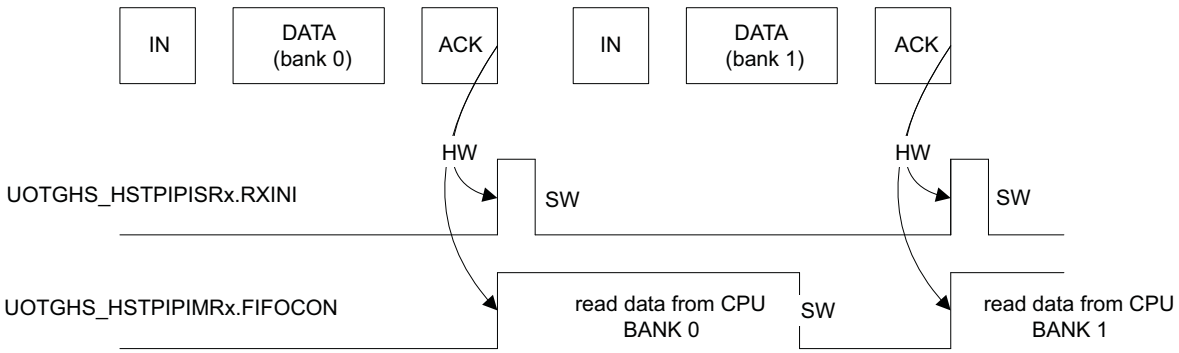


Figure 39-25. Example of an IN Pipe with 2 Data Banks



39.5.3.11 Management of OUT Pipes

OUT packets are sent by the host. All data which acknowledges or not the bank can be written when it is full.

The pipe must be configured and unfrozen first.

The Transmitted OUT Data Interrupt (UOTGHS_HSTPIPISRx.TXOUTI) bit is set at the same time as UOTGHS_HSTPIPIMRx.FIFOCON when the current bank is free. This triggers a PEP_x interrupt if the Transmitted OUT Data Interrupt Enable (UOTGHS_HSTPIPIMRx.TXOUTE) bit is one.

UOTGHS_HSTPIPISRx.TXOUTI shall be cleared by software (by writing a one to the Transmitted OUT Data Interrupt Clear (UOTGHS_HSTPIPIDRx.TXOUTIC) bit) to acknowledge the interrupt, which has no effect on the pipe FIFO.

The user then writes into the FIFO and clears the UOTGHS_HSTPIPIDRx.FIFOCON bit to allow the UOTGHS to send the data. If the OUT pipe is composed of multiple banks, this also switches to the next bank. The UOTGHS_HSTPIPISRx.TXOUTI and UOTGHS_HSTPIPIMRx.FIFOCON bits are updated in accordance with the status of the next bank.

UOTGHS_HSTPIPISRx.TXOUTI shall always be cleared before clearing UOTGHS_HSTPIPIMRx.FIFOCON.

The UOTGHS_HSTPIPISRx.RWALL bit is set when the current bank is not full, i.e., the software can write further data into the FIFO.

Notes:
1. If the user decides to switch to the Suspend state (by writing a zero to the UOTGHS_HSTCTRL.SOFE bit) while a bank is ready to be sent, the UOTGHS automatically exits this state and the bank is sent.
2. In High-Speed operating mode, the host controller automatically manages the PING protocol to maximize the USB bandwidth. The user can tune the PING protocol by handling the Ping Enable (PINGEN) bit and the bInterval Parameter for the Bulk-Out/Ping Transaction (BINTERVALL) field in UOTGHS_HSTPIPCFGx. See the Section 39.6.3.13 for more details.

Figure 39-26. Example of an OUT Pipe with one Data Bank

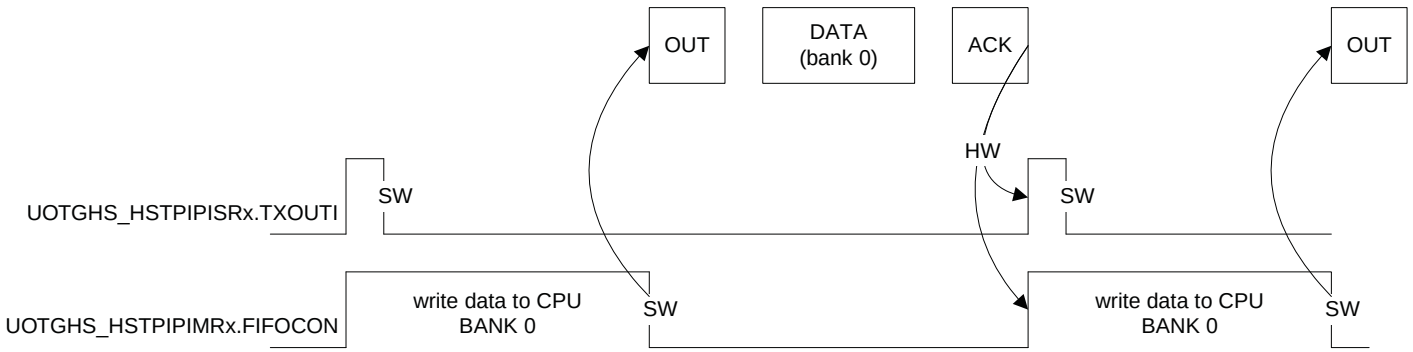


Figure 39-27. Example of an OUT Pipe with two Data Banks and no Bank Switching Delay

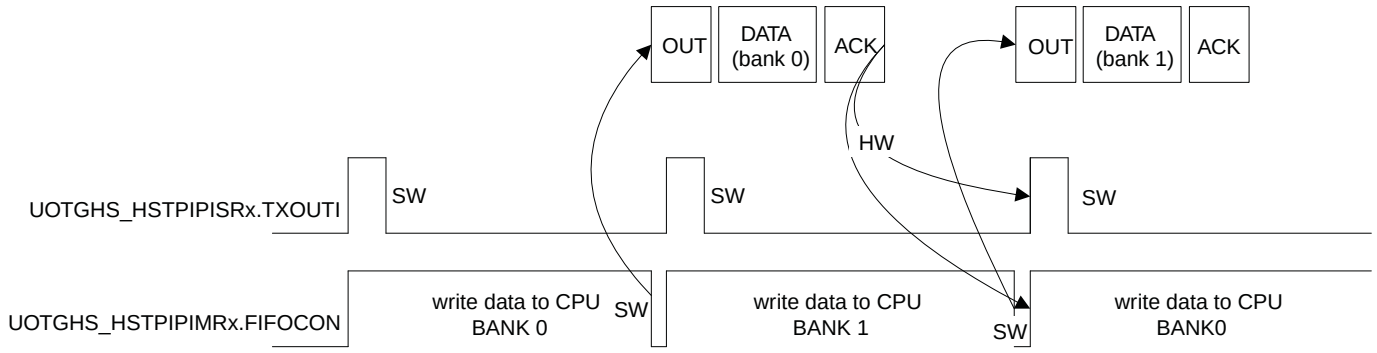
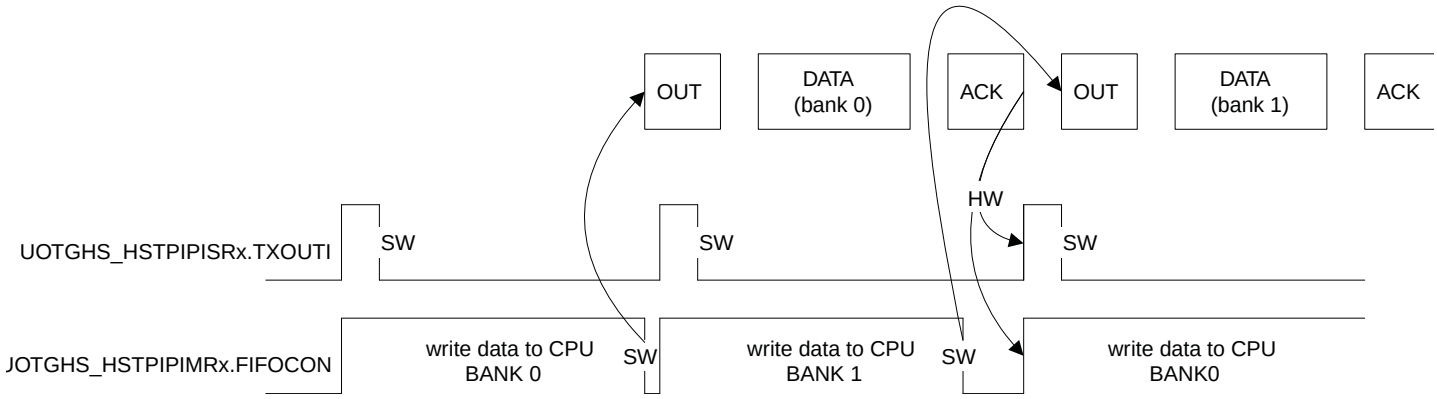


Figure 39-28. Example of an OUT Pipe with two Data Banks and a Bank Switching Delay



39.5.3.12 CRC Error

This error exists only for isochronous IN pipes. It sets the CRC Error Interrupt (UOTGHS_HSTPIPISRx.CRCERRI) bit, which triggers a PEP_x interrupt if then the CRC Error Interrupt Enable (UOTGHS_HSTPIPIMRx.CRCERRE) bit is one.

A CRC error can occur during IN stage if the UOTGHS detects a corrupted received packet. The IN packet is stored in the bank as if no CRC error had occurred (UOTGHS_HSTPIPISRx.RXINI is set).

39.5.3.13 Interrupts

See the structure of the USB host interrupt system on Figure 39-6 on page 1059.

There are two kinds of host interrupts: processing, i.e. their generation is part of the normal processing, and exception, i.e. errors (not related to CPU exceptions).

Global interrupts

The processing host global interrupts are:

- The Device Connection Interrupt (UOTGHS_HSTISR.DCONNI)
- The Device Disconnection Interrupt (UOTGHS_HSTISR.DDISCI)
- The USB Reset Sent Interrupt (UOTGHS_HSTISR.RSTI)
- The Downstream Resume Sent Interrupt (UOTGHS_HSTISR.RSMEDI)
- The Upstream Resume Received Interrupt (UOTGHS_HSTISR.RXRSMI)
- The Host Start of Frame Interrupt (UOTGHS_HSTISR.HSOFI)
- The Host Wake-Up Interrupt (UOTGHS_HSTISR.HWUPI)
- The Pipe x Interrupt (UOTGHS_HSTISR.PEP_x)
- The DMA Channel x Interrupt (UOTGHS_HSTISR.DMAxINT)

There is no exception host global interrupt.

Pipe interrupts

The processing host pipe interrupts are:

- The Received IN Data Interrupt (UOTGHS_HSTPIPISRx.RXINI)
- The Transmitted OUT Data Interrupt (UOTGHS_HSTPIPISRx.TXOUTI)
- The Transmitted SETUP Interrupt (UOTGHS_HSTPIPISRx.TXSTPI)
- The Short Packet Interrupt (UOTGHS_HSTPIPISRx.SHORTPACKETI)
- The Number of Busy Banks (UOTGHS_HSTPIPISRx.NBUSYBK) interrupt

The exception host pipe interrupts are:

- The Underflow Interrupt (UOTGHS_HSTPIPISRx.UNDERFI)
- The Pipe Error Interrupt (UOTGHS_HSTPIPISRx.PERRI)
- The NAKed Interrupt (UOTGHS_HSTPIPISRx.NAKEDI)
- The Overflow Interrupt (UOTGHS_HSTPIPISRx.OVERFI)
- The Received STALLed Interrupt (UOTGHS_HSTPIPISRx.RXSTALLDI)
- The CRC Error Interrupt (UOTGHS_HSTPIPISRx.CRCERRI)

DMA interrupts

The processing host DMA interrupts are:

- The End of USB Transfer Status (UOTGHS_HSTDMASTATUSx.END_TR_ST) interrupt
- The End of Channel Buffer Status (UOTGHS_HSTDMASTATUSx.END_BF_ST) interrupt
- The Descriptor Loaded Status (UOTGHS_HSTDMASTATUSx.DESC_LDST) interrupt

There is no exception host DMA interrupt.

39.5.4 USB DMA Operation

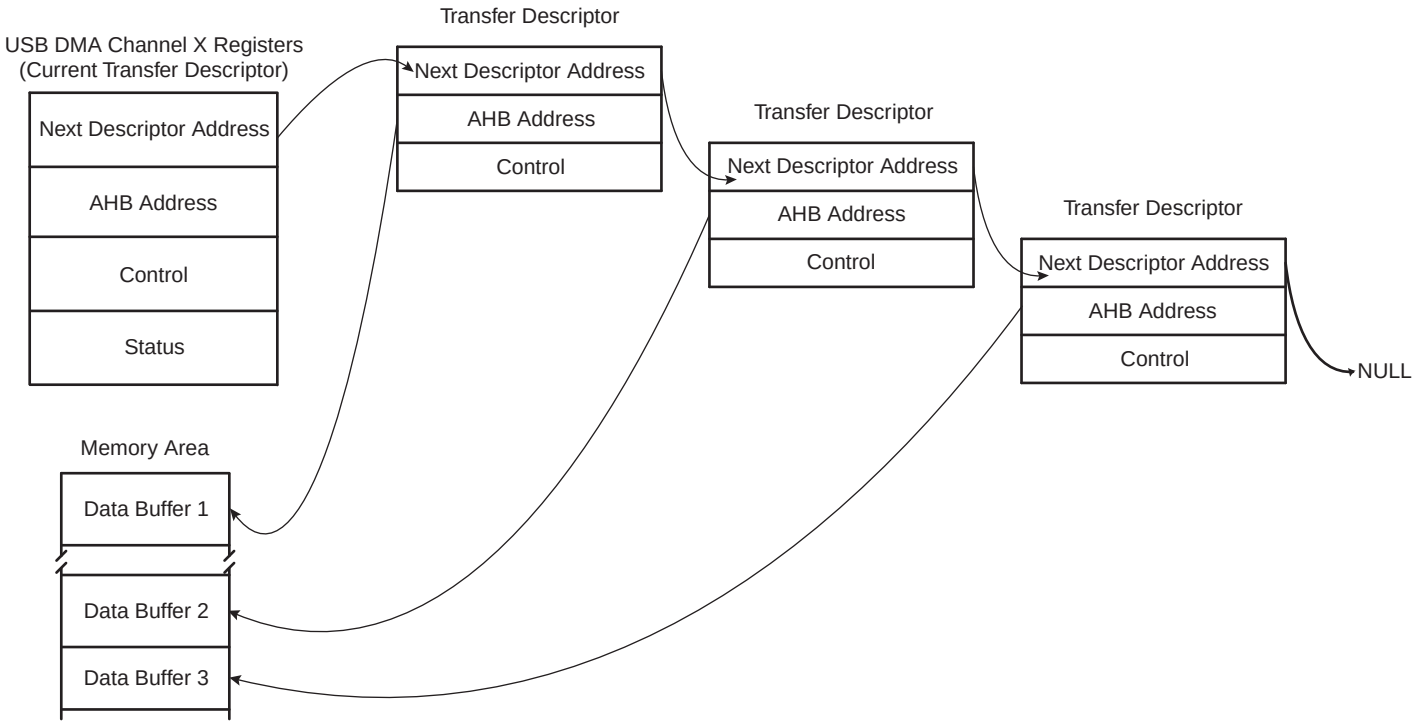
USB packets of any length may be transferred when required by the UOTGHS. These transfers always feature sequential addressing. These two characteristics mean that in case of high UOTGHS throughput, both AHB ports will benefit from “incrementing burst of unspecified length” since the average access latency of AHB slaves can then be reduced.

The DMA uses word “incrementing burst of unspecified length” of up to 256 beats for both data transfers and channel descriptor loading. A burst may last on the AHB busses for the duration of a whole USB packet transfer, unless otherwise broken by the AHB arbitration or the AHB 1 kbyte boundary crossing.

Packet data AHB bursts may be locked on a DMA buffer basis for drastic overall AHB bus bandwidth performance boost with paged memories. This is because these memory row (or bank) changes, which are very clock-cycle consuming, will then likely not occur or occur once instead of dozens of times during a single big USB packet DMA transfer in case other AHB masters address the memory. This means up to 128 words single cycle unbroken AHB bursts for bulk pipes/endpoints and 256 words single cycle unbroken bursts for isochronous pipes/endpoints. This maximal burst length is then controlled by the lowest programmed USB pipe/endpoint size (UOTGHS_HSTPIPCFGx.PSIZE / UOTGHS_DEVEPTCFGx.EPSIZE) and the Buffer Byte Length (UOTGHS_HSTDMACONTROLx.BUFF_LENGTH / UOTGHS_DEVDMACONTROLx.BUFF_LENGTH) field.

The UOTGHS average throughput may be up to nearly 480 Mbps. Its average access latency decreases as burst length increases due to the zero wait-state side effect of unchanged pipe/endpoint. Word access allows reducing the AHB bandwidth required for the USB by four, as compared to native byte access. If at least 0 wait-state word burst capability is also provided by the other DMA AHB bus slaves, each of both DMA AHB busses need less than 60% bandwidth allocation for full USB bandwidth usage at 33 MHz, and less than 30% at 66 MHz.

Figure 39-29. Example of DMA Chained List



39.5.5 USB DMA Channel Transfer Descriptor

The DMA channel transfer descriptor is loaded from the memory. Be careful with the alignment of this buffer. The structure of the DMA channel transfer descriptor is defined by three parameters as described below:

Offset 0:

- The address must be aligned: 0xXXXX0
- Next Descriptor Address Register: UOTGHS_xxxDMANXTDSCx

Offset 4:

- The address must be aligned: 0xXXXX4
- DMA Channelx Address Register: UOTGHS_xxxDMAADDRESSx

Offset 8:

- The address must be aligned: 0xXXXX8
- DMA Channelx Control Register: UOTGHS_xxxDMACONTROLx

To use the DMA channel transfer descriptor, fill the structures with the correct value (as described in the following pages). Then write directly in UOTGHS_xxxDMANXTDSCx the address of the descriptor to be used first.

Then write 1 in UOTGHS_xxxDMACONTROLx.LDNXT_DSC bit (load next channel transfer descriptor). The descriptor is automatically loaded upon pipe x / endpoint x request for packet transfer.

39.6 USB On-The-Go Interface (UOTGHS) User Interface

Table 39-5. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Device General Control Register	UOTGHS_DEVCTRL	Read-write	0x00000100
0x0004	Device Global Interrupt Status Register	UOTGHS_DEVISR	Read-only	0x00000000
0x0008	Device Global Interrupt Clear Register	UOTGHS_DEVICR	Write-only	
0x000C	Device Global Interrupt Set Register	UOTGHS_DEVIFR	Write-only	
0x0010	Device Global Interrupt Mask Register	UOTGHS_DEVIMR	Read-only	0x00000000
0x0014	Device Global Interrupt Disable Register	UOTGHS_DEVIDR	Write-only	
0x0018	Device Global Interrupt Enable Register	UOTGHS_DEVIER	Write-only	
0x001C	Device Endpoint Register	UOTGHS_DEVEPT	Read-write	0x00000000
0x0020	Device Frame Number Register	UOTGHS_DEVFNUM	Read-only	0x00000000
0x0100 + (n * 0x04) + 0x00	Device Endpoint Configuration Register	UOTGHS_DEVEPTCFG	Read-write	0x00002000
0x0100 + (n * 0x04) + 0x30	Device Endpoint Status Register	UOTGHS_DEVEPTISR	Read-only	0x00000100
0x0100 + (n * 0x04) + 0x60	Device Endpoint Clear Register	UOTGHS_DEVEPTICR	Write-only	
0x0100 + (n * 0x04) + 0x90	Device Endpoint Set Register	UOTGHS_DEVEPTIFR	Write-only	
0x0100 + (n * 0x04) + 0x0C0	Device Endpoint Mask Register	UOTGHS_DEVEPTIMR	Read-only	0x00000000
0x0100 + (n * 0x04) + 0x0F0	Device Endpoint Enable Register	UOTGHS_DEVEPTIER	Write-only	
0x0100 + (n * 0x04) + 0x0120	Device Endpoint Disable Register	UOTGHS_DEVEPTIDR	Write-only	
0x0300 + (n * 0x10)+0x00	Device DMA Channel Next Descriptor Address Register	UOTGHS_DEVDMANXTDSC	Read-write	0x00000000
0x0300 + (n * 0x10)+0x04	Device DMA Channel Address Register	UOTGHS_DEVDMAADDRESS	Read-write	0x00000000
0x0300 + (n * 0x10)+0x08	Device DMA Channel Control Register	UOTGHS_DEVDMACONTROL	Read-write	0x00000000
0x0300 + (n * 0x10)+0x0C	Device DMA Channel Status Register	UOTGHS_DEVDMASTATUS	Read-write	0x00000000
0x0400	Host General Control Register	UOTGHS_HSTCTRL	Read-write	0x00000000
0x0404	Host Global Interrupt Status Register	UOTGHS_HSTISR	Read-only	0x00000000
0x0408	Host Global Interrupt Clear Register	UOTGHS_HSTICR	Write-only	
0x040C	Host Global Interrupt Set Register	UOTGHS_HSTIFR	Write-only	
0x0410	Host Global Interrupt Mask Register	UOTGHS_HSTIMR	Read-only	0x00000000
0x0414	Host Global Interrupt Disable Register	UOTGHS_HSTIDR	Write-only	
0x0418	Host Global Interrupt Enable Register	UOTGHS_HSTIER	Write-only	
0x0041C	Host Pipe Register	UOTGHS_HSTPIP	Read-write	0x00000000
0x0420	Host Frame Number Register	UOTGHS_HSTFNUM	Read-write	0x00000000
0x0424	Host Address 1 Register	UOTGHS_HSTADDR1	Read-write	0x00000000
0x0428	Host Address 2 Register	UOTGHS_HSTADDR2	Read-write	0x00000000
0x042C	Host Address 3 Register	UOTGHS_HSTADDR3	Read-write	0x00000000
0x0500 + (n * 0x04) + 0x00	Host Pipe Configuration Register	UOTGHS_HSTPIPCFG	Read-write	0x00000000
0x0500 + (n * 0x04) + 0x30	Host Pipe Status Register	UOTGHS_HSTPIPISR	Read-only	0x00000000

Table 39-5. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0500 + (n * 0x04) + 0x60	Host Pipe Clear Register	UOTGHS_HSTPIPICR	Write-only	
0x0500 + (n * 0x04) + 0x90	Host Pipe Set Register	UOTGHS_HSTPIPIFR	Write-only	
0x0500 + (n * 0x04) + 0xC0	Host Pipe Mask Register	UOTGHS_HSTPIPIMR	Read-only	0x00000000
0x0500 + (n * 0x04) + 0xF0	Host Pipe Enable Register	UOTGHS_HSTPIPIER	Write-only	
0x0500+ (n * 0x04) + 0x120	Host Pipe Disable Register	UOTGHS_HSTPIPIDR	Write-only	
0x0500+ (n * 0x04) + 0x150	Host Pipe IN Request Register	UOTGHS_HSTPIPINRQ	Read-write	0x00000000
0x0500 + (n * 0x04) + 0x180	Host Pipe Error Register	UOTGHS_HSTPIPERR	Read-write	0x00000000
0x0700 + (n * 0x10) + 0x00	Host DMA Channel Next Descriptor Address Register	UOTGHS_HSTDMANXTDSC	Read-write	0x00000000
0x0700 + (n * 0x10) + 0x04	Host DMA Channel Address Register	UOTGHS_HSTDMAADDRESS	Read-write	0x00000000
0x0700 + (n * 0x10) + 0x08	Host DMA Channel Control Register	UOTGHS_HSTDMACONTROL	Read-write	0x00000000
0x0700 + (n * 0x10) + 0x0C	Host DMA Channel Status Register	UOTGHS_HSTDMASTATUS	Read-write	0x00000000
0x0800	General Control Register	UOTGHS_CTRL	Read-write	0x03004000
0x0804	General Status Register	UOTGHS_SR	Read-only	0x00000400
0x0808	General Status Clear Register	UOTGHS_SCR	Write-only	
0x080C	General Status Set Register	UOTGHS_SFR	Write-only	
0x082C	General Finite State Machine Register	UOTGHS_FSM	Read-only	0x00000009

39.6.1 USB General Registers

39.6.1.1 General Control Register

Name: UOTGHS_CTRL

Address: 0x400AC800

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	UIMOD	UIDE
23	22	21	20	19	18	17	16
–	UNLOCK	TIMPAGE		–	–	TIMVALUE	
15	14	13	12	11	10	9	8
USBE	FRZCLK	VBUSPO	OTGPADE	HNPREQ	SRPREQ	SRPSEL	VBUSHWC
7	6	5	4	3	2	1	0
STOE	HNPERRE	ROLEEXE	BCERRE	VBERRE	SRPE	VBUSTE	IDTE

- **UIMOD: UOTGHS Mode**

This bit has no effect when UIDE is one (UOTGID input pin activated).

0 (Host): The module is in USB host mode.

1 (Device): The module is in USB device mode.

This bit can be written even if USBE is zero or FRZCLK is one. Disabling the UOTGHS (by writing a zero to the USBE bit) does not reset this bit.

- **UIDE: UOTGID Pin Enable**

0 (UIMOD): The USB mode (device/host) is selected from the UIMOD bit.

1 (UOTGID): The USB mode (device/host) is selected from the UOTGID input pin.

This bit can be written even if USBE is zero or FRZCLK is one. Disabling the UOTGHS (by writing a zero to the USBE bit) does not reset this bit.

- **UNLOCK: Timer Access Unlock**

0: The TIMPAGE and TIMVALUE fields are locked.

1: The TIMPAGE and TIMVALUE fields are unlocked.

The TIMPAGE and TIMVALUE fields can always be read, whatever the value of UNLOCK.

- **TIMPAGE: Timer Page**

This field contains the page value to access a special timer register.

- **TIMVALUE: Timer Value**

This field selects the timer value that is written to the special time register selected by TIMPAGE. See Section 39.5.1.8 for details.

- **USBE: UOTGHS Enable**

Writing a zero to this bit will reset the UOTGHS, disable the USB transceiver, and disable the UOTGHS clock inputs. Unless explicitly stated, all registers will then become read-only and will be reset.

0: The UOTGHS is disabled.

1: The UOTGHS is enabled.

This bit can be written even if FRZCLK is one.

- **FRZCLK: Freeze USB Clock**

0: The clock inputs are enabled.

1: The clock inputs are disabled (the resume detection is still active).This reduces the power consumption. Unless explicitly stated, all registers then become read-only.

This bit can be written even if USBE is zero. Disabling the UOTGHS (by writing a zero to the USBE bit) does not reset this bit, but this freezes the clock inputs whatever its value.

- **VBUSPO: VBus Polarity Off**

0: The UOTGVBOF output signal is in its default mode (active high).

1: The UOTGVBOF output signal is inverted (active low).

To be generic. May be useful to control an external VBus power module.

This bit can be written even if USBE is zero or FRZCLK is one. Disabling the UOTGHS (by writing a zero to the USBE bit) does not reset this bit.

- **OTGPADE: OTG Pad Enable**

0: The OTG pad is disabled.

1: The OTG pad is enabled.

This bit can be written even if USBE is zero or FRZCLK is one. Disabling the UOTGHS (by writing a zero to the USBE bit) does not reset this bit.

- **HNPREQ: HNP Request**

When the controller is in device mode:

- Writing a one to this bit will initiate an HNP (Host Negotiation Protocol).
- Writing a zero to this bit has no effect.

This bit is cleared when the controller has initiated an HNP.

When the controller is in host mode:

- Writing a one to this bit will accept an HNP.
- Writing a zero to this bit will reject an HNP.

- **SRPREQ: SRP Request**

Writing a one to this bit will initiate an SRP when the controller is in device mode.

Writing a zero to this bit has no effect.

This bit is cleared when the controller has initiated an SRP.

- **SRPSEL: SRP Selection**

0: Data line pulsing is selected as an SRP method.

1: VBus pulsing is selected as an SRP method.

- **VBUSHWC: VBus Hardware Control**

0: The hardware control over the UOTGVBOF output pin is enabled. The UOTGHS resets the UOTGVBOF output pin when a VBUS problem occurs.

1: The hardware control over the UOTGVBOF output pin is disabled.

- **STOE: Suspend Time-Out Interrupt Enable**

0: The Suspend Time-Out Interrupt (UOTGHS_SR.STOI) is disabled.

1: The Suspend Time-Out Interrupt (UOTGHS_SR.STOI) is enabled.

- **HNPERRE: HNP Error Interrupt Enable**

0: The HNP Error Interrupt (UOTGHS_SR.HNPERRI) is disabled.

1: The HNP Error Interrupt (UOTGHS_SR.HNPERRI) is enabled.

- **ROLEEXE: Role Exchange Interrupt Enable**

0: The Role Exchange Interrupt (UOTGHS_SR.ROLEEXI) is disabled.

1: The Role Exchange Interrupt (UOTGHS_SR.ROLEEXI) is enabled.

- **BCERRE: B-Connection Error Interrupt Enable**

0: The B-Connection Error Interrupt (UOTGHS_SR.BCERRI) is disabled.

1: The B-Connection Error Interrupt (UOTGHS_SR.BCERRI) is enabled.

- **VBERRE: VBus Error Interrupt Enable**

0: The VBus Error Interrupt (UOTGHS_SR.VBERRI) is disabled.

1: The VBus Error Interrupt (UOTGHS_SR.VBERRI) is enabled.

- **SRPE: SRP Interrupt Enable**

0: The SRP Interrupt (UOTGHS_SR.SRPI) is disabled.

1: The SRP Interrupt (UOTGHS_SR.SRPI) is enabled.

- **VBUSTE: VBus Transition Interrupt Enable**

0: The VBus Transition Interrupt (UOTGHS_SR.VBUSTI) is disabled.

1: The VBus Transition Interrupt (UOTGHS_SR.VBUSTI) is enabled.

- **IDTE: ID Transition Interrupt Enable**

0: The ID Transition interrupt (UOTGHS_SR.IDTI) is disabled.

1: The ID Transition interrupt (UOTGHS_SR.IDTI) is enabled.

39.6.1.2 General Status Register

Name: UOTGHS_SR

Address: 0x400AC804

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
–	CLKUSABLE	SPEED		VBUS	ID	VBUSRQ	–
7	6	5	4	3	2	1	0
STOI	HNPERRI	ROLEEXI	BCERRI	VBERRI	SRPI	VBUSTI	IDTI

- **CLKUSABLE: UTMI Clock Usable**

This bit is set when the UTMI 30 MHz is usable.

This bit is cleared when the UTMI 30 MHz is not usable.

- **SPEED: Speed Status**

This field is set according to the controller speed mode. This field shall only be used in device mode.

Value	Name	Description
0	FULL_SPEED	Full-Speed mode
2	LOW_SPEED	Low-Speed mode
1	HIGH_SPEED	High-Speed mode
3		Reserved

- **VBUS: VBus Level**

This bit is set when the VBus line level is high, even if UOTGHS_CTRL.USBE is zero.

This bit is cleared when the VBus line level is low, even if UOTGHS_CTRL.USBE is zero.

This bit can be used in device mode to monitor the USB bus connection state of the application.

- **ID: UOTGID Pin State**

This bit is cleared when the UOTGID level is low, even if UOTGHS_CTRL.USBE is zero.

This bit is set when the UOTGID level is high, event if UOTGHS_CTRL.USBE is zero.

- **VBUSRQ: VBus Request**

This bit is set when the UOTGHS_SFR.VBUSRQS bit is written to one.

This bit is cleared when the UOTGHS_SCR.VBUSRQC bit is written to one or when a VBus error occurs and UOTGHS_CTRL.VBUSHWC is zero.

0: The UOTGVBOF output pin is driven low to disable the VBUS power supply generation.

1: The UOTGVBOF output pin is driven high to enable the VBUS power supply generation.

This bit shall only be used in host mode.

- **STOI: Suspend Time-Out Interrupt**

This bit is set when a time-out error (more than 200 ms) has been detected after a suspend. This triggers a USB interrupt if UOTGHS_CTRL.STOE is one.

This bit is cleared when the UOTGHS_SCR.STOIC bit is written to one.

This bit shall only be used in host mode.

- **HNPERRI: HNP Error Interrupt**

This bit is set when an error has been detected during a HNP negotiation. This triggers a USB interrupt if UOTGHS_CTRL.HNPERRE is one.

This bit is cleared when the UOTGHS_SCR.HNPERRIC bit is written to one.

This bit shall only be used in device mode.

- **ROLEEXI: Role Exchange Interrupt**

This bit is set when the UOTGHS has successfully switched its mode because of an HNP negotiation (host to device or device to host). This triggers a USB interrupt if UOTGHS_CTRL.ROLEEXE is one.

This bit is cleared when the UOTGHS_SCR.ROLEEXIC bit is written to one.

- **BCERRI: B-Connection Error Interrupt**

This bit is set when an error occurs during the B-connection. This triggers a USB interrupt if UOTGHS_CTRL.BCERRE is one.

This bit is cleared when the UOTGHS_SCR.BCERRIC bit is written to one.

This bit shall only be used in host mode.

- **VBERRI: VBus Error Interrupt**

This bit is set when a VBus drop has been detected. This triggers a USB interrupt if UOTGHS_CTRL.VBERRE is one.

This bit is cleared when the UOTGHS_SCR.VBERRIC bit is written to one.

This bit shall only be used in host mode.

If a VBus problem occurs, then the VBERRI interrupt is generated even if the UOTGHS does not go to an error state because UOTGHS_CTRL.VBUSHWC is one.

- **SRPI: SRP Interrupt**

This bit is set when an SRP has been detected. This triggers a USB interrupt if UOTGHS_CTRL.SRPE is one.

This bit is cleared when the UOTGHS_SCR.SRPIC bit is written to one.

This bit shall only be used in host mode.

- **VBUSTI: VBus Transition Interrupt**

This bit is set when a transition (high to low, low to high) has been detected on the VBUS pad. This triggers a USB interrupt if UOTGHS_CTRL.VBUSTE is one.

This bit is cleared when the UOTGHS_SCR.VBUSTIC bit is written to one.

This interrupt is generated even if the clock is frozen by the UOTGHS_CTRL.FRZCLK bit.

- **IDTI: ID Transition Interrupt**

This bit is set when a transition (high to low, low to high) has been detected on the UOTGID input pin. This triggers a USB interrupt if UOTGHS_CTRL.IDTE is one.

This bit is cleared when the UOTGHS_SCR.IDTIC bit is written to one.

This interrupt is generated even if the clock is frozen by the UOTGHS_CTRL.FRZCLK bit.

39.6.1.3 General Status Clear Register

Name: UOTGHS_SCR

Address: 0x400AC808

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
–	–	–	–	–	–	VBUSRQC	–
7	6	5	4	3	2	1	0
STOIC	HNPERRIC	ROLEEXIC	BCERRIC	VBERRIC	SRPIC	VBUSTIC	IDTIC

- **VBUSRQC: VBus Request Clear**

Writing a one will clear VBUSRQ bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **STOIC: Suspend Time-Out Interrupt Clear**

Writing a one will clear STOI bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **HNPERRIC: HNP Error Interrupt Clear**

Writing a one twill clear HNPERRI bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **ROLEEXIC: Role Exchange Interrupt Clear**

Writing a one twill clear ROLEEXI bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **BCERRIC: B-Connection Error Interrupt Clear**

Writing a one will clear BCERRI bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **VBERRIC: VBus Error Interrupt Clear**

Writing a one will clear VBERRI bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **SRPIC: SRP Interrupt Clear**

Writing a one will clear SRPI bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **VBUSTIC: VBus Transition Interrupt Clear**

Writing a one will clear VBUSTI bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **IDTIC: ID Transition Interrupt Clear**

Writing a one will clear IDTI bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

39.6.1.4 General Status Set Register

Name: UOTGHS_SFR

Address: 0x400AC80C

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
–	–	–	–	–	–	VBUSRQS	–
7	6	5	4	3	2	1	0
STOIS	HNPERRIS	ROLEEXIS	BCERRIS	VBERRIS	SRPIS	VBUSTIS	IDTIS

- **VBUSRQS: VBus Request Set**

Writing a one will set VBUSRQ bit in UOTGHS_SR.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **STOIS: Suspend Time-Out Interrupt Set**

Writing a one will set STOI bit in UOTGHS_SR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **HNPERRIS: HNP Error Interrupt Set**

Writing a one will set HNPERRI bit in UOTGHS_SR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **ROLEEXIS: Role Exchange Interrupt Set**

Writing a one will set ROLEEXI bit in UOTGHS_SR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **BCERRIS: B-Connection Error Interrupt Set**

Writing a one will set BCERRI bit in UOTGHS_SR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **VBERRIS: VBus Error Interrupt Set**

Writing a one will set VBERRI bit in UOTGHS_SR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **SRPIS: SRP Interrupt Set**

Writing a one will set SRPI bit in UOTGHS_SR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **VBUSTIS: VBus Transition Interrupt Set**

Writing a one will set VBUSTI bit in UOTGHS_SR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always read as zero.

- **IDTIS: ID Transition Interrupt Set**

Writing a one will set IDTI bit in UOTGHS_SR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always read as zero.

39.6.1.5 General Finite State Machine Register

Name: UOTGHS_FSM

Address: 0x400AC82C

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
–	–	–	–	DRDSTATE			

- **DRDSTATE**

This field indicates the state of the UOTGHS.

Refer to the OTG specification for more details.

Value	Name	Description
0	A_IDLESTATE	This is the start state for A-devices (when the ID pin is 0)
1	A_WAIT_VRISE	In this state, the A-device waits for the voltage on VBus to rise above the A-device VBus Valid threshold (4.4 V).
2	A_WAIT_BCON	In this state, the A-device waits for the B-device to signal a connection.
3	A_HOST	In this state, the A-device that operates in Host mode is operational.
4	A_SUSPEND	The A-device operating as a host is in the suspend mode.
5	A_PERIPHERAL	The A-device operates as a peripheral.
6	A_WAIT_VFALL	In this state, the A-device waits for the voltage on VBus to drop below the A-device Session Valid threshold (1.4 V).
7	A_VBUS_ERR	In this state, the A-device waits for recovery of the over-current condition that caused it to enter this state.
8	A_WAIT_DISCHARGE	In this state, the A-device waits for the data USB line to discharge (100 us).
9	B_IDLE	This is the start state for B-device (when the ID pin is 1).
10	B_PERIPHERAL	In this state, the B-device acts as the peripheral.
11	B_WAIT_BEGIN_HNP	In this state, the B-device is in suspend mode and waits until 3 ms before initiating the HNP protocol if requested.
12	B_WAIT_DISCHARGE	In this state, the B-device waits for the data USB line to discharge (100 us) before becoming Host.
13	B_WAIT_ACON	In this state, the B-device waits for the A-device to signal a connect before becoming B-Host.
14	B_HOST	In this state, the B-device acts as the Host.
15	B_SRP_INIT	In this state, the B-device attempts to start a session using the SRP protocol.

39.6.2 USB Device Registers

39.6.2.1 Device General Control Register

Name: UOTGHS_DEVCTRL

Address: 0x400AC000

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	OPMODE2
15	14	13	12	11	10	9	8
TSTPCKT	TSTK	TSTJ	LS	SPDCONF		RMWKUP	DETACH
7	6	5	4	3	2	1	0
ADDEN	UADD						

- **OPMODE2: Specific Operational mode**

0: The UTMI transceiver is in normal operation mode.

1: The UTMI transceiver is in the «disable bit stuffing and NRZI encoding» operational mode for test purpose.

- **TSTPCKT: Test packet mode**

0: The UTMI transceiver is in normal operation mode.

1: The UTMI transceiver generates test packets for test purpose.

- **TSTK: Test mode K**

0: The UTMI transceiver is in normal operation mode.

1: The UTMI transceiver generates high-speed K state for test purpose.

- **TSTJ: Test mode J**

0: The UTMI transceiver is in normal operation mode.

1: The UTMI transceiver generates high-speed J state for test purpose.

- **LS: Low-Speed Mode Force**

0: The full-speed mode is active.

1: The low-speed mode is active.

This bit can be written even if UOTGHS_CTRL.USBE is zero or UOTGHS_CTRL.FRZCLK is one. Disabling the UOTGHS (by writing a zero to the UOTGHS_CTRL.USBE bit) does not reset this bit.

- **SPDCONF: Mode Configuration**

This field contains the peripheral speed:

Value	Name	Description
0	NORMAL	The peripheral starts in full-speed mode and performs a high-speed reset to switch to the high-speed mode if the host is high-speed capable.
1	LOW_POWER	For a better consumption, if high-speed is not needed.
2	HIGH_SPEED	Forced high speed.
3	FORCED_FS	The peripheral remains in full-speed mode whatever the host speed capability.

- **RMWKUP: Remote Wake-Up**

Writing a one to this bit will send an upstream resume to the host for a remote wake-up.

Writing a zero to this bit has no effect.

This bit is cleared when the UOTGHS receives a USB reset or once the upstream resume has been sent.

- **DETACH: Detach**

Writing a one to this bit will physically detach the device (disconnect internal pull-up resistor from D+ and D-).

Writing a zero to this bit will reconnect the device.

- **ADDEN: Address Enable**

Writing a one to this bit will activate the UADD field (USB address).

Writing a zero to this bit has no effect.

This bit is cleared when a USB reset is received.

- **UADD: USB Address**

This field contains the device address.

This field is cleared when a USB reset is received.

39.6.2.2 Device Global Interrupt Status Register

Name: UOTGHS_DEVISR

Address: 0x400AC004

Access: Read-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	PEP_9	PEP_8	PEP_7	PEP_6	PEP_5	PEP_4
15	14	13	12	11	10	9	8
PEP_3	PEP_2	PEP_1	PEP_0	–	–	–	–
7	6	5	4	3	2	1	0
–	UPRSM	EORSM	WAKEUP	EORST	SOF	MSOF	SUSP

- **DMA_x: DMA Channel x Interrupt**

This bit is set when an interrupt is triggered by the DMA channel x. This triggers a USB interrupt if DMA_x is one.

This bit is cleared when the UOTGHS_DEVDMASTATUSx interrupt source is cleared.

- **PEP_x: Endpoint x Interrupt**

This bit is set when an interrupt is triggered by the endpoint x (UOTGHS_DEVEPTISRx, UOTGHS_DEVEPTIMRx). This triggers a USB interrupt if UOTGHS_DEVIMR.PEP_x is one.

This bit is cleared when the interrupt source is serviced.

- **UPRSM: Upstream Resume Interrupt**

This bit is set when the UOTGHS sends a resume signal called “Upstream Resume”. This triggers a USB interrupt if UOTGHS_DEVIMR.UPRSME is one.

This bit is cleared when the UOTGHS_DEVICR.UPRSMC bit is written to one to acknowledge the interrupt (USB clock inputs must be enabled before).

- **EORSM: End of Resume Interrupt**

This bit is set when the UOTGHS detects a valid “End of Resume” signal initiated by the host. This triggers a USB interrupt if UOTGHS_DEVIMR.EORSME is one.

This bit is cleared when the UOTGHS_DEVICR.EORSMC bit is written to one to acknowledge the interrupt.

- **WAKEUP: Wake-Up Interrupt**

This bit is set when the UOTGHS is reactivated by a filtered non-idle signal from the lines (not by an upstream resume). This triggers an interrupt if UOTGHS_DEVIMR.WAKEUPE is one.

This bit is cleared when the UOTGHS_DEVICR.WAKEUPC bit is written to one to acknowledge the interrupt (USB clock inputs must be enabled before).

This bit is cleared when the Suspend (SUSP) interrupt bit is set.

This interrupt is generated even if the clock is frozen by the UOTGHS_CTRL.FRZCLK bit.

- **EORST: End of Reset Interrupt**

This bit is set when a USB “End of Reset” has been detected. This triggers a USB interrupt if UOTGHS_DEVIMR.EORSTE is one.

This bit is cleared when the UOTGHS_DEVICR.EORSTC bit is written to one to acknowledge the interrupt.

- **SOF: Start of Frame Interrupt**

This bit is set when a USB “Start of Frame” PID (SOF) has been detected (every 1 ms). This triggers a USB interrupt if SOFE is one. The FNUM field is updated. In High-speed mode, the MFNUM field is cleared.

This bit is cleared when the UOTGHS_DEVICR.SOFC bit is written to one to acknowledge the interrupt.

- **MSOF: Micro Start of Frame Interrupt**

This bit is set in High-speed mode when a USB “Micro Start of Frame” PID (SOF) has been detected (every 125 us). This triggers a USB interrupt if MSOFE is one. The MFNUM field is updated. The FNUM field is unchanged.

This bit is cleared when the UOTGHS_DEVICR.MSOFC bit is written to one to acknowledge the interrupt.

- **SUSP: Suspend Interrupt**

This bit is set when a USB “Suspend” idle bus state has been detected for 3 frame periods (J state for 3 ms). This triggers a USB interrupt if UOTGHS_DEVIMR.SUSPE is one.

This bit is cleared when the UOTGHS_DEVICR.SUSPC bit is written to one to acknowledge the interrupt.

This bit is cleared when the Wake-Up (WAKEUP) interrupt bit is set.

39.6.2.3 Device Global Interrupt Clear Register

Name: UOTGHS_DEVICR

Address: 0x400AC008

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
–	UPRSMC	EORSMC	WAKEUPC	EORSTC	SOFC	MSOFC	SUSPC

- **UPRSMC: Upstream Resume Interrupt Clear**

Writing a one to this bit will clear UPRSM bit in UOTGHS_DEVISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EORSMC: End of Resume Interrupt Clear**

Writing a one to this bit will clear EORSM bit in UOTGHS_DEVISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **WAKEUPC: Wake-Up Interrupt Clear**

Writing a one to this bit will clear WAKEUP bit in UOTGHS_DEVISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EORSTC: End of Reset Interrupt Clear**

Writing a one to this bit will clear EORST bit in UOTGHS_DEVISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SOFC: Start of Frame Interrupt Clear**

Writing a one to this bit will clear SOF bit in UOTGHS_DEVISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **MSOFC: Micro Start of Frame Interrupt Clear**

Writing a one to this bit will clear MSOF bit in UOTGHS_DEVISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SUSPC: Suspend Interrupt Clear**

Writing a one to this bit will clear SUSP bit in UOTGHS_DEVISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.2.4 Device Global Interrupt Set Register

Name: UOTGHS_DEVIFR

Address: 0x400AC00C

Access: Write-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	UPRSMS	EORSMS	WAKEUPS	EORSTS	SOFS	MSOFS	SUSPS

- **DMA_x: DMA Channel x Interrupt Set**

Writing a one to this bit will set the corresponding bit in UOTGHS_DEVISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UPRSMS: Upstream Resume Interrupt Set**

Writing a one to this bit will set UPRSM bit in UOTGHS_DEVISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EORSMS: End of Resume Interrupt Set**

Writing a one to this bit will set EORSM bit in UOTGHS_DEVISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **WAKEUPS: Wake-Up Interrupt Set**

Writing a one to this bit will set WAKEUP bit in UOTGHS_DEVISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EORSTS: End of Reset Interrupt Set**

Writing a one to this bit will set EORST bit in UOTGHS_DEVISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SOFS: Start of Frame Interrupt Set**

Writing a one to this bit will set SOF bit in UOTGHS_DEVISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **MSOFS: Micro Start of Frame Interrupt Set**

Writing a one to this bit will set MSOF bit in UOTGHS_DEVISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SUSPS: Suspend Interrupt Set**

Writing a one to this bit will set SUSP bit in UOTGHS_DEVISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.2.5 Device Global Interrupt Mask Register

Name: UOTGHS_DEVIMR

Address: 0x400AC010

Access: Read-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	PEP_9	PEP_8	PEP_7	PEP_6	PEP_5	PEP_4
15	14	13	12	11	10	9	8
PEP_3	PEP_2	PEP_1	PEP_0	–	–	–	–
7	6	5	4	3	2	1	0
–	UPRSME	EORSME	WAKEUPE	EORSTE	SOFE	MSOFE	SUSPE

- **DMA_x: DMA Channel x Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when DMA_x bit in UOTGHS_DEVIER is written to one.

This bit is cleared when DMA_x bit in UOTGHS_DEVIDR is written to one.

- **PEP_x: Endpoint x Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when PEP_x bit in UOTGHS_DEVIER is written to one.

This bit is cleared when PEP_x bit in UOTGHS_DEVIDR is written to one.

- **UPRSME: Upstream Resume Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when UPRSMES bit in UOTGHS_DEVIER is written to one.

This bit is cleared when UPRSMEC bit in UOTGHS_DEVIDR is written to one.

- **EORSME: End of Resume Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when EORSMES bit in UOTGHS_DEVIER is written to one.

This bit is cleared when EORSMEC bit in UOTGHS_DEVIDR is written to one.

- **WAKEUPE: Wake-Up Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when WAKEUPES bit in UOTGHS_DEVIER is written to one.

This bit is cleared when WAKEUPEC bit in UOTGHS_DEVIDR is written to one.

- **EORSTE: End of Reset Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when EORSTES bit in UOTGHS_DEVIER is written to one.

This bit is cleared when EORSTEC bit in UOTGHS_DEVIDR is written to one.

- **SOFE: Start of Frame Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when SOFES bit in UOTGHS_DEVIER is written to one.

This bit is cleared when SOFEC bit in UOTGHS_DEVIDR is written to one.

- **MSOFE: Micro Start of Frame Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when MSOFES bit in UOTGHS_DEVIER is written to one.

This bit is cleared when MSOFEC bit in UOTGHS_DEVIDR is written to one.

- **SUSPE: Suspend Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

This bit is set when SUSPES bit in UOTGHS_DEVIER is written to one.

This bit is cleared when SUSPEC bit in UOTGHS_DEVIDR is written to one.

39.6.2.6 Device Global Interrupt Disable Register

Name: UOTGHS_DEVIDR

Address: 0x400AC014

Access: Write-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	PEP_9	PEP_8	PEP_7	PEP_6	PEP_5	PEP_4
15	14	13	12	11	10	9	8
PEP_3	PEP_2	PEP_1	PEP_0	–	–	–	–
7	6	5	4	3	2	1	0
–	UPRSMEC	EORSMEC	WAKEUPEC	EORSTEC	SOFEC	MSOFEC	SUSPEC

- **DMA_x: DMA Channel x Interrupt Disable**

Writing a one to this bit will clear the corresponding bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PEP_x: Endpoint x Interrupt Disable**

Writing a one to this bit will clear the corresponding bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UPRSMEC: Upstream Resume Interrupt Disable**

Writing a one to this bit will clear UPRSME bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EORSMEC: End of Resume Interrupt Disable**

Writing a one to this bit will clear EORSME bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **WAKEUPEC: Wake-Up Interrupt Disable**

Writing a one to this bit will clear WAKEUPE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EORSTEC: End of Reset Interrupt Disable**

Writing a one to this bit will clear EORSTE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SOFEC: Start of Frame Interrupt Disable**

Writing a one to this bit will clear SOFE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **MSOFEC: Micro Start of Frame Interrupt Disable**

Writing a one to this bit will clear MSOFE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SUSPEC: Suspend Interrupt Disable**

Writing a one to this bit will clear SUSPE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.2.7 Device Global Interrupt Enable Register

Name: UOTGHS_DEVIER

Address: 0x400AC018

Access: Write-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	PEP_9	PEP_8	PEP_7	PEP_6	PEP_5	PEP_4
15	14	13	12	11	10	9	8
PEP_3	PEP_2	PEP_1	PEP_0	–	–	–	–
7	6	5	4	3	2	1	0
–	UPRSMES	EORSMES	WAKEUPES	EORSTES	SOFES	MSOFES	SUSPES

- **DMA_x: DMA Channel x Interrupt Enable**

Writing a one to this bit will set the corresponding bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PEP_x: Endpoint x Interrupt Enable**

Writing a one to this bit will set the corresponding bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UPRSMES: Upstream Resume Interrupt Enable**

Writing a one to this bit will set UPRSME bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EORSMES: End of Resume Interrupt Enable**

Writing a one to this bit will set EORSME bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **WAKEUPES: Wake-Up Interrupt Enable**

Writing a one to this bit will set WAKEUPE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EORSTES: End of Reset Interrupt Enable**

Writing a one to this bit will set EORSTE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SOFES: Start of Frame Interrupt Enable**

Writing a one to this bit will set SOFE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **MSOFES: Micro Start of Frame Interrupt Enable**

Writing a one to this bit will set MSOFE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SUSPES: Suspend Interrupt Enable**

Writing a one to this bit will set SUSPE bit in UOTGHS_DEVIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.2.8 Device Endpoint Register

Name: UOTGHS_DEVEPT

Address: 0x400AC01C

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	EPRST8
23	22	21	20	19	18	17	16
EPRST7	EPRST6	EPRST5	EPRST4	EPRST3	EPRST2	EPRST1	EPRST0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	EPEN8
7	6	5	4	3	2	1	0
EPEN7	EPEN6	EPEN5	EPEN4	EPEN3	EPEN2	EPEN1	EPEN0

- **EPRSTx: Endpoint x Reset**

Writing a one to this bit will reset the endpoint x FIFO prior to any other operation, upon hardware reset or when a USB bus reset has been received. This resets the endpoint x registers (UOTGHS_DEVEPTCFGx, UOTGHS_DEVEPTISRx, UOTGHS_DEVEPTIMRx) but not the endpoint configuration (UOTGHS_DEVEPTCFGx.ALLOC, UOTGHS_DEVEPTCFGx.EPBK, UOTGHS_DEVEPTCFGx.EPSIZE, UOTGHS_DEVEPTCFGx.EPDIR, UOTGHS_DEVEPTCFGx.EPTYPE).

All the endpoint mechanism (FIFO counter, reception, transmission, etc.) is reset apart from the Data Toggle Sequence field (UOTGHS_DEVEPTISRx.DTSEQ) which can be cleared by setting the UOTGHS_DEVEPTIMRx.RSTDT bit (by writing a one to the UOTGHS_DEVEPTIERx.RSTDTS bit).

The endpoint configuration remains active and the endpoint is still enabled.

Writing a zero to this bit will complete the reset operation and start using the FIFO.

This bit is cleared upon receiving a USB reset.

- **EPENx: Endpoint x Enable**

0: The endpoint x is disabled, what forces the endpoint x state to inactive (no answer to USB requests) and resets the endpoint x registers (UOTGHS_DEVEPTCFGx, UOTGHS_DEVEPTISRx, UOTGHS_DEVEPTIMRx) but not the endpoint configuration (UOTGHS_DEVEPTCFGx.ALLOC, UOTGHS_DEVEPTCFGx.EPBK, UOTGHS_DEVEPTCFGx.EPSIZE, UOTGHS_DEVEPTCFGx.EPDIR, UOTGHS_DEVEPTCFGx.EPTYPE).

1: The endpoint x is enabled.

39.6.2.9 Device Frame Number Register

Name: UOTGHS_DEVFNUM

Address: 0x400AC020

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
FNCERR	–	FNUM					
7	6	5	4	3	2	1	0
FNUM					MFNUM		

- **FNCERR: Frame Number CRC Error**

This bit is set when a corrupted frame number (or micro-frame number) is received. This bit and the SOF (or MSOF) interrupt bit are updated at the same time.

This bit is cleared upon receiving a USB reset.

- **FNUM: Frame Number**

This field contains the 11-bit frame number information. It is provided in the last received SOF packet.

This field is cleared upon receiving a USB reset.

FNUM is updated even if a corrupted SOF is received.

- **MFNUM: Micro Frame Number**

This field contains the 3-bit micro frame number information. It is provided in the last received MSOF packet.

This field is cleared at the beginning of each start of frame (SOF interrupt) or upon receiving a USB reset.

MFNUM is updated even if a corrupted MSOF is received.

39.6.2.10 Device Endpoint x Configuration Register

Name: UOTGHS_DEVEPTCFGx [x=0..9]

Address: 0x400AC100

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
–	NBTRANS		EPTYPE		–	AUTOSW	EPDIR
7	6	5	4	3	2	1	0
–	EPSIZE			EPBK		ALLOC	–

- **NBTRANS: Number of transaction per microframe for isochronous endpoint**

This field shall be written to the number of transaction per microframe to perform high-bandwidth isochronous transfer

This field can be written only for endpoint that have this capability (see UOTGHS_FEATURES.ENHBISOx bit). This field is 0 otherwise.

This field is irrelevant for non-isochronous endpoint.

Value	Name	Description
0	0_TRANS	reserved to endpoint that does not have the high-bandwidth isochronous capability.
1	1_TRANS	default value: one transaction per micro-frame.
2	2_TRANS	2 transactions per micro-frame. This endpoint should be configured as double-bank.
3	3_TRANS	3 transactions per micro-frame. This endpoint should be configured as triple-bank.

- **EPTYPE: Endpoint Type**

This field shall be written to select the endpoint type:

Value	Name	Description
0	CTRL	Control
1	ISO	Isochronous
2	BLK	Bulk
3	INTRPT	Interrupt

This field is cleared upon receiving a USB reset.

- **AUTOSW: Automatic Switch**

This bit is cleared upon receiving a USB reset.

0: The automatic bank switching is disabled.

1: The automatic bank switching is enabled.

- **EPDIR: Endpoint Direction**

This bit is cleared upon receiving a USB reset.

0 (OUT): The endpoint direction is OUT.

1 (IN): The endpoint direction is IN (nor for control endpoints).

- **EPSIZE: Endpoint Size**

This field shall be written to select the size of each endpoint bank:

Value	Name	Description
0	8_BYTE	8 bytes
1	16_BYTE	16 bytes
2	32_BYTE	32 bytes
3	64_BYTE	64 bytes
4	128_BYTE	128 bytes
5	256_BYTE	256 bytes
6	512_BYTE	512 bytes
7	1024_BYTE	1024 bytes

This field is cleared upon receiving a USB reset (except for the endpoint 0).

- **EPBK: Endpoint Banks**

This field shall be written to select the number of banks for the endpoint:

Value	Name	Description
0	1_BANK	Single-bank endpoint
1	2_BANK	Double-bank endpoint
2	3_BANK	Triple-bank endpoint
3		Reserved

For control endpoints, a single-bank endpoint (0b00) shall be selected.

This field is cleared upon receiving a USB reset (except for the endpoint 0).

- **ALLOC: Endpoint Memory Allocate**

Writing a one to this bit will allocate the endpoint memory. The user should check the UOTGHS_DEVEPTISRx.CFGOK bit to know whether the allocation of this endpoint is correct.

Writing a zero to this bit will free the endpoint memory.

This bit is cleared upon receiving a USB reset (except for the endpoint 0).

39.6.2.11 Device Endpoint x Status Register

Name: UOTGHS_DEVEPTISRx [x=0..9]

Address: 0x400AC130

Access: Read-only 0x0100

31	30	29	28	27	26	25	24
–	BYCT						
23	22	21	20	19	18	17	16
BYCT				–	CFGOK	CTRLDIR	RWALL
15	14	13	12	11	10	9	8
CURRBK		NBUSYBK		–	ERRORTRANS	DTSEQ	
7	6	5	4	3	2	1	0
SHORTPACKET	STALLEDI/ CRCERRI	OVERFI	NAKINI/ HBISOFLUSHI	NAKOUTI/ HBISOINERRI	RXSTPI/ UNDERFI	RXOUTI	TXINI

- **BYCT: Byte Count**

This field is set with the byte count of the FIFO.

For IN endpoints, incremented after each byte written by the software into the endpoint and decremented after each byte sent to the host.

For OUT endpoints, incremented after each byte received from the host and decremented after each byte read by the software from the endpoint.

This field may be updated one clock cycle after the RWALL bit changes, so the user should not poll this field as an interrupt bit.

- **CFGOK: Configuration OK Status**

This bit is updated when the UOTGHS_DEVEPTCFGx.ALLOC bit is written to one.

This bit is set if the endpoint x number of banks (UOTGHS_DEVEPTCFGx.EPBK) and size (UOTGHS_DEVEPTCFGx.EPSIZE) are correct compared to the maximal allowed number of banks and size for this endpoint and to the maximal FIFO size (i.e. the DPRAM size).

If this bit is cleared, the user shall rewrite correct values to the UOTGHS_DEVEPTCFGx.EPBK and UOTGHS_DEVEPTCFGx.EPSIZE fields.

- **CTRLDIR: Control Direction**

This bit is set after a SETUP packet to indicate that the following packet is an IN packet.

This bit is cleared after a SETUP packet to indicate that the following packet is an OUT packet.

Writing a zero or a one to this bit has no effect.

- **RWALL: Read-write Allowed**

This bit is set for IN endpoints when the current bank is not full, i.e., the user can write further data into the FIFO.

This bit is set for OUT endpoints when the current bank is not empty, i.e., the user can read further data from the FIFO.

This bit is never set if UOTGHS_DEVEPTIMRx.STALLRQ is one or in case of error.

This bit is cleared otherwise.

This bit shall not be used for control endpoints.

- **CURRBK: Current Bank**

This bit is set for non-control endpoints, to indicate the current bank:

Value	Name	Description
0	BANK0	Current bank is bank0
1	BANK1	Current bank is bank1
2	BANK2	Current bank is bank2
3		Reserved

This field may be updated one clock cycle after the RWALL bit changes, so the user should not poll this field as an interrupt bit.

- **NBUSYBK: Number of Busy Banks**

This field is set to indicate the number of busy banks:

Value	Name	Description
0	0_BUSY	0 busy bank (all banks free)
1	1_BUSY	1 busy bank
2	2_BUSY	2 busy banks
3	3_BUSY	3 busy banks

For IN endpoints, it indicates the number of banks filled by the user and ready for IN transfer. When all banks are free, this triggers a PEP_x interrupt if NBUSYBKE is one.

For OUT endpoints, it indicates the number of banks filled by OUT transactions from the host. When all banks are busy, this triggers a PEP_x interrupt if NBUSYBKE is one.

When the UOTGHS_DEVEPTIMRx.FIFOCON bit is cleared (by writing a one to the UOTGHS_DEVEPTIMRx.FIFOCONC bit) to validate a new bank, this field is updated two or three clock cycles later to calculate the address of the next bank.

A PEP_x interrupt is triggered if:

- for IN endpoint, UOTGHS_DEVEPTIMRx.NBUSYBKE is one and all the banks are free.
- for OUT endpoint, UOTGHS_DEVEPTIMRx.NBUSYBKE is one and all the banks are busy.

- **ERRORTRANS: High-bandwidth isochronous OUT endpoint transaction error Interrupt**

This bit is set when a transaction error occurs during the current micro-frame (the data toggle sequencing does not respect the USB 2.0 standard). This triggers a PEP_x interrupt if UOTGHS_DEVEPTIMRx.ERRORTRANSE is one.

This bit is set as long as the current bank (CURRBK) belongs to the bad n-transactions (n=1,2 or 3) transferred during the micro-frame. Shall be cleared by software by clearing (at least once) the UOTGHS_DEVEPTIMRx.FIFOCON bit to switch to the bank that belongs to the next n-transactions (next micro-frame).

- **DTSEQ: Data Toggle Sequence**

This field is set to indicate the PID of the current bank:

Value	Name	Description
0	DATA0	Data0 toggle sequence
1	DATA1	Data1 toggle sequence
2	DATA2	Data2 toggle sequence (for high-bandwidth isochronous endpoint)
3	MDATA	MData toggle sequence (for high-bandwidth isochronous endpoint)

For IN transfers, it indicates the data toggle sequence that will be used for the next packet to be sent. This is not relative to the current bank.

For OUT transfers, this value indicates the last data toggle sequence received on the current bank.

By default, DTSEQ is 0b01, as if the last data toggle sequence was Data1, so the next sent or expected data toggle sequence should be Data0.

For High-bandwidth isochronous endpoint, a PEP_x interrupt is triggered if:

- UOTGHS_DEVEPTIMRx.MDATAE is one and a MData packet has been received (DTSEQ=MData and UOTGHS_DEVEPTISRx.RXOUTI is one).
- UOTGHS_DEVEPTISRx.DATAXE is one and a Data0/1/2 packet has been received (DTSEQ=Data0/1/2 and UOTGHS_DEVEPTISRx.RXOUTI is one)

- **SHORTPACKET: Short Packet Interrupt**

This bit is set for non-control OUT endpoints, when a short packet has been received.

This triggers a PEP_x interrupt if UOTGHS_DEVEPTIMRx.SHORTPACKETE is one.

This bit is cleared when the SHORTPACKETC bit is written to one. This will acknowledge the interrupt.

- **STALLEDI: STALLed Interrupt**

This bit is set to signal that a STALL handshake has been sent. To do that, the software has to set the STALLRQ bit (by writing a one to the STALLRQS bit). This triggers a PEP_x interrupt if STALLEDE is one.

This bit is cleared when the STALLEDIC bit is written to one. This will acknowledge the interrupt.

- **CRCERRI: CRC Error Interrupt**

This bit is set to signal that a CRC error has been detected in an isochronous OUT endpoint. The OUT packet is stored in the bank as if no CRC error had occurred. This triggers a PEP_x interrupt if CRCERRE is one.

This bit is cleared when the CRCERRIC bit is written to one. This will acknowledge the interrupt.

- **OVERFI: Overflow Interrupt**

This bit is set when an overflow error occurs. This triggers a PEP_x interrupt if OVERFE is one.

For all endpoint types, an overflow can occur during OUT stage if the host attempts to write into a bank that is too small for the packet. The packet is acknowledged and the UOTGHS_DEVEPTISRx.RXOUTI bit is set as if no overflow had occurred. The bank is filled with all the first bytes of the packet that fit in.

This bit is cleared when the OVERFIC bit is written to one. This will acknowledge the interrupt.

- **NAKINI: NAKed IN Interrupt**

This bit is set when a NAK handshake has been sent in response to an IN request from the host. This triggers a PEP_x interrupt if NAKINE is one.

This bit is cleared when the NAKINIC bit is written to one. This will acknowledge the interrupt.

- **HBISOFLUSHI: High Bandwidth Isochronous IN Flush Interrupt**

This bit is set, for High-bandwidth isochronous IN endpoint (with NBTRANS=2 or 3), at the end of the micro-frame, if less than N transaction has been completed by the UOTGHS without underflow error. This may occur in case of a missing IN token. In this case, the bank are flushed out to ensure the data synchronization between the host and the device. This triggers a PEP_x interrupt if HBISOFLUSHE is one.

This bit is cleared when the HBISOFLUSHIC bit is written to one. This will acknowledge the interrupt.

- **NAKOUTI: NAKed OUT Interrupt**

This bit is set when a NAK handshake has been sent in response to an OUT request from the host. This triggers a PEP_x interrupt if NAKOUTE is one.

This bit is cleared when the NAKOUTIC bit is written to one. This will acknowledge the interrupt.

- **HBISOINERRI: High bandwidth isochronous IN Underflow Error Interrupt**

This bit is set, for High-bandwidth isochronous IN endpoint (with NBTRANS=2 or 3), at the end of the microframe, if less than N bank was written by the cpu within this micro-frame. This triggers a PEP_x interrupt if HBISOINERRE is one.

This bit is cleared when the HBISOINERRIC bit is written to one. This will acknowledge the interrupt.

- **UNDERFI: Underflow Interrupt**

This bit is set, for isochronous IN/OUT endpoints, when an underflow error occurs. This triggers a PEP_x interrupt if UNDERFE is one.

An underflow can occur during IN stage if the host attempts to read from an empty bank. A zero-length packet is then automatically sent by the UOTGHS.

An underflow can also occur during OUT stage if the host sends a packet while the bank is already full. Typically, the CPU is not fast enough. The packet is lost.

Shall be cleared by writing a one to the UNDERFIC bit. This will acknowledge the interrupt.

This bit is inactive (cleared) for bulk and interrupt IN/OUT endpoints and it means RXSTPI for control endpoints.

- **RXSTPI: Received SETUP Interrupt**

This bit is set, for control endpoints, to signal that the current bank contains a new valid SETUP packet. This triggers a PEP_x interrupt if RXSTPE is one.

Shall be cleared by writing a one to the RXSTPIC bit. This will acknowledge the interrupt and free the bank.

This bit is inactive (cleared) for bulk and interrupt IN/OUT endpoints and it means UNDERFI for isochronous IN/OUT endpoints.

- **RXOUTI: Received OUT Data Interrupt**

This bit is set, for control endpoints, when the current bank contains a bulk OUT packet (data or status stage). This triggers a PEP_x interrupt if UOTGHS_DEVEPTIMRx.RXOUTE is one.

Shall be cleared for control end points, by writing a one to the RXOUTIC bit. This will acknowledge the interrupt and free the bank.

This bit is set for isochronous, bulk and, interrupt OUT endpoints, at the same time as UOTGHS_DEVEPTIMRx.FIFOCON when the current bank is full. This triggers a PEP_x interrupt if UOTGHS_DEVEPTIMRx.RXOUTE is one.

Shall be cleared for isochronous, bulk and, interrupt OUT endpoints, by writing a one to the RXOUTIC bit. This will acknowledge the interrupt, what has no effect on the endpoint FIFO.

The user then reads from the FIFO and clears the UOTGHS_DEVEPTIMRx.FIFOCON bit to free the bank. If the OUT endpoint is composed of multiple banks, this also switches to the next bank. The UOTGHS_DEVEPTISRx.RXOUTI and UOTGHS_DEVEPTIMRx.FIFOCON bits are set/cleared in accordance with the status of the next bank.

UOTGHS_DEVEPTISRx.RXOUTI shall always be cleared before clearing UOTGHS_DEVEPTIMRx.FIFOCON.

This bit is inactive (cleared) for isochronous, bulk and interrupt IN endpoints.

- **TXINI: Transmitted IN Data Interrupt**

This bit is set for control endpoints, when the current bank is ready to accept a new IN packet. This triggers a PEP_x interrupt if TXINE is one.

This bit is cleared when the TXINIC bit is written to one. This will acknowledge the interrupt and send the packet.

This bit is set for isochronous, bulk and interrupt IN endpoints, at the same time as UOTGHS_DEVEPTIMRx.FIFOCON when the current bank is free. This triggers a PEP_x interrupt if TXINE is one.

This bit is cleared when the TXINIC bit is written to one. This will acknowledge the interrupt, what has no effect on the endpoint FIFO.

The user then writes into the FIFO and clears the UOTGHS_DEVEPTIMRx.FIFOCON bit to allow the UOTGHS to send the data. If the IN endpoint is composed of multiple banks, this also switches to the next bank. The UOTGHS_DEVEPTISRx.TXINI and UOTGHS_DEVEPTIMRx.FIFOCON bits are set/cleared in accordance with the status of the next bank.

UOTGHS_DEVEPTISRx.TXINI shall always be cleared before clearing UOTGHS_DEVEPTIMRx.FIFOCON.

This bit is inactive (cleared) for isochronous, bulk and interrupt OUT endpoints.

39.6.2.12 Device Endpoint x Clear Register

Name: UOTGHS_DEVEPTICRx [x=0..9]

Address: 0x400AC160

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
SHORT PACKETC	STALLEDIC/ CRCERRIC	OVERFIC	NAKINIC/ HBISOFLUSHIC	NAKOUTIC/ HBISOINERRIC	RXSTPIC/ UNDERFIC	RXOUTIC	TXINIC

- **SHORTPACKETC: Short Packet Interrupt Clear**

Writing a one to this bit will clear SHORTPACKET in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **STALLEDIC: STALLed Interrupt Clear**

Writing a one to this bit will clear STALLEDI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **CRCERRIC: CRC Error Interrupt Clear**

Writing a one to this bit will clear CRCERRI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **OVERFIC: Overflow Interrupt Clear**

Writing a one to this bit will clear OVERFI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKINIC: NAKed IN Interrupt Clear**

Writing a one to this bit will clear NAKINI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HBISOFLUSHIC: High Bandwidth Isochronous IN Flush Interrupt Clear**

Writing a one to this bit will clear **HBISOFLUSHI** in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKOUTIC: NAKed OUT Interrupt Clear**

Writing a one to this bit will clear NAKOUTI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HBISOINERRIC: High bandwidth isochronous IN Underflow Error Interrupt Clear**

Writing a one to this bit will clear **HBISOINERRI** in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXSTPIC: Received SETUP Interrupt Clear**

Writing a one to this bit will clear RXSTPI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UNDERFIC: Underflow Interrupt Clear**

Writing a one to this bit will clear UNDERFI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXOUTIC: Received OUT Data Interrupt Clear**

Writing a one to this bit will clear RXOUTI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXINIC: Transmitted IN Data Interrupt Clear**

Writing a one to this bit will clear TXINI in UOTGHS_DEVEPTISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.2.13 Device Endpoint x Set Register

Name: UOTGHS_DEVEPTIFRx [x=0..9]

Address: 0x400AC190

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
–	–	–	NBUSYBKS	–	–	–	–
7	6	5	4	3	2	1	0
SHORT PACKETS	STALLEDIS/ CRCERRIS	OVERFIS	NAKINIS/ HBISOFLUSHIS	NAKOUTIS/ HBISOINERRIS	RXSTPIS/ UNDERFIS	RXOUTIS	TXINIS

- **NBUSYBKS: Number of Busy Banks Interrupt Set**

Writing a one to this bit will set NBUSYBK in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SHORTPACKETS: Short Packet Interrupt Set**

Writing a one to this bit will set SHORTPACKET in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **STALLEDIS: STALLed Interrupt Set**

Writing a one to this bit will set STALLEDI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **CRCERRIS: CRC Error Interrupt Set**

Writing a one to this bit will set CRCERRI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **OVERFIS: Overflow Interrupt Set**

Writing a one to this bit will set OVERFI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKINIS: NAKed IN Interrupt Set**

Writing a one to this bit will set NAKINI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HBISOFLUSHIS: High Bandwidth Isochronous IN Flush Interrupt Set**

Writing a one to this bit will set **HBISOFLUSHI** in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKOUTIS: NAKed OUT Interrupt Set**

Writing a one to this bit will set NAKOUTI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HBISOINERRIS: High bandwidth isochronous IN Underflow Error Interrupt Set**

Writing a one to this bit will set **HBISOINERRI** in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXSTPIS: Received SETUP Interrupt Set**

Writing a one to this bit will set RXSTPI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UNDERFIS: Underflow Interrupt Set**

Writing a one to this bit will set UNDERFI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXOUTIS: Received OUT Data Interrupt Set**

Writing a one to this bit will set RXOUTI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXINIS: Transmitted IN Data Interrupt Set**

Writing a one to this bit will set TXINI in UOTGHS_DEVEPTISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.2.14 Device Endpoint x Mask Register

Name: UOTGHS_DEVEPTIMRx [x=0..9]

Address: 0x400AC1C0

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	STALLRQ	RSTDT	NYETDIS	EPDISHDMA
15	14	13	12	11	10	9	8
–	FIFOCON	KILLBK	NBUSYBKE	–	ERRORTRANSE	DATAXE	MDATAE
7	6	5	4	3	2	1	0
SHORTPACKETE	STALLEDE/ CRCERRE	OVERFE	NAKINE/ HBISOFLUSHE	NAKOUTE/ HBISOINERRE	RXSTPE/ UNDERFE	RXOUTE	TXINE

- **STALLRQ: STALL Request**

This bit is set when UOTGHS_DEVEPTIERx.STALLRQS bit is written to one. This will request to send a STALL handshake to the host.

This bit is cleared when a new SETUP packet is received or when the UOTGHS_DEVEPTIDRx.STALLRQC bit is written to zero.

- **RSTDT: Reset Data Toggle**

This bit is set when UOTGHS_DEVEPTIERx.RSTDTS bit is written to one. This will clear the data toggle sequence, i.e., set to Data0 the data toggle sequence of the next sent (IN endpoints) or received (OUT endpoints) packet.

This bit is cleared instantaneously.

The user does not have to wait for this bit to be cleared.

- **NYETDIS: NYET Token Disable**

This bit is set when UOTGHS_DEVEPTIERx.NYETDISS bit is written to one. This will send a ACK handshake instead of a NYET handshake in high-speed mode.

This bit is cleared when the UOTGHS_DEVEPTIDRx.NYETDISC bit is written to one.This will let the UOTGHS handling the high-speed handshake following the USB 2.0 standard.

- **EPDISHDMA: Endpoint Interrupts Disable HDMA Request**

This bit is set when the UOTGHS_DEVEPTIERx.EPDISHDMAS is written to one. This will pause the on-going DMA channel x transfer on any Endpoint x interrupt (PEP_x), whatever the state of the Endpoint x Interrupt Enable bit (PEP_x).

The user then has to acknowledge or to disable the interrupt source (e.g. UOTGHS_DEVEPTISRx.RXOUTI) or to clear the EPDISHDMA bit (by writing a one to UOTGHS_DEVEPTIDRx.EPDISHDMAC bit) in order to complete the DMA transfer.

In ping-pong mode, if the interrupt is associated to a new system-bank packet (e.g. Bank1) and the current DMA transfer is running on the previous packet (Bank0), then the previous-packet DMA transfer completes normally, but the new-packet DMA transfer will not start (not requested).

If the interrupt is not associated to a new system-bank packet (UOTGHS_DEVEPTISRx.NAKINI, NAKOUTI, etc.), then the request cancellation may occur at any time and may immediately pause the current DMA transfer.

This may be used for example to identify erroneous packets, to prevent them from being transferred into a buffer, to complete a DMA transfer by software after reception of a short packet, etc.

- **FIFOCON: FIFO Control**

For control endpoints:

The FIFOCON and RWALL bits are irrelevant. The software shall therefore never use them on these endpoints. When read, their value is always 0.

For IN endpoints:

This bit is set when the current bank is free, at the same time as UOTGHS_DEVEPTISRx.TXINI.

This bit is cleared (by writing a one to UOTGHS_DEVEPTIDRx.FIFOCONC bit) to send the FIFO data and to switch to the next bank.

For OUT endpoints:

This bit is set when the current bank is full, at the same time as UOTGHS_DEVEPTISRx.RXOUTI.

This bit is cleared (by writing a one to UOTGHS_DEVEPTIDRx.FIFOCONC bit) to free the current bank and to switch to the next bank.

- **KILLBK: Kill IN Bank**

This bit is set when UOTGHS_DEVEPTIERx.KILLBKS bit is written to one. This will kill the last written bank.

This bit is cleared when the bank is killed.

Caution: The bank is really cleared when the “kill packet” procedure is accepted by the UOTGHS core. This bit is automatically cleared after the end of the procedure:

The bank is really killed: UOTGHS_DEVEPTISRx.NBUSYBK is decremented.

The bank is not cleared but sent (IN transfer): UOTGHS_DEVEPTISRx.NBUSYBK is decremented.

The bank is not cleared because it was empty.

The user shall wait for this bit to be cleared before trying to kill another packet.

This kill request is refused if at the same time an IN token is coming and the last bank is the current one being sent on the USB line. If at least 2 banks are ready to be sent, there is no problem to kill a packet even if an IN token is coming. Indeed, in this case, the current bank is sent (IN transfer) while the last bank is killed.

- **NBUSYBKE: Number of Busy Banks Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.NBUSYBKES bit is written to one. This will enable the Number of Busy Banks interrupt (UOTGHS_DEVEPTISRx.NBUSYBK).

This bit is cleared when UOTGHS_DEVEPTIDRx.NBUSYBKEC bit is written to zero. This will disable the Number of Busy Banks interrupt (UOTGHS_DEVEPTISRx.NBUSYBK).

- **ERRORTRANSE: Transaction Error Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.ERRORTRANSES bit is written to one. This will enable the transaction error interrupt (UOTGHS_DEVEPTISRx.ERRORTRANS).

This bit is cleared when UOTGHS_DEVEPTIDRx.ERRORTRANSEC bit is written to one. This will disable the transaction error interrupt (UOTGHS_DEVEPTISRx.ERRORTRANS).

- **DATAXE: DataX Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.DATAXES bit is written to one. This will enable the DATAX interrupt. (see DTSEQ bits)

This bit is cleared when UOTGHS_DEVEPTIDRx.DATAXEC bit is written to one. This will disable the DATAX interrupt.

- **MDATAE: MData Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.MDATAES bit is written to one. This will enable the Multiple DATA interrupt. (see DTSEQ bits)

This bit is cleared when UOTGHS_DEVEPTIDRx.MDATAEC bit is written to one. This will disable the Multiple DATA interrupt.

- **SHORTPACKETE: Short Packet Interrupt**

If this bit is set for non-control IN endpoints, a short packet transmission is guaranteed upon ending a DMA transfer, thus signaling an end of isochronous frame or a bulk or interrupt end of transfer, provided that he End of DMA Buffer Output Enable (END_B_EN) bit and the Automatic Switch (AUTOSW) bit are written to one.

This bit is set when UOTGHS_DEVEPTIERx.SHORTPACKETES bit is written to one. This will enable the Short Packet interrupt (UOTGHS_DEVEPTISRx.SHORTPACKET).

This bit is cleared when UOTGHS_DEVEPTIDRx.SHORTPACKETEC bit is written to one. This will disable the Short Packet interrupt (UOTGHS_DEVEPTISRx.SHORTPACKET).

- **STALLEDE: STALLed Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.STALLEDES bit is written to one. This will enable the STALLed interrupt (UOTGHS_DEVEPTISRx.STALLEDI).

This bit is cleared when UOTGHS_DEVEPTIDRx.STALLEDEC bit is written to one. This will disable the STALLed interrupt (UOTGHS_DEVEPTISRx.STALLEDI).

- **CRCERRE: CRC Error Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.CRCERRES bit is written to one. This will enable the CRC Error interrupt (UOTGHS_DEVEPTISRx.CRCERRI).

This bit is cleared when UOTGHS_DEVEPTIDRx.CRCERREC bit is written to one. This will disable the CRC Error interrupt (UOTGHS_DEVEPTISRx.CRCERRI).

- **OVERFE: Overflow Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.OVERFES bit is written to one. This will enable the Overflow interrupt (UOTGHS_DEVEPTISRx.OVERFI).

This bit is cleared when UOTGHS_DEVEPTIDRx.OVERFEC bit is written to one. This will disable the Overflow interrupt (UOTGHS_DEVEPTISRx.OVERFI).

- **NAKINE: NAKed IN Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.NAKINES bit is written to one. This will enable the NAKed IN interrupt (UOTGHS_DEVEPTISRx.NAKINI).

This bit is cleared when UOTGHS_DEVEPTIDRx.NAKINEC bit is written to one. This will disable the NAKed IN interrupt (UOTGHS_DEVEPTISRx.NAKINI).

- **HBISOFLUSHE: High Bandwidth Isochronous IN Flush Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.HBISOFLUSHES bit is written to one. This will enable the HBISOFLUSHI interrupt.

This bit is cleared when UOTGHS_DEVEPTIDRx.HBISOFLUSHEC bit disable the HBISOFLUSHI interrupt.

- **NAKOUTE: NAKed OUT Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.NAKOUTES bit is written to one. This will enable the NAKed OUT interrupt (UOTGHS_DEVEPTISRx.NAKOUTI).

This bit is cleared when UOTGHS_DEVEPTIDRx.NAKOUTEC bit is written to one. This will disable the NAKed OUT interrupt (UOTGHS_DEVEPTISRx.NAKOUTI).

- **HBISOINERRE: High Bandwidth Isochronous IN Error Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.HBISOINERRES bit is written to one. This will enable the HBISOINERRI interrupt.

This bit is cleared when UOTGHS_DEVEPTIDRx.HBISOINERREC bit disable the HBISOINERRI interrupt.

- **RXSTPE: Received SETUP Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.RXSTPES bit is written to one. This will enable the Received SETUP interrupt (UOTGHS_DEVEPTISRx.RXSTPI).

This bit is cleared when UOTGHS_DEVEPTIERx.RXSTPEC bit is written to one. This will disable the Received SETUP interrupt (UOTGHS_DEVEPTISRx.RXSTPI).

- **UNDERFE: Underflow Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.UNDERFES bit is written to one. This will enable the Underflow interrupt (UOTGHS_DEVEPTISRx.UNDERFI).

This bit is cleared when UOTGHS_DEVEPTIDRx.UNDERFEC bit is written to one. This will disable the Underflow interrupt (UOTGHS_DEVEPTISRx.UNDERFI).

- **RXOUTE: Received OUT Data Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.RXOUTES bit is written to one. This will enable the Received OUT Data interrupt (UOTGHS_DEVEPTISRx.RXOUTI).

This bit is cleared when the UOTGHS_DEVEPTIDRx.RXOUTEC bit is written to one. This will disable the Received OUT Data interrupt (UOTGHS_DEVEPTISRx.RXOUTI).

- **TXINE: Transmitted IN Data Interrupt**

This bit is set when UOTGHS_DEVEPTIERx.TXINES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_DEVEPTISRx.TXINI).

This bit is cleared when UOTGHS_DEVEPTIDRx.TXINEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_DEVEPTISRx.TXINI).

39.6.2.15 Device Endpoint x Disable Register

Name: UOTGHS_DEVEPTIDRx [x=0..9]

Address: 0x400AC220

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	STALLRQC	–	NYETDISC	EPDISHDMAC
15	14	13	12	11	10	9	8
–	FIFOCONC	–	NBUSYBKEC	–	ERRORTRANSEC	DATAXEC	MDATEC
7	6	5	4	3	2	1	0
SHORT PACKETEC	STALLEDEC/ CRCERREC	OVERFEC	NAKINEC/ HBISOFLUSHEC	NAKOUTEC/ HBISOINERREC	RXSTPEC/ UNDERFEC	RXOUTEC	TXINEC

- **STALLRQC: STALL Request Clear**

Writing a one to this bit will clear STALLRQ bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NYETDISC: NYET Token Disable Clear**

Writing a one to this bit will clear NYETDIS bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EPDISHDMAC: Endpoint Interrupts Disable HDMA Request Clear**

Writing a one to this bit will clear EPDISHDMA bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **FIFOCONC: FIFO Control Clear**

Writing a one to this bit will clear FIFOCON bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NBUSYBKEC: Number of Busy Banks Interrupt Clear**

Writing a one to this bit will clear NBUSYBKE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **ERRORTRANSEC: Transaction Error Interrupt Clear**

Writing a one to this bit will clear ERRORTRANSE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **DATAXEC: DataX Interrupt Clear**

Writing a one to this bit will clear DATAXE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **MDATEC: MData Interrupt Clear**

Writing a one to this bit will clear MDATE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SHORTPACKETEC: Shortpacket Interrupt Clear**

Writing a one to this bit will clear SHORTPACKETE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **STALLEDEC: STALLed Interrupt Clear**

Writing a one to this bit will clear STALLEDE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **CRCERREC: CRC Error Interrupt Clear**

Writing a one to this bit will clear CRCERRE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **OVERFEC: Overflow Interrupt Clear**

Writing a one to this bit will clear OVERFE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKINEC: NAKed IN Interrupt Clear**

Writing a one to this bit will clear NAKINE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HBISOFLUSHEC: High Bandwidth Isochronous IN Flush Interrupt Clear**

Writing a one to this bit will clear HBISOFLUSHE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKOUTEC: NAKed OUT Interrupt Clear**

Writing a one to this bit will clear NAKOUTE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HBISOINERREC: High Bandwidth Isochronous IN Error Interrupt Clear**

Writing a one to this bit will clear HBISOINERRE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXSTPEC: Received SETUP Interrupt Clear**

Writing a one to this bit will clear RXSTPE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UNDERFEC: Underflow Interrupt Clear**

Writing a one to this bit will clear UNDERFE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXOUTEC: Received OUT Data Interrupt Clear**

Writing a one to this bit will clear RXOUTE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXINEC: Transmitted IN Interrupt Clear**

Writing a one to this bit will clear TXINE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.2.16 Device Endpoint x Enable Register

Name: UOTGHS_DEVEPTIERx [x=0..9]

Address: 0x400AC1F0

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	STALLRQS	RSTDTS	NYETDISS	EPDISHDMAS
15	14	13	12	11	10	9	8
–	–	KILLBKS	NBUSYBKES	–	ERRORTRANSES	DATAXES	MDATAES
7	6	5	4	3	2	1	0
SHORTPACKETES	STALLEDES/CRCERRES	OVERFES	NAKINES/HBISOFLUSHES	NAKOUTES/HBISOINERRES	RXSTPES/UNDERFES	RXOUTES	TXINES

- **STALLRQS: STALL Request Enable**

Writing a one to this bit will set STALLRQ bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RSTDTS: Reset Data Toggle Enable**

Writing a one to this bit will set RSTDT bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NYETDISS: NYET Token Disable Enable**

Writing a one to this bit will set NYETDIS bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **EPDISHDMAS: Endpoint Interrupts Disable HDMA Request Enable**

Writing a one to this bit will set EPDISHDMA bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **FIFOCONS: FIFO Control**

Writing a one to this bit will set FIFOCON bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **KILLBKS: Kill IN Bank**

Writing a one to this bit will set KILLBK bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NBUSYBKES: Number of Busy Banks Interrupt Enable**

Writing a one to this bit will set NBUSYBKE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **ERRORTRANSES: Transaction Error Interrupt Enable**

Writing a one to this bit will set ERRORTRANSE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **DATAXES: DataX Interrupt Enable**

Writing a one to this bit will set DATAXE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **MDATAES: MData Interrupt Enable**

Writing a one to this bit will set MDATAE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SHORTPACKETES: Short Packet Interrupt Enable**

Writing a one to this bit will set SHORTPACKETE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **STALLEDES: STALLed Interrupt Enable**

Writing a one to this bit will set STALLEDE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **CRCERRES: CRC Error Interrupt Enable**

Writing a one to this bit will set CRCERRE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **OVERFES: Overflow Interrupt Enable**

Writing a one to this bit will set OVERFE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKINES: NAKed IN Interrupt Enable**

Writing a one to this bit will set NAKINE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HBISOFLUSHES: High Bandwidth Isochronous IN Flush Interrupt Enable**

Writing a one to this bit will set HBISOFLUSHE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKOUTES: NAKed OUT Interrupt Enable**

Writing a one to this bit will set NAKOUTE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HBISOINERRES: High Bandwidth Isochronous IN Error Interrupt Enable**

Writing a one to this bit will set HBISOINERRE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXSTPES: Received SETUP Interrupt Enable**

Writing a one to this bit will set RXSTPE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UNDERFES: Underflow Interrupt Enable**

Writing a one to this bit will set UNDERFE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXOUTES: Received OUT Data Interrupt Enable**

Writing a one to this bit will set RXOUTE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXINES: Transmitted IN Data Interrupt Enable**

Writing a one to this bit will set TXINE bit in UOTGHS_DEVEPTIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.2.17 Device DMA Channel x Next Descriptor Address Register

Name: UOTGHS_DEVDMANXTDSCx [x=1..6]

Address: 0x400AC310 [1], 0x400AC320 [2], 0x400AC330 [3], 0x400AC340 [4], 0x400AC350 [5], 0x400AC360 [6], 0x400AC370 [7]

Access: Read-write

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

- **NXT_DSC_ADD: Next Descriptor Address**

This field points to the next channel descriptor to be processed. This channel descriptor must be aligned, so bits 0 to 3 of the address must be equal to zero.

39.6.2.18 Device DMA Channel x Address Register

Name: UOTGHS_DEVDMAADDRESSx [x=1..6]

Address: 0x400AC314 [1], 0x400AC324 [2], 0x400AC334 [3], 0x400AC344 [4], 0x400AC354 [5], 0x400AC364 [6], 0x400AC374 [7]

Access: Read-write

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

- **BUFF_ADD: Buffer Address**

This field determines the AHB bus starting address of a DMA channel transfer.

Channel start and end addresses may be aligned on any byte boundary.

The firmware may write this field only when the UOTGHS_DEVDMASTATUS.CHANN_ENB bit is clear.

This field is updated at the end of the address phase of the current access to the AHB bus. It is incrementing of the access byte width. The access width is 4 bytes (or less) at packet start or end, if the start or end address is not aligned on a word boundary.

The packet start address is either the channel start address or the next channel address to be accessed in the channel buffer.

The packet end address is either the channel end address or the latest channel address accessed in the channel buffer.

The channel start address is written by software or loaded from the descriptor, whereas the channel end address is either determined by the end of buffer or the USB device, USB end of transfer if the UOTGHS_DEVDMACONTROLx.END_TR_EN bit is set.

39.6.2.19 Device DMA Channel x Control Register

Name: UOTGHS_DEVDMACONTROLx [x=1..6]

Address: 0x400AC318 [1], 0x400AC328 [2], 0x400AC338 [3], 0x400AC348 [4], 0x400AC358 [5], 0x400AC368 [6], 0x400AC378 [7]

Access: Read-write

31	30	29	28	27	26	25	24
BUFF_LENGTH							
23	22	21	20	19	18	17	16
BUFF_LENGTH							
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
BURST_LCK	DESC_LD_IT	END_BUFFIT	END_TR_IT	END_B_EN	END_TR_EN	LDNXT_DSC	CHANN_ENB

- **CHANN_ENB: Channel Enable Command**

0: DMA channel is disabled at and no transfer will occur upon request. This bit is also cleared by hardware when the channel source bus is disabled at end of buffer.

If LDNXT_DSC bit has been cleared by descriptor loading, the firmware will have to set the corresponding CHANN_ENB bit to start the described transfer, if needed.

If LDNXT_DSC bit is cleared, the channel is frozen and the channel registers may then be read and/or written reliably as soon as both UOTGHS_DEVDMASTATUS.CHANN_ENB and CHANN_ACT flags read as 0.

If a channel request is currently serviced when this bit is cleared, the DMA FIFO buffer is drained until it is empty, then the UOTGHS_DEVDMASTATUS.CHANN_ENB bit is cleared.

If LDNXT_DSC bit is set at or after this bit clearing, then the currently loaded descriptor is skipped (no data transfer occurs) and the next descriptor is immediately loaded.

1: UOTGHS_DEVDMASTATUS.CHANN_ENB bit will be set, thus enabling DMA channel data transfer. Then any pending request will start the transfer. This may be used to start or resume any requested transfer.

- **LDNXT_DSC: Load Next Channel Transfer Descriptor Enable Command**

0: no channel register is loaded after the end of the channel transfer.

1: the channel controller loads the next descriptor after the end of the current transfer, i.e. when UOTGHS_DEVDMASTATUS.CHANN_ENB bit is reset.

If CHANN_ENB bit is cleared, the next descriptor is immediately loaded upon transfer request.

DMA Channel Control Command Summary

Value	Name	Description
0	STOP_NOW	Stop now
1	RUN_AND_STOP	Run and stop at end of buffer
2	LOAD_NEXT_DESC	Load next descriptor now
3	RUN_AND_LINK	Run and link at end of buffer

- **END_TR_EN: End of Transfer Enable Control**

Used for OUT transfers only.

0: USB end of transfer is ignored.

1: UOTGHS device can put an end to the current buffer transfer.

When set, a BULK or INTERRUPT short packet or the last packet of an ISOCHRONOUS (micro) frame (DATAX) will close the current buffer and the UOTGHS_DEVDMASTATUSx.END_TR_ST flag will be raised.

This is intended for UOTGHS non-prenegotiated end of transfer (BULK or INTERRUPT) or ISOCHRONOUS microframe data buffer closure.

- **END_B_EN: End of Buffer Enable Control**

0: DMA Buffer End has no impact on USB packet transfer.

1: the endpoint can validate the packet (according to the values programmed in the UOTGHS_DEVEPTCFGx.AUTOSW field, and in the UOTGHS_DEVEPTIERx.SHORTPACKETES field) at DMA Buffer End, i.e. when the UOTGHS_DEVDMASTATUS.BUFF_COUNT reaches 0.

This is mainly for short packet IN validation initiated by the DMA reaching end of buffer, but could be used for OUT packet truncation (discarding of unwanted packet data) at the end of DMA buffer.

- **END_TR_IT: End of Transfer Interrupt Enable**

0: UOTGHS device initiated buffer transfer completion will not trigger any interrupt at UOTGHS_DEVDMASTATUSx.END_TR_ST rising.

1: an interrupt is sent after the buffer transfer is complete, if the UOTGHS device has ended the buffer transfer.

Use when the receive size is unknown.

- **END_BUFFIT: End of Buffer Interrupt Enable**

0: UOTGHS_DEVDMA_STATUSx.END_BF_ST rising will not trigger any interrupt.

1: an interrupt is generated when the UOTGHS_HSTDMASTATUSx.BUFF_COUNT reaches zero.

- **DESC_LD_IT: Descriptor Loaded Interrupt Enable**

0: UOTGHS_DEVDMASTATUSx.DESC_LDST rising will not trigger any interrupt.

1: an interrupt is generated when a descriptor has been loaded from the bus.

- **BURST_LCK: Burst Lock Enable**

0: the DMA never locks bus access.

1: USB packets AHB data bursts are locked for maximum optimization of the bus bandwidth usage and maximization of fly-by AHB burst duration.

- **BUFF_LENGTH: Buffer Byte Length (Write-only)**

This field determines the number of bytes to be transferred until end of buffer. The maximum channel transfer size (32 KBytes) is reached when this field is 0 (default value). If the transfer size is unknown, this field should be set to 0, but the transfer end may occur earlier under USB device control.

When this field is written, The UOTGHS_DEVDMASTATUSx.BUFF_COUNT field is updated with the write value.

Notes:
1. Bits [31:2] are only writable when issuing a channel Control Command other than “Stop Now”.
2. For reliability it is highly recommended to wait for both UOTGHS_DEVDMASTATUSx.CHAN_ACT and UOTGHS_DEVDMASTATUSx.CHAN_ENB flags are at 0, thus ensuring the channel has been stopped before issuing a command other than “Stop Now”.

39.6.2.20 Device DMA Channel x Status Register

Name: UOTGHS_DEVDMASTATUSx [x=1..6]

Address: 0x400AC31C [1], 0x400AC32C [2], 0x400AC33C [3], 0x400AC34C [4], 0x400AC35C [5], 0x400AC36C [6], 0x400AC37C [7]

Access: Read-write

31	30	29	28	27	26	25	24
BUFF_COUNT							
23	22	21	20	19	18	17	16
BUFF_COUNT							
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	DESC_LDST	END_BF_ST	END_TR_ST	–	–	CHANN_ACT	CHANN_ENB

- **CHANN_ENB: Channel Enable Status**

0: if cleared, the DMA channel no longer transfers data, and may load the next descriptor if the UOTGHS_DEVDMACONTROLx.LDNXT_DSC bit is set.

When any transfer is ended either due to an elapsed byte count or a UOTGHS device initiated transfer end, this bit is automatically reset.

1: if set, the DMA channel is currently enabled and transfers data upon request.

This bit is normally set or cleared by writing into the UOTGHS_DEVDMACONTROLx.CHANN_ENB bit field either by software or descriptor loading.

If a channel request is currently serviced when the UOTGHS_DEVDMACONTROLx.CHANN_ENB bit is cleared, the DMA FIFO buffer is drained until it is empty, then this status bit is cleared.

- **CHANN_ACT: Channel Active Status**

0: the DMA channel is no longer trying to source the packet data.

When a packet transfer is ended this bit is automatically reset.

1: the DMA channel is currently trying to source packet data, i.e. selected as the highest-priority requesting channel.

When a packet transfer cannot be completed due to an END_BF_ST, this flag stays set during the next channel descriptor load (if any) and potentially until UOTGHS packet transfer completion, if allowed by the new descriptor.

- **END_TR_ST: End of Channel Transfer Status**

0: cleared automatically when read by software.

1: set by hardware when the last packet transfer is complete, if the UOTGHS device has ended the transfer.

Valid until the CHANN_ENB flag is cleared at the end of the next buffer transfer.

- **END_BF_ST: End of Channel Buffer Status**

0: cleared automatically when read by software.

1: set by hardware when the BUFF_COUNT count-down reaches zero.

Valid until the CHANN_ENB flag is cleared at the end of the next buffer transfer.

- **DESC_LDST: Descriptor Loaded Status**

0: cleared automatically when read by software.

1: set by hardware when a descriptor has been loaded from the system bus.

Valid until the CHANN_ENB flag is cleared at the end of the next buffer transfer.

- **BUFF_COUNT: Buffer Byte Count**

This field determines the current number of bytes still to be transferred for this buffer.

This field is decremented from the AHB source bus access byte width at the end of this bus address phase.

The access byte width is 4 by default, or less, at DMA start or end, if the start or end address is not aligned on a word boundary.

At the end of buffer, the DMA accesses the UOTGHS device only for the number of bytes needed to complete it.

Note: For OUT endpoints, if the receive buffer byte length (BUFF_LENGTH) has been defaulted to zero because the USB transfer length is unknown, the actual buffer byte length received will be 0x10000-BUFF_COUNT.

39.6.3 USB Host Registers

39.6.3.1 Host General Control Register

Name: UOTGHS_HSTCTRL

Address: 0x400AC400

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
–	–	SPDCONF		–	RESUME	RESET	SOFE
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **SPDCONF: Mode Configuration**

This field contains the host speed capability:.

Value	Name	Description
0	NORMAL	The host starts in full-speed mode and performs a high-speed reset to switch to the high-speed mode if the downstream peripheral is high-speed capable.
1	LOW_POWER	For a better consumption, if high-speed is not needed.
2	HIGH_SPEED	Forced high speed.
3	FORCED_FS	The host remains to full-speed mode whatever the peripheral speed capability.

- **RESUME: Send USB Resume**

Writing a one to this bit will generate a USB Resume on the USB bus.

This bit is cleared when the USB Resume has been sent or when a USB reset is requested.

Writing a zero to this bit has no effect.

This bit should be written to one only when the start of frame generation is enable. (SOFE bit is one).

- **RESET: Send USB Reset**

Writing a one to this bit will generate a USB Reset on the USB bus.

This bit is cleared when the USB Reset has been sent.

It may be useful to write a zero to this bit when a device disconnection is detected (UOTGHS_HSTISR.DDISCI is one) whereas a USB Reset is being sent.

- **SOFE: Start of Frame Generation Enable**

Writing a one to this bit will generate SOF on the USB bus in full or high speed mode and keep alive in low speed mode.

Writing a zero to this bit will disable the SOF generation and to leave the USB bus in idle state.

This bit is set when a USB reset is requested or an upstream resume interrupt is detected (UOTGHS_HSTISR.TXRSMI).

39.6.3.2 Host Global Interrupt Status Register

Name: UOTGHS_HSTISR

Address: 0x400AC404

Access: Read-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	PEP_9	PEP_8
15	14	13	12	11	10	9	8
PEP_7	PEP_6	PEP_5	PEP_4	PEP_3	PEP_2	PEP_1	PEP_0
7	6	5	4	3	2	1	0
–	HWUPI	HSOFI	RXRSMI	RSMEDI	RSTI	DDISCI	DCONNI

- **DMA_x: DMA Channel x Interrupt**

This bit is set when an interrupt is triggered by the DMA channel x. This triggers a USB interrupt if the corresponding bit in UOTGHS_HSTIMR is one.

This bit is cleared when the UOTGHS_HSTDMASTATUSx interrupt source is cleared.

- **PEP_x: Pipe x Interrupt**

This bit is set when an interrupt is triggered by the endpoint x (UOTGHS_HSTPIPISRx). This triggers a USB interrupt if the corresponding bit in UOTGHS_HSTIMR is one.

This bit is cleared when the interrupt source is served.

- **HWUPI: Host Wake-Up Interrupt**

This bit is set when the host controller is in the suspend mode (SOFE is zero) and an upstream resume from the peripheral is detected.

This bit is set when the host controller is in the suspend mode (SOFE is zero) and a peripheral disconnection is detected.

This bit is set when the host controller is in the Idle state (UOTGHS_SR.VBUSRQ is zero, no VBus is generated), and an OTG SRP event initiated by the peripheral is detected (UOTGHS_SR.SRPI is one).

This interrupt is generated even if the clock is frozen by UOTGHS_CTRL.FRZCLK bit.

- **HSOFI: Host Start of Frame Interrupt**

This bit is set when a SOF is issued by the Host controller. This triggers a USB interrupt when HSOFE is one. When using the host controller in low speed mode, this bit is also set when a keep-alive is sent.

This bit is cleared when UOTGHS_HSTICR.HSOFIC bit is written to one.

- **RXRSMI: Upstream Resume Received Interrupt**

This bit is set when an Upstream Resume has been received from the Device.

This bit is cleared when UOTGHS_HSTICR.RXRSMIC is written to one.

- **RSMEDI: Downstream Resume Sent Interrupt**

This bit set when a Downstream Resume has been sent to the Device.

This bit is cleared when UOTGHS_HSTICR.RSMEDIC bit is written to one.

- **RSTI: USB Reset Sent Interrupt**

This bit is set when a USB Reset has been sent to the device.

This bit is cleared when UOTGHS_HSTICR.RSTIC bit is written to one.

- **DDISCI: Device Disconnection Interrupt**

This bit is set when the device has been removed from the USB bus.

This bit is cleared when UOTGHS_HSTICR.DDISCIC bit is written to one.

- **DCONNI: Device Connection Interrupt**

This bit is set when a new device has been connected to the USB bus.

This bit is cleared when UOTGHS_HSTICR.DCONNIC bit is written to one.

39.6.3.3 Host Global Interrupt Clear Register

Name: UOTGHS_HSTICR

Address: 0x400AC408

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
–	HWUPIC	HSOFIC	RXRSMIC	RSMEDIC	RSTIC	DDISCIC	DCONNIC

- **HWUPIC: Host Wake-Up Interrupt Clear**

Writing a one to this bit will clear HWUPI bit in UOTGHS_HSTISR.

Writing a zero to a bit in this register has no effect.

This bit always reads as zero.

- **HSOFIC: Host Start of Frame Interrupt Clear**

Writing a one to this bit will clear HSOFI bit in UOTGHS_HSTISR.

Writing a zero to a bit in this register has no effect.

This bit always reads as zero.

- **RXRSMIC: Upstream Resume Received Interrupt Clear**

Writing a one to this bit will clear RXRSMI bit in UOTGHS_HSTISR.

Writing a zero to a bit in this register has no effect.

This bit always reads as zero.

- **RSMEDIC: Downstream Resume Sent Interrupt Clear**

Writing a one to this bit will clear RSMEDI bit in UOTGHS_HSTISR.

Writing a zero to a bit in this register has no effect.

This bit always reads as zero.

- **RSTIC: USB Reset Sent Interrupt Clear**

Writing a one to this bit will clear RSTI bit in UOTGHS_HSTISR.

Writing a zero to a bit in this register has no effect.

This bit always reads as zero.

- **DDISCIC: Device Disconnection Interrupt Clear**

Writing a one to this bit will clear DDISCI bit in UOTGHS_HSTISR.

Writing a zero to a bit in this register has no effect.

This bit always reads as zero.

- **DCONNIC: Device Connection Interrupt Clear**

Writing a one to this bit will clear DCONNI bit in UOTGHS_HSTISR.

Writing a zero to a bit in this register has no effect.

This bit always reads as zero.

39.6.3.4 Host Global Interrupt Set Register

Name: UOTGHS_HSTIFR

Address: 0x400AC40C

Access: Write-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	HWUPIS	HSOFIS	RXRSMIS	RSMEDIS	RSTIS	DDISCIS	DCONNIS

- **DMA_x: DMA Channel x Interrupt Set**

Writing a one to this bit will set the corresponding bit in UOTGHS_HSTISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HWUPIS: Host Wake-Up Interrupt Set**

Writing a one to this bit will set HWUPI bit in UOTGHS_HSTISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HSOFIS: Host Start of Frame Interrupt Set**

Writing a one to this bit will set HSOFI bit in UOTGHS_HSTISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXRSMIS: Upstream Resume Received Interrupt Set**

Writing a one to this bit will set RXRSMI bit in UOTGHS_HSTISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RSMEDIS: Downstream Resume Sent Interrupt Set**

Writing a one to this bit will set RSMEDI bit in UOTGHS_HSTISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RSTIS: USB Reset Sent Interrupt Set**

Writing a one to this bit will set RSTI bit in UOTGHS_HSTISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **DDISCIS: Device Disconnection Interrupt Set**

Writing a one to this bit will set DDISCI bit in UOTGHS_HSTISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **DCONNIS: Device Connection Interrupt Set**

Writing a one to this bit will set DCONNI bit in UOTGHS_HSTISR, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.3.5 Host Global Interrupt Mask Register

Name: UOTGHS_HSTIMR

Address: 0x400AC410

Access: Read-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	PEP_9	PEP_8
15	14	13	12	11	10	9	8
PEP_7	PEP_6	PEP_5	PEP_4	PEP_3	PEP_2	PEP_1	PEP_0
7	6	5	4	3	2	1	0
–	HWUPIE	HSOFIE	RXRSMIE	RSMEDIE	RSTIE	DDISCIE	DCONNIE

- **DMA_x: DMA Channel x Interrupt Enable**

This bit is set when the corresponding bit in UOTGHS_HSTIER is written to one. This will enable the DMA Channel x Interrupt (UOTGHS_HSTISR.DMA_x).

This bit is cleared when the corresponding bit in UOTGHS_HSTIDR is written to one. This will disable the DMA Channel x Interrupt (UOTGHS_HSTISR.DMA_x).

- **PEP_x: Pipe x Interrupt Enable**

This bit is set when the corresponding bit in UOTGHS_HSTIER is written to one. This will enable the Pipe x Interrupt (UOTGHS_HSTISR.PEP_x).

This bit is cleared when the PEP_x bit is written to one. This will disable the Pipe x Interrupt (PEP_x).

- **HWUPIE: Host Wake-Up Interrupt Enable**

This bit is set when UOTGHS_HSTIER.HWUPIES bit is written to one. This will enable the Host Wake-up Interrupt (UOTGHS_HSTISR.HWUPI).

This bit is cleared when UOTGHS_HSTIDR.HWUPIEC bit is written to one. This will disable the Host Wake-up Interrupt (UOTGHS_HSTISR.HWUPI).

- **HSOFIE: Host Start of Frame Interrupt Enable**

This bit is set when UOTGHS_HSTIER.HSOFIES bit is written to one. This will enable the Host Start of Frame interrupt (UOTGHS_HSTISR.HSOFI).

This bit is cleared when UOTGHS_HSTIDR.HSOFIEC bit is written to one. This will disable the Host Start of Frame interrupt (UOTGHS_HSTISR.HSOFI).

- **RXRSMIE: Upstream Resume Received Interrupt Enable**

This bit is set when UOTGHS_HSTIER.RXRSMIES bit is written to one. This will enable the Upstream Resume Received interrupt (UOTGHS_HSTISR.RXRSMI).

This bit is cleared when UOTGHS_HSTIDR.RXRSMIEC bit is written to one. This will disable the Downstream Resume interrupt (UOTGHS_HSTISR.RXRSMI).

- **RSMEDIE: Downstream Resume Sent Interrupt Enable**

This bit is set when UOTGHS_HSTIER.RSMEDIES bit is written to one. This will enable the Downstream Resume interrupt (UOTGHS_HSTISR.RSMEDI).

This bit is cleared when UOTGHS_HSTIDR.RSMEDIEC bit is written to one. This will disable the Downstream Resume interrupt (UOTGHS_HSTISR.RSMEDI).

- **RSTIE: USB Reset Sent Interrupt Enable**

This bit is set when UOTGHS_HSTIER.RSTIES bit is written to one. This will enable the USB Reset Sent interrupt (UOTGHS_HSTISR.RSTI).

This bit is cleared when UOTGHS_HSTIDR.RSTIEC bit is written to one. This will disable the USB Reset Sent interrupt (UOTGHS_HSTISR.RSTI).

- **DDISCIE: Device Disconnection Interrupt Enable**

This bit is set when UOTGHS_HSTIER.DDISCIES bit is written to one. This will enable the Device Disconnection interrupt (UOTGHS_HSTISR.DDISCI).

This bit is cleared when UOTGHS_HSTIDR.DDISCIEC bit is written to one. This will disable the Device Disconnection interrupt (UOTGHS_HSTISR.DDISCI).

- **DCONNIE: Device Connection Interrupt Enable**

This bit is set when UOTGHS_HSTIER.DCONNIES bit is written to one. This will enable the Device Connection interrupt (UOTGHS_HSTISR.DCONNI).

This bit is cleared when UOTGHS_HSTIDR.DCONNIEC bit is written to one. This will disable the Device Connection interrupt (UOTGHS_HSTISR.DCONNI).

39.6.3.6 Host Global Interrupt Disable Register

Name: UOTGHS_HSTIDR

Address: 0x400AC414

Access: Write-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	PEP_9	PEP_8
15	14	13	12	11	10	9	8
PEP_7	PEP_6	PEP_5	PEP_4	PEP_3	PEP_2	PEP_1	PEP_0
7	6	5	4	3	2	1	0
–	HWUPIEC	HSOFIEC	RXRSMIEC	RSMEDIEC	RSTIEC	DDISCIEC	DCONNIEC

- **DMA_x: DMA Channel x Interrupt Disable**

Writing a one to this bit will clear the corresponding bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PEP_x: Pipe x Interrupt Disable**

Writing a one to this bit will clear the corresponding bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HWUPIEC: Host Wake-Up Interrupt Disable**

Writing a one to this bit will clear HWUPIE bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HSOFIEC: Host Start of Frame Interrupt Disable**

Writing a one to this bit will clear HSOFIE bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXRSMIEC: Upstream Resume Received Interrupt Disable**

Writing a one to this bit will clear RXRSMIE bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RSMEDIEC: Downstream Resume Sent Interrupt Disable**

Writing a one to this bit will clear RSMEDIE bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RSTIEC: USB Reset Sent Interrupt Disable**

Writing a one to this bit will clear RSTIE bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **DDISCIEC: Device Disconnection Interrupt Disable**

Writing a one to this bit will clear DDISCIE bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **DCONNIEC: Device Connection Interrupt Disable**

Writing a one to this bit will clear DCONNIE bit in UOTGHS_HSTIMR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.3.7 Host Global Interrupt Enable Register

Name: UOTGHS_HSTIER

Address: 0x400AC418

Access: Write-only

31	30	29	28	27	26	25	24
–	DMA_6	DMA_5	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	PEP_9	PEP_8
15	14	13	12	11	10	9	8
PEP_7	PEP_6	PEP_5	PEP_4	PEP_3	PEP_2	PEP_1	PEP_0
7	6	5	4	3	2	1	0
–	HWUPIES	HSOFIES	RXRSMIES	RSMEDIES	RSTIES	DDISCIES	DCONNIES

- **DMA_x: DMA Channel x Interrupt Enable**

Writing a one to this bit will set the corresponding bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PEP_x: Pipe x Interrupt Enable**

Writing a one to this bit will set the corresponding bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HWUPIES: Host Wake-Up Interrupt Enable**

Writing a one to this bit will set HWUPI bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **HSOFIES: Host Start of Frame Interrupt Enable**

Writing a one to this bit will set HSOFI bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXRSMIES: Upstream Resume Received Interrupt Enable**

Writing a one to this bit will set RXRSMI bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RSMEDIES: Downstream Resume Sent Interrupt Enable**

Writing a one to this bit will set RSMEDI bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RSTIES: USB Reset Sent Interrupt Enable**

Writing a one to this bit will set RSTI bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **DDISCIES: Device Disconnection Interrupt Enable**

Writing a one to this bit will set DDISCI bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **DCONNIES: Device Connection Interrupt Enable**

Writing a one to this bit will set DCONNI bit in UOTGHS_HSTISR.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.3.8 Host Frame Number Register

Name: UOTGHS_HSTFNUM

Address: 0x400AC420

Access: Read-write

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

- **FLENHIGH: Frame Length**

In High speed mode, this field contains the 8 high-order bits of the 16-bit internal frame counter (at 30 MHz, counter length is 3750 to ensure a SOF generation every 125 us).

- **FNUM: Frame Number**

This field contains the current SOF number.

This field can be written. In this case, the MFNUM field is reset to zero.

- **MFNUM: Micro Frame Number**

This field contains the current Micro Frame number (can vary from 0 to 7) updated every 125 us.

When operating in full-speed mode, this field is tied to zero.

39.6.3.9 Host Address 1 Register

Name: UOTGHS_HSTADDR1

Address: 0x400AC424

Access: Read-write

31	30	29	28	27	26	25	24
–	HSTADDRP3						
23	22	21	20	19	18	17	16
–	HSTADDRP2						
15	14	13	12	11	10	9	8
–	HSTADDRP1						
7	6	5	4	3	2	1	0
–	HSTADDRP0						

- **HSTADDRP3: USB Host Address**

This field contains the address of the Pipe3 of the USB Device.

This field is cleared when a USB reset is requested.

- **HSTADDRP2: USB Host Address**

This field contains the address of the Pipe2 of the USB Device.

This field is cleared when a USB reset is requested.

- **HSTADDRP1: USB Host Address**

This field contains the address of the Pipe1 of the USB Device.

This field is cleared when a USB reset is requested.

- **HSTADDRP0: USB Host Address**

This field contains the address of the Pipe0 of the USB Device.

This field is cleared when a USB reset is requested.

39.6.3.10 Host Address 2 Register

Name: UOTGHS_HSTADDR2

Address: 0x400AC428

Access: Read-write

31	30	29	28	27	26	25	24
–	HSTADDRP7						
23	22	21	20	19	18	17	16
–	HSTADDRP6						
15	14	13	12	11	10	9	8
–	HSTADDRP5						
7	6	5	4	3	2	1	0
–	HSTADDRP4						

- **HSTADDRP7: USB Host Address**

This field contains the address of the Pipe7 of the USB Device.

This field is cleared when a USB reset is requested.

- **HSTADDRP6: USB Host Address**

This field contains the address of the Pipe6 of the USB Device.

This field is cleared when a USB reset is requested.

- **HSTADDRP5: USB Host Address**

This field contains the address of the Pipe5 of the USB Device.

This field is cleared when a USB reset is requested.

- **HSTADDRP4: USB Host Address**

This field contains the address of the Pipe4 of the USB Device.

This field is cleared when a USB reset is requested.

39.6.3.11 Host Address 3 Register

Name: UOTGHS_HSTADDR3

Address: 0x400AC42C

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
–	HSTADDRP9						
7	6	5	4	3	2	1	0
–	HSTADDRP8						

- **HSTADDRP9: USB Host Address**

This field contains the address of the Pipe9 of the USB Device.

This field is cleared when a USB reset is requested.

- **HSTADDRP8: USB Host Address**

This field contains the address of the Pipe8 of the USB Device.

This field is cleared when a USB reset is requested.

39.6.3.12 Host Pipe Register

Name: UOTGHS_HSTPIP

Address: 0x400AC41C

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	PRST8
23	22	21	20	19	18	17	16
PRST7	PRST6	PRST5	PRST4	PRST3	PRST2	PRST1	PRST0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	PEN8
7	6	5	4	3	2	1	0
PEN7	PEN6	PEN5	PEN4	PEN3	PEN2	PEN1	PEN0

- **PRSTx: Pipe x Reset**

Writing a one to this bit will reset the Pipe x FIFO.

This resets the endpoint x registers (UOTGHS_HSTPIPCFGx, UOTGHS_HSTPIPISRx, UOTGHS_HSTPIPIMRx) but not the endpoint configuration (ALLOC, PBK, PSIZE, PTOKEN, PTYPE, PEPNUM, INTFRQ).

All the endpoint mechanism (FIFO counter, reception, transmission, etc.) is reset apart from the Data Toggle management.

The endpoint configuration remains active and the endpoint is still enabled.

Writing a zero to this bit will complete the reset operation and allow to start using the FIFO.

- **PENx: Pipe x Enable**

Writing a one to this bit will enable the Pipe x.

Writing a zero to this bit will disable the Pipe x, which forces the Pipe x state to inactive and resets the pipe x registers (UOTGHS_HSTPIPCFGx, UOTGHS_HSTPIPISRx, UOTGHS_HSTPIPIMRx) but not the pipe configuration (UOTGHS_HSTPIPCFGx.ALLOC, UOTGHS_HSTPIPCFGx.PBK, UOTGHS_HSTPIPCFGx.PSIZE).

39.6.3.13 Host Pipe x Configuration Register

Name: UOTGHS_HSTPIPCFGx [x=0..9]

Address: 0x400AC500

Access: Read-write

31	30	29	28	27	26	25	24
INTFRQ/BINTERVAL							
23	22	21	20	19	18	17	16
–	–	–	PINGEN	PEPNUM			
15	14	13	12	11	10	9	8
–	–	PTYPE		–	AUTOSW	PTOKEN	
7	6	5	4	3	2	1	0
–	PSIZE			PBK		ALLOC	–

- **INTFRQ: Pipe Interrupt Request Frequency**

This field contains the maximum value in millisecond of the polling period for an Interrupt Pipe.

This value has no effect for a non-Interrupt Pipe.

This field is cleared upon sending a USB reset.

- **BINTERVAL: bInterval parameter for the Bulk-Out/Ping transaction**

This field contains the Ping/Bulk-out period.

- If BINTERVAL>0 and PINGEN=1, one PING token is sent every BINTERVAL micro-frame until it is ACKed by the peripheral.
- If BINTERVAL=0 and PINGEN=1, multiple consecutive PING token is sent in the same micro-frame until it is ACKed.
- If BINTERVAL>0 and PINGEN=0, one OUT token is sent every BINTERVAL micro-frame until it is ACKed by the peripheral.
- If BINTERVAL=0 and PINGEN=0, multiple consecutive OUT token is sent in the same micro-frame until it is ACKed.

This value must be in the range from 0 to 255.

- **PINGEN: Ping Enable**

This bit is relevant for High-speed Bulk-out transaction only (including the control data stage and the control status stage).

Writing a zero to this bit will disable the ping protocol.

Writing a one to this bit will enable the ping mechanism according to the USB 2.0 standard.

This bit is cleared upon sending a USB reset.

- **PEPNUM: Pipe Endpoint Number**

This field contains the number of the endpoint targeted by the pipe. This value is from 0 to 10.

This field is cleared upon sending a USB reset.

- **PTYPE: Pipe Type**

This field contains the pipe type.

Value	Name	Description
0	CTRL	Control
1	ISO	Isochronous
2	BLK	Bulk
3	INTRPT	Interrupt

This field is cleared upon sending a USB reset.

- **AUTOSW: Automatic Switch**

This bit is cleared upon sending a USB reset.

0: The automatic bank switching is disabled.

1: The automatic bank switching is enabled.

- **PTOKEN: Pipe Token**

This field contains the endpoint token.

Value	Name	Description
0	SETUP	SETUP
1	IN	IN
2	OUT	OUT
3		reserved

- **PSIZE: Pipe Size**

This field contains the size of each pipe bank.

Value	Name	Description
0	8_BYTE	8 bytes
1	16_BYTE	16 bytes
2	32_BYTE	32 bytes
3	64_BYTE	64 bytes
4	128_BYTE	128 bytes
5	256_BYTE	256 bytes
6	512_BYTE	512 bytes
7	1024_BYTE	1024 bytes

This field is cleared upon sending a USB reset.

- **PBK: Pipe Banks**

This field contains the number of banks for the pipe.

Value	Name	Description
0	1_BANK	Single-bank pipe
1	2_BANK	Double-bank pipe
2	3_BANK	Triple-bank pipe
3		Reserved

For control endpoints, a single-bank pipe (0b00) should be selected.

This field is cleared upon sending a USB reset.

- **ALLOC: Pipe Memory Allocate**

Writing a one to this bit will allocate the pipe memory.

Writing a zero to this bit will free the pipe memory.

This bit is cleared when a USB Reset is requested.

Refer to the DPRAM Management chapter for more details.

39.6.3.14 Host Pipe x Status Register

Name: UOTGHS_HSTPIPISRx [x=0..9]

Address: 0x400AC530

Access: Read-only

31	30	29	28	27	26	25	24
–	PBYCT						
23	22	21	20	19	18	17	16
PBYCT				–	CFGOK	–	RWALL
15	14	13	12	11	10	9	8
CURRBK		NBUSYBK		–	–	DTSEQ	
7	6	5	4	3	2	1	0
SHORTPACKETI	RXSTALLDI/CRCERRI	OVERFI	NAKEDI	PERRI	TXSTPI/UNDERFI	TXOUTI	RXINI

- **PBYCT: Pipe Byte Count**

This field contains the byte count of the FIFO.

For OUT pipe, incremented after each byte written by the user into the pipe and decremented after each byte sent to the peripheral.

For IN pipe, incremented after each byte received from the peripheral and decremented after each byte read by the user from the pipe.

This field may be updated 1 clock cycle after the RWALL bit changes, so the user should not poll this field as an interrupt bit.

- **CFGOK: Configuration OK Status**

This bit is set/cleared when the UOTGHS_HSTPIPCFGx.ALLOC bit is set.

This bit is set if the pipe x number of banks (UOTGHS_HSTPIPCFGx.PBK) and size (UOTGHS_HSTPIPCFGx.PSIZE) are correct compared to the maximal allowed number of banks and size for this pipe and to the maximal FIFO size (i.e., the DPRAM size).

If this bit is cleared, the user should rewrite correct values of the PBK and PSIZE field in the UOTGHS_HSTPIPCFGx register.

- **RWALL: Read-write Allowed**

For OUT pipe, this bit is set when the current bank is not full, i.e., the software can write further data into the FIFO.

For IN pipe, this bit is set when the current bank is not empty, i.e., the software can read further data from the FIFO.

This bit is cleared otherwise.

This bit is also cleared when the RXSTALLDI or the PERRI bit is one.

- **CURRBK: Current Bank**

For non-control pipe, this field indicates the number of the current bank.

Value	Name	Description
0	BANK0	Current bank is bank0
1	BANK1	Current bank is bank1
2	BANK2	Current bank is bank2
3		Reserved

This field may be updated 1 clock cycle after the RWALL bit changes, so the user shall not poll this field as an interrupt bit.

- **NBUSYBK: Number of Busy Banks**

This field indicates the number of busy banks.

For OUT pipe, this field indicates the number of busy banks, filled by the user, ready for OUT transfer. When all banks are busy, this triggers a PEP_x interrupt if UOTGHS_HSTPIPIMRx.NBUSYBKE is one.

For IN pipe, this field indicates the number of busy banks filled by IN transaction from the Device. When all banks are free, this triggers a PEP_x interrupt if UOTGHS_HSTPIPIMRx.NBUSYBKE is one.

Value	Name	Description
0	0_BUSY	0 busy bank (all banks free)
1	1_BUSY	1 busy bank
2	2_BUSY	2 busy banks
3	3_BUSY	3 busy banks

- **DTSEQ: Data Toggle Sequence**

This field indicates the data PID of the current bank.

Value	Name	Description
0	DATA0	Data0 toggle sequence
1	DATA1	Data1 toggle sequence
2		reserved
3		reserved

For OUT pipe, this field indicates the data toggle of the next packet that will be sent.

For IN pipe, this field indicates the data toggle of the received packet stored in the current bank.

- **SHORTPACKETI: Short Packet Interrupt**

This bit is set when a short packet is received by the host controller (packet length inferior to the PSIZE programmed field).

This bit is cleared when UOTGHS_HSTPIPICR.SHORTPACKETIC bit is written to one.

- **RXSTALLDI: Received STALLed Interrupt**

This bit is set, for all endpoints but isochronous, when a STALL handshake has been received on the current bank of the pipe. The Pipe is automatically frozen. This triggers an interrupt if UOTGHS_HSTPIPIMR.RXSTALLE bit is one.

This bit is cleared when UOTGHS_HSTPIPICR.RXSTALLDIC bit is written to one.

- **CRCERRI: CRC Error Interrupt**

This bit is set, for isochronous endpoint, when a CRC error occurs on the current bank of the Pipe. This triggers an interrupt if UOTGHS_HSTPIPIMR.TXSTPE bit is one.

This bit is cleared when UOTGHS_HSTPIPICR.CRCERRIC bit is written to one.

- **OVERFI: Overflow Interrupt**

This bit is set when the current pipe has received more data than the maximum length of the current pipe. An interrupt is triggered if UOTGHS_HSTPIPIMR.OVERFIE bit is one.

This bit is cleared when UOTGHS_HSTPIPICR.OVERFIC bit is written to one.

- **NAKEDI: NAKed Interrupt**

This bit is set when a NAK has been received on the current bank of the pipe. This triggers an interrupt if UOTGHS_HSTPIPIMR.NAKEDE bit is one.

This bit is cleared when UOTGHS_HSTPIPICR.NAKEDIC bit written to one.

- **PERRI: Pipe Error Interrupt**

This bit is set when an error occurs on the current bank of the pipe. This triggers an interrupt if UOTGHS_HSTPIPIMR.PERRE bit is set. Refer to the UOTGHS_HSTPIPERRx register to determine the source of the error.

This bit is cleared when the error source bit is cleared.

- **TXSTPI: Transmitted SETUP Interrupt**

This bit is set, for Control endpoints, when the current SETUP bank is free and can be filled. This triggers an interrupt if UOTGHS_HSTPIPIMR.TXSTPE bit is one.

This bit is cleared when UOTGHS_HSTPIPICR.TXSTPIC bit is written to one.

- **UNDERFI: Underflow Interrupt**

This bit is set, for isochronous and Interrupt IN/OUT pipe, when an error flow occurs. This triggers an interrupt if the UNDERFIE bit is one.

This bit is set, for Isochronous or interrupt OUT pipe, when a transaction underflow occurs in the current pipe. (The pipe cannot send the OUT data packet in time because the current bank is not ready). A zero-length-packet (ZLP) will be sent instead.

This bit is set, for Isochronous or interrupt IN pipe, when a transaction flow error occurs in the current pipe. i.e, the current bank of the pipe is not free whereas a new IN USB packet is received. This packet is not stored in the bank. For Interrupt pipe, the overflowed packet is ACKed to respect the USB standard.

This bit is cleared when UOTGHS_HSTPIPICR.UNDERFIEC bit is written to one.

- **TXOUTI: Transmitted OUT Data Interrupt**

This bit is set when the current OUT bank is free and can be filled. This triggers an interrupt if UOTGHS_HSTPIPIMR.TXOUTE bit is one.

This bit is cleared when UOTGHS_HSTPIPICR.TXOUTIC bit is written to one.

- **RXINI: Received IN Data Interrupt**

This bit is set when a new USB message is stored in the current bank of the pipe. This triggers an interrupt if UOTGHS_HSTPIPIMR.RXINE bit is one.

This bit is cleared when UOTGHS_HSTPIPICR.RXINIC bit is written to one.

39.6.3.15 Host Pipe x Clear Register

Name: UOTGHS_HSTPIPICRx [x=0..9]

Address: 0x400AC560

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
SHORT PACKETIC	RXSTALLDIC /CRCERRIC	OVERFIC	NAKEDIC	–	TXSTPIC/ UNDERFIC	TXOUTIC	RXINIC

- **SHORTPACKETIC: Short Packet Interrupt Clear**

Writing a one to this bit will clear SHORTPACKETI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXSTALLDIC: Received STALLed Interrupt Clear**

Writing a one to this bit will clear RXSTALLDI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **CRCERRIC: CRC Error Interrupt Clear**

Writing a one to this bit will clear CRCERRI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **OVERFIC: Overflow Interrupt Clear**

Writing a one to this bit will clear OVERFI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKEDIC: NAKed Interrupt Clear**

Writing a one to this bit will clear NAKEDI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXSTPIC: Transmitted SETUP Interrupt Clear**

Writing a one to this bit will clear TXSTPI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UNDERFIC: Underflow Interrupt Clear**

Writing a one to this bit will clear UNDERFI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXOUTIC: Transmitted OUT Data Interrupt Clear**

Writing a one to this bit will clear TXOUTI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXINIC: Received IN Data Interrupt Clear**

Writing a one to this bit will clear RXINI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.3.16 Host Pipe x Set Register

Name: UOTGHS_HSTPIPIFRx [x=0..9]

Address: 0x400AC590

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
–	–	–	NBUSYBKS	–	–	–	–
7	6	5	4	3	2	1	0
SHORT PACKETIS	RXSTALLDIS/ CRCERRIS	OVERFIS	NAKEDIS	PERRIS	TXSTPIS/ UNDERFIS	TXOUTIS	RXINIS

- **NBUSYBKS: Number of Busy Banks Set**

Writing a one to this bit will set NBUSYBK bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SHORTPACKETIS: Short Packet Interrupt Set**

Writing a one to this bit will set SHORTPACKETI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXSTALLDIS: Received STALLed Interrupt Set**

Writing a one to this bit will set RXSTALLDI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **CRCERRIS: CRC Error Interrupt Set**

Writing a one to this bit will set CRCERRI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **OVERFIS: Overflow Interrupt Set**

Writing a one to this bit will set OVERFI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKEDIS: NAKed Interrupt Set**

Writing a one to this bit will set NAKEDI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PERRIS: Pipe Error Interrupt Set**

Writing a one to this bit will set PERRI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXSTPIS: Transmitted SETUP Interrupt Set**

Writing a one to this bit will set TXSTPI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UNDERFIS: Underflow Interrupt Set**

Writing a one to this bit will set UNDERFI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXOUTIS: Transmitted OUT Data Interrupt Set**

Writing a one to this bit will set TXOUTI bit in UOTGHS_HSTPIPISRx, which may be useful for test or debug purposes.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXINIS: Received IN Data Interrupt Set**

Writing a one to this bit will set RXINI bit in UOTGHS_HSTPIPISRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.3.17 Host Pipe x Mask Register

Name: UOTGHS_HSTPIPIMRx [x=0..9]

Address: 0x400AC5C0

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	RSTDT	PFREEZE	PDISHDMA
15	14	13	12	11	10	9	8
–	FIFOCON	–	NBUSYBKE	–	–	–	–
7	6	5	4	3	2	1	0
SHORT PACKETIE	RXSTALLDE/ CRCERRE	OVERFIE	NAKEDE	PERRE	TXSTPE/ UNDERFIE	TXOUTE	RXINE

- **RSTDT: Reset Data Toggle**

This bit is set when UOTGHS_HSTPIPIER.RSTDTS bit is written to one. This will reset the Data Toggle to its initial value for the current Pipe.

This bit is cleared when proceed.

- **PFREEZE: Pipe Freeze**

This bit is set when UOTGHS_HSTPIPIER.PFREEZES bit is written to one or when the pipe is not configured or when a STALL handshake has been received on this Pipe or when an error occurs on the Pipe (UOTGHS_HSTPIPISR.PERRI is one) or when (INRQ+1) In requests have been processed or when after a Pipe reset (UOTGHS_HSTPIP.PRSTx rising) or a Pipe Enable (UOTGHS_HSTPIP.PEN rising). This will Freeze the Pipe requests generation.

This bit is cleared when UOTGHS_HSTPIPIDR.PFREEZEC bit is written to one. This will enable the Pipe request generation.

- **PDISHDMA: Pipe Interrupts Disable HDMA Request Enable**

See the UOTGHS_DEVEPTIMR.EPDISHDMA bit description.

- **FIFOCON: FIFO Control**

For OUT and SETUP Pipe:

This bit is set when the current bank is free, at the same time than UOTGHS_HSTPIPISR.TXOUTI or TXSTPI.

This bit is cleared when UOTGHS_HSTPIPIDR.FIFOCONC bit is written to one. This will send the FIFO data and switch the bank.

For IN Pipe:

This bit is set when a new IN message is stored in the current bank, at the same time than UOTGHS_HSTPIPISR.RXINI.

This bit is cleared when UOTGHS_HSTPIPIDR.FIFOCONC bit is written to one. This will free the current bank and switch to the next bank.

- **NBUSYBKE: Number of Busy Banks Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.NBUSYBKES bit is written to one.This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.NBUSYBKE).

This bit is cleared when UOTGHS_HSTPIPIDR.NBUSYBKEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.NBUSYBKE).

- **SHORTPACKETIE: Short Packet Interrupt Enable**

If this bit is set for non-control OUT pipes, a short packet transmission is guaranteed upon ending a DMA transfer, thus signaling an end of transfer, provided that the End of DMA Buffer Output Enable (UOTGHS_HSTDMACONTROL.END_B_EN) bit and the Automatic Switch (UOTGHS_HSTPIPCFG.AUTOSW) bit are written to one.

This bit is set when UOTGHS_HSTPIPIER.SHORTPACKETIES bit is written to one. This will enable the Transmitted IN Data IT (UOTGHS_HSTPIPIMR.SHORTPACKETIE).

This bit is cleared when UOTGHS_HSTPIPIDR.SHORTPACKETEC bit is written to one. This will disable the Transmitted IN Data IT (UOTGHS_HSTPIPIMR.SHORTPACKETE).

- **RXSTALLDE: Received STALLed Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.RXSTALLDES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.RXSTALLDE).

This bit is cleared when UOTGHS_HSTPIPIDR.RXSTALLDEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.RXSTALLDE).

- **CRCERRE: CRC Error Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.CRCERRES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.CRCERRE).

This bit is cleared when UOTGHS_HSTPIPIDR.CRCERREC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.CRCERRE).

- **OVERFIE: Overflow Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.OVERFIES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.OVERFIE).

This bit is cleared when UOTGHS_HSTPIPIDR.OVERFIEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.OVERFIE).

- **NAKEDE: NAKed Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.NAKEDES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.NAKEDE).

This bit is cleared when UOTGHS_HSTPIPIDR.NAKEDEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.NAKEDE).

- **PERRE: Pipe Error Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.PERRES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.PERRE).

This bit is cleared when UOTGHS_HSTPIPIDR.PERREC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.PERRE).

- **TXSTPE: Transmitted SETUP Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.TXSTPES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.TXSTPE).

This bit is cleared when UOTGHS_HSTPIPIDR.TXSTPEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.TXSTPE).

- **UNDERFIE: Underflow Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.UNDERFIES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.UNDERFIE).

This bit is cleared when UOTGHS_HSTPIPIDR.UNDERFIEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.UNDERFIE).

- **TXOUTE: Transmitted OUT Data Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.TXOUTES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.TXOUTE).

This bit is cleared when UOTGHS_HSTPIPIDR.TXOUTEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.TXOUTE).

- **RXINE: Received IN Data Interrupt Enable**

This bit is set when UOTGHS_HSTPIPIER.RXINES bit is written to one. This will enable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.RXINE).

This bit is cleared when UOTGHS_HSTPIPIDR.RXINEC bit is written to one. This will disable the Transmitted IN Data interrupt (UOTGHS_HSTPIPIMR.RXINE).

39.6.3.18 Host Pipe x Disable Register

Name: UOTGHS_HSTPIPIDRx [x=0..9]

Address: 0x400AC620

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	PFREEZEC	PDISHDMAC
15	14	13	12	11	10	9	8
–	FIFOCONC	–	NBUSYBKEC	–	–	–	–
7	6	5	4	3	2	1	0
SHORT PACKETIEC	RXSTALLDEC/ CRCERREC	OVERFIEC	NAKEDEC	PERREC	TXSTPEC/ UNDERFIEC	TXOUTEC	RXINEC

- **PFREEZEC: Pipe Freeze Disable**

Writing a one to this bit will clear PFREEZE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PDISHDMAC: Pipe Interrupts Disable HDMA Request Disable**

Writing a one to this bit will clear PDISHDMA bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **FIFOCONC: FIFO Control Disable**

Writing a one to this bit will clear FIFOCON bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NBUSYBKEC: Number of Busy Banks Disable**

Writing a one to this bit will clear NBUSYBKE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SHORTPACKETIEC: Short Packet Interrupt Disable**

Writing a one to this bit will clear SHORTPACKETIE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXSTALLDEC: Received STALLed Interrupt Disable**

Writing a one to this bit will clear RXSTALLDE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **CRCERREC: CRC Error Interrupt Disable**

Writing a one to this bit will clear CRCERRE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **OVERFIEC: Overflow Interrupt Disable**

Writing a one to this bit will clear OVERFIE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKEDEC: NAKed Interrupt Disable**

Writing a one to this bit will clear NAKEDE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PERREC: Pipe Error Interrupt Disable**

Writing a one to this bit will clear PERRE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXSTPEC: Transmitted SETUP Interrupt Disable**

Writing a one to this bit will clear TXSTPE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UNDERFIEC: Underflow Interrupt Disable**

Writing a one to this bit will clear UNDERFIE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXOUTEC: Transmitted OUT Data Interrupt Disable**

Writing a one to this bit will clear TXOUTE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXINEC: Received IN Data Interrupt Disable**

Writing a one to this bit will clear RXINI bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.3.19 Host Pipe x Enable Register

Name: UOTGHS_HSTPIPIERx [x=0..9]

Address: 0x400AC5F0

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	RSTDTS	PFREEZES	PDISHDMAS
15	14	13	12	11	10	9	8
–	–	–	NBUSYBKES	–	–	–	–
7	6	5	4	3	2	1	0
SHORT PACKETIES	RXSTALLDES/ CRCERRES	OVERFIES	NAKEDES	PERRES	TXSTPES/ UNDERFIES	TXOUTES	RXINES

- **RSTDTS: Reset Data Toggle Enable**

Writing a one to this bit will set RSTDT bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PFREEZES: Pipe Freeze Enable**

Writing a one to this bit will set PFREEZE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PDISHDMAS: Pipe Interrupts Disable HDMA Request Enable**

Writing a one to this bit will set PDISHDMA bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NBUSYBKES: Number of Busy Banks Enable**

Writing a one to this bit will set NBUSYBKE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **SHORTPACKETIES: Short Packet Interrupt Enable**

Writing a one to this bit will set SHORTPACKETIE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXSTALLDES: Received STALLed Interrupt Enable**

Writing a one to this bit will set RXSTALLDE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **CRCERRES: CRC Error Interrupt Enable**

Writing a one to this bit will set CRCERRE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **OVERFIES: Overflow Interrupt Enable**

Writing a one to this bit will set OVERFIE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **NAKEDES: NAKed Interrupt Enable**

Writing a one to this bit will set NAKEDE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **PERRES: Pipe Error Interrupt Enable**

Writing a one to this bit will set PERRE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXSTPES: Transmitted SETUP Interrupt Enable**

Writing a one to this bit will set TXSTPE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **UNDERFIES: Underflow Interrupt Enable**

Writing a one to this bit will set UNDERFIE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **TXOUTES: Transmitted OUT Data Interrupt Enable**

Writing a one to this bit will set TXOUTE bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

- **RXINES: Received IN Data Interrupt Enable**

Writing a one to this bit will set RXINI bit in UOTGHS_HSTPIPIMRx.

Writing a zero to this bit has no effect.

This bit always reads as zero.

39.6.3.20 Host Pipe x IN Request Register

Name: UOTGHS_HSTPIPINRQx [x=0..9]

Address: 0x400AC650

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
–	–	–	–	–	–	–	INMODE
7	6	5	4	3	2	1	0
INRQ							

- **INMODE: IN Request Mode**

Writing a one to this bit will allow the UOTGHS to perform infinite IN requests when the Pipe is not frozen.

Writing a zero to this bit will perform a pre-defined number of IN requests. This number is the INRQ field.

- **INRQ: IN Request Number before Freeze**

This field contains the number of IN transactions before the UOTGHS freezes the pipe. The UOTGHS will perform (INRQ+1) IN requests before to freeze the Pipe. This counter is automatically decreased by 1 each time a IN request has been successfully performed.

This register has no effect when the INMODE bit is one (infinite IN requests generation till the pipe is not frozen).

39.6.3.21 Host Pipe x Error Register

Name: UOTGHS_HSTPIPERRx [x=0..9]

Address: 0x400AC680

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
–	COUNTER		CRC16	TIMEOUT	PID	DATAPID	DATATGL

- **COUNTER: Error Counter**

This field is incremented each time an error occurs (CRC16, TIMEOUT, PID, DATAPID or DATATGL).

This field is cleared when receiving a good USB packet without any error.

When this field reaches 3 (i.e., 3 consecutive errors), this pipe is automatically frozen (UOTGHS_HSTPIPIMRx.PFREEZE is set).

Writing 0b00 to this field will clear the counter.

- **CRC16: CRC16 Error**

This bit is set when a CRC16 error has been detected.

Writing a zero to this bit will clear the bit.

Writing a one to this bit has no effect.

- **TIMEOUT: Time-Out Error**

This bit is set when a Time-Out error has been detected.

Writing a zero to this bit will clear the bit.

Writing a one to this bit has no effect.

- **PID: PID Error**

This bit is set when a PID error has been detected.

Writing a zero to this bit will clear the bit.

Writing a one to this bit has no effect.

- **DATAPID: Data PID Error**

This bit is set when a Data PID error has been detected.

Writing a zero to this bit will clear the bit.

Writing a one to this bit has no effect.

- **DATATGL: Data Toggle Error**

This bit is set when a Data Toggle error has been detected.

Writing a zero to this bit will clear the bit.

Writing a one to this bit has no effect.

39.6.3.22 Host DMA Channel x Next Descriptor Address Register

Name: UOTGHS_HSTDMANXTDSCx [x=1..6]

Address: 0x400AC710 [1], 0x400AC720 [2], 0x400AC730 [3], 0x400AC740 [4], 0x400AC750 [5], 0x400AC760 [6], 0x400AC770 [7]

Access: Read-write

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

- **NXT_DSC_ADD: Next Descriptor Address**

This field points to the next channel descriptor to be processed. This channel descriptor must be aligned, so bits 0 to 3 of the address must be equal to zero.

39.6.3.23 Host DMA Channel x Address Register

Name: UOTGHS_HSTDMAADDRESSx [x=1..6]

Address: 0x400AC714 [1], 0x400AC724 [2], 0x400AC734 [3], 0x400AC744 [4], 0x400AC754 [5], 0x400AC764 [6], 0x400AC774 [7]

Access: Read-write

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

- **BUFF_ADD: Buffer Address**

This field determines the AHB bus starting address of a DMA channel transfer.

Channel start and end addresses may be aligned on any byte boundary.

The firmware may write this field only when the UOTGHS_HSTDMASTATUS.CHANN_ENB bit is clear.

This field is updated at the end of the address phase of the current access to the AHB bus. It is incrementing of the access byte width. The access width is 4 bytes (or less) at packet start or end, if the start or end address is not aligned on a word boundary.

The packet start address is either the channel start address or the next channel address to be accessed in the channel buffer.

The packet end address is either the channel end address or the latest channel address accessed in the channel buffer.

The channel start address is written by software or loaded from the descriptor, whereas the channel end address is either determined by the end of buffer or the USB device, USB end of transfer if the UOTGHS_HSTDMACONTROLx.END_TR_EN bit is set.

39.6.3.24 Host DMA Channel x Control Register

Name: UOTGHS_HSTDMACONTROLx [x=1..6]

Address: 0x400AC718 [1], 0x400AC728 [2], 0x400AC738 [3], 0x400AC748 [4], 0x400AC758 [5], 0x400AC768 [6], 0x400AC778 [7]

Access: Read-write

31	30	29	28	27	26	25	24
BUFF_LENGTH							
23	22	21	20	19	18	17	16
BUFF_LENGTH							
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
BURST_LCK	DESC_LD_IT	END_BUFFIT	END_TR_IT	END_B_EN	END_TR_EN	LDNXT_DSC	CHANN_ENB

- **CHANN_ENB: Channel Enable Command**

0: DMA channel is disabled and no transfer will occur upon request. This bit is also cleared by hardware when the channel source bus is disabled at the end of buffer.

If LDNXT_DSC bit has been cleared by descriptor loading, the firmware will have to set the corresponding CHANN_ENB bit to start the described transfer, if needed.

If LDNXT_DSC bit is cleared, the channel is frozen and the channel registers may then be read and/or written reliably as soon as both UOTGHS_HSTDMASTATUS.CHANN_ENB and CHANN_ACT flags read as 0.

If a channel request is currently serviced when this bit is cleared, the DMA FIFO buffer is drained until it is empty, then UOTGHS_HSTDMASTATUS.CHANN_ENB bit is cleared.

If LDNXT_DSC bit is set or after this bit clearing, then the currently loaded descriptor is skipped (no data transfer occurs) and the next descriptor is immediately loaded.

1: UOTGHS_HSTDMASTATUS.CHANN_ENB bit will be set, thus enabling DMA channel data transfer. Then any pending request will start the transfer. This may be used to start or resume any requested transfer.

- **LDNXT_DSC: Load Next Channel Transfer Descriptor Enable Command**

0: no channel register is loaded after the end of the channel transfer.

1: the channel controller loads the next descriptor after the end of the current transfer, i.e. when the UOTGHS_HSTDMASTATUS.CHANN_ENB bit is reset.

If CHANN_ENB bit is cleared, the next descriptor is immediately loaded upon transfer request.

DMA Channel Control Command Summary

Value	Name	Description
0	STOP_NOW	Stop now
1	RUN_AND_STOP	Run and stop at end of buffer
2	LOAD_NEXT_DESC	Load next descriptor now
3	RUN_AND_LINK	Run and link at end of buffer

- **END_TR_EN: End of Transfer Enable (Control)**

Used for OUT transfers only.

0: USB end of transfer is ignored.

1: UOTGHS device can put an end to the current buffer transfer.

When set, a BULK or INTERRUPT short packet will close the current buffer and the UOTGHS_HSTDMASTATUSx.END_TR_ST flag will be raised.

This is intended for a UOTGHS non-prenegotiated end of transfer (BULK or INTERRUPT) data buffer closure.

- **END_B_EN: End of Buffer Enable Control**

0: DMA Buffer End has no impact on USB packet transfer.

1: the pipe can validate the packet (according to the values programmed in the UOTGHS_HSTPIPCFGx.AUTOSW field, and in the UOTGHS_HSTPIPIMRx.SHORTPACKETIE field) at DMA Buffer End, i.e. when UOTGHS_HSTDMASTATUS.BUFF_COUNT reaches 0.

This is mainly for short packet OUT validation initiated by the DMA reaching the end of buffer, but could be used for IN packet truncation (discarding of unwanted packet data) at the end of DMA buffer.

- **END_TR_IT: End of Transfer Interrupt Enable**

0: UOTGHS device initiated buffer transfer completion will not trigger any interrupt at UOTGHS_HSTDMASTATUSx.END_TR_ST rising.

1: an interrupt is sent after the buffer transfer is complete, if the UOTGHS device has ended the buffer transfer.

Use when the receive size is unknown.

- **END_BUFFIT: End of Buffer Interrupt Enable**

0: UOTGHS_HSTDMASTATUSx.END_BF_ST rising will not trigger any interrupt.

1: an interrupt is generated when UOTGHS_HSTDMASTATUSx.BUFF_COUNT reaches zero.

- **DESC_LD_IT: Descriptor Loaded Interrupt Enable**

0: UOTGHS_HSTDMASTATUSx.DESC_LDST rising will not trigger any interrupt.

1: an interrupt is generated when a descriptor has been loaded from the bus.

- **BURST_LCK: Burst Lock Enable**

0: the DMA never locks the bus access.

1: USB packets AHB data bursts are locked for maximum optimization of the bus bandwidth usage and maximization of fly-by AHB burst duration.

- **BUFF_LENGTH: Buffer Byte Length (Write-only)**

This field determines the number of bytes to be transferred until end of buffer. The maximum channel transfer size (32 KBytes) is reached when this field is 0 (default value). If the transfer size is unknown, this field should be set to 0, but the transfer end may occur earlier under USB device control.

When this field is written, the UOTGHS_HSTDMASTATUSx.BUFF_COUNT field is updated with the write value.

Notes:
1. Bits [31:2] are only writable when issuing a channel Control Command other than “Stop Now”.
2. For reliability it is highly recommended to wait for both UOTGHS_HSTDMASTATUSx.CHAN_ACT and CHAN_ENB flags are at 0, thus ensuring the channel has been stopped before issuing a command other than “Stop Now”.

39.6.3.25 Host DMA Channel x Status Register

Name: UOTGHS_HSTDMASTATUSx [x=1..6]

Address: 0x400AC71C [1], 0x400AC72C [2], 0x400AC73C [3], 0x400AC74C [4], 0x400AC75C [5], 0x400AC76C [6], 0x400AC77C [7]

Access: Read-write

31	30	29	28	27	26	25	24
BUFF_COUNT							
23	22	21	20	19	18	17	16
BUFF_COUNT							
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	DESC_LDST	END_BF_ST	END_TR_ST	–	–	CHANN_ACT	CHANN_ENB

- **CHANN_ENB: Channel Enable Status**

0: if cleared, the DMA channel no longer transfers data, and may load the next descriptor if UOTGHS_HSTDMACONTROLx.LDNXT_DSC bit is set.

When any transfer is ended either due to an elapsed byte count or a UOTGHS device initiated transfer end, this bit is automatically reset.

1: if set, the DMA channel is currently enabled and transfers data upon request.

This bit is normally set or cleared by writing into UOTGHS_HSTDMACONTROLx.CHANN_ENB bit field either by software or descriptor loading.

If a channel request is currently serviced when UOTGHS_HSTDMACONTROLx.CHANN_ENB bit is cleared, the DMA FIFO buffer is drained until it is empty, then this status bit is cleared.

- **CHANN_ACT: Channel Active Status**

0: the DMA channel is no longer trying to source the packet data.

When a packet transfer is ended this bit is automatically reset.

1: the DMA channel is currently trying to source packet data, i.e. selected as the highest-priority requesting channel.

When a packet transfer cannot be completed due to an END_BF_ST, this flag stays set during the next channel descriptor load (if any) and potentially until UOTGHS packet transfer completion, if allowed by the new descriptor.

- **END_TR_ST: End of Channel Transfer Status**

0: cleared automatically when read by software.

1: set by hardware when the last packet transfer is complete, if the UOTGHS device has ended the transfer.

Valid until CHANN_ENB flag is cleared at the end of the next buffer transfer.

- **END_BF_ST: End of Channel Buffer Status**

0: cleared automatically when read by software.

1: set by hardware when BUFF_COUNT count-down reaches zero.

Valid until CHANN_ENB flag is cleared at the end of the next buffer transfer.

- **DESC_LDST: Descriptor Loaded Status**

0: cleared automatically when read by software.

1: set by hardware when a descriptor has been loaded from the system bus.

Valid until the CHANN_ENB flag is cleared at the end of the next buffer transfer.

- **BUFF_COUNT: Buffer Byte Count**

This field determines the current number of bytes still to be transferred for this buffer.

This field is decremented from the AHB source bus access byte width at the end of this bus address phase.

The access byte width is 4 by default, or less, at DMA start or end, if the start or end address is not aligned on a word boundary.

At the end of buffer, the DMA accesses the UOTGHS device only for the number of bytes needed to complete it.

Note: For IN pipes, if the receive buffer byte length (UOTGHS_HSTDMACONTROL.BUFF_LENGTH) has been defaulted to zero because the USB transfer length is unknown, the actual buffer byte length received will be 0x10000-BUFF_COUNT.

40. Controller Area Network (CAN)

40.1 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 Mbit/sec.

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.

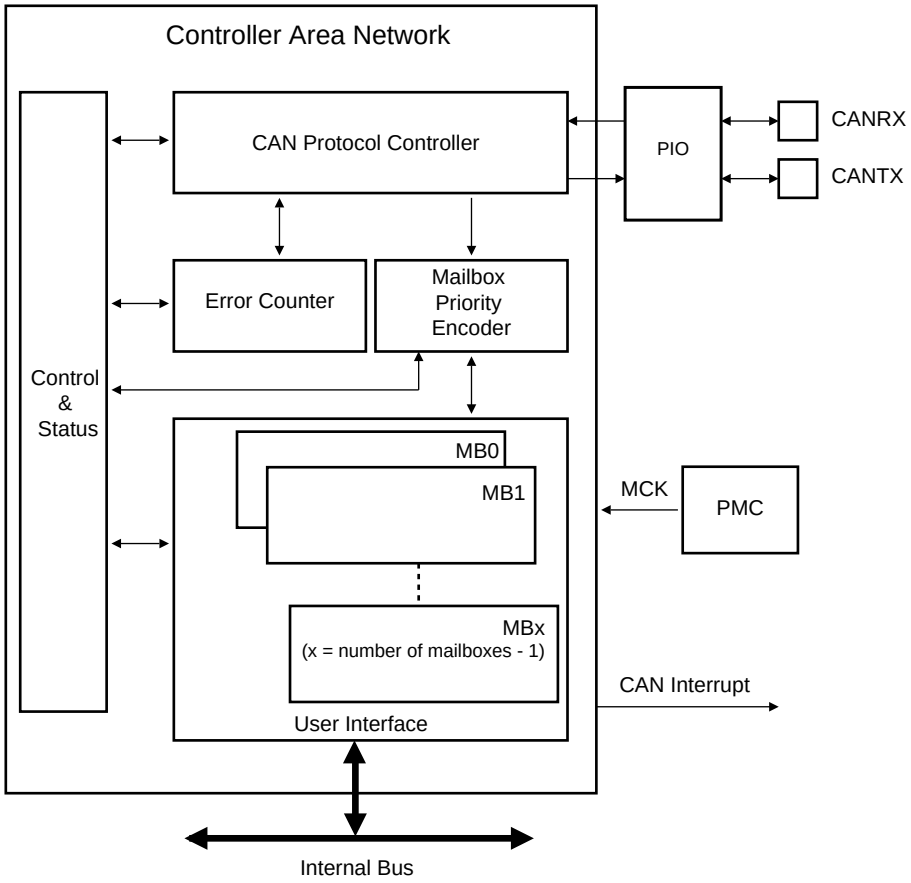
The CAN controller offers optimized features to support the Time Triggered Communication (TTC) protocol.

40.2 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

40.3 Block Diagram

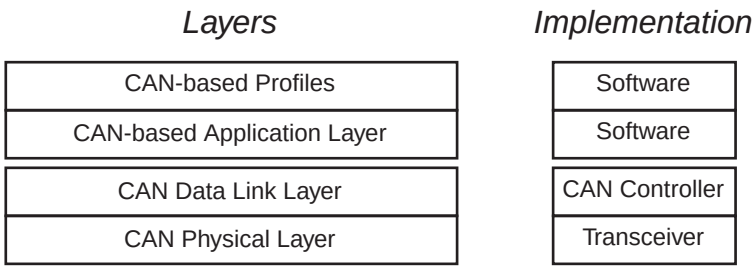
Figure 40-1. CAN Block Diagram



40.4 Application Block Diagram

Figure 40-2.

Application Block Diagram



40.5 I/O Lines Description

Table 40-1. I/O Lines Description

Name	Description	Type
CANRX	CAN Receive Serial Data	Input
CANTX	CAN Transmit Serial Data	Output

40.6 Product Dependencies

40.6.1 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 40-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
CAN0	CANRX0	PA1	A
CAN0	CANTX0	PA0	A
CAN1	CANRX1	PB15	A
CAN1	CANTX1	PB14	A

40.6.2 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.

40.6.3 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 40-3. Peripheral IDs

Instance	ID
CAN0	43
CAN1	44

40.7 CAN Controller Features

40.7.1 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.

40.7.2 Mailbox Organization

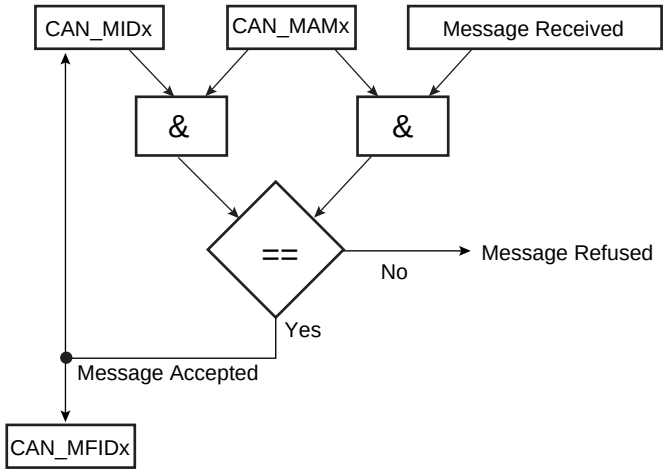
The CAN module has 8 buffers, also called channels or mailboxes. An identifier that corresponds to the CAN identifier is defined for each active mailbox. Message identifiers can match the standard frame identifier or the extended frame identifier. This identifier is defined for the first time during 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.

40.7.2.1 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 40-3. 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 10100010010000100100000100 0 11 00b
ID1 10100010010000100100000100 0 11 01b
ID2 10100010010000100100000100 0 11 10b
ID3 10100010010000100100000100 0 11 11b
ID4 10100010010000100100000100 1 11 00b
ID5 10100010010000100100000100 1 11 01b
ID6 10100010010000100100000100 1 11 10b
ID7 10100010010000100100000100 1 11 11b
```

The CAN_MIDx and CAN_MAMx of Mailbox x must be initialized to the corresponding values:

```
CAN_MIDx = 001 10100010010000100100000100 x 11 xxb
CAN_MAMx = 001 11111111111111111111111111 0 11 00b
```

If Mailbox x receives a message with ID6, then CAN_MIDx and CAN_MFIDx are set:

```
CAN_MIDx = 001 10100010010000100100000100 1 11 10b
CAN_MFIDx = 00000000000000000000000000000110b
```

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]();
```

40.7.2.2 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 40-4.

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.

40.7.2.3 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 40-5.

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 Section 40.7.3). 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.

40.7.3 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 $CAN_MSR_{last_mailbox_number}$ 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.

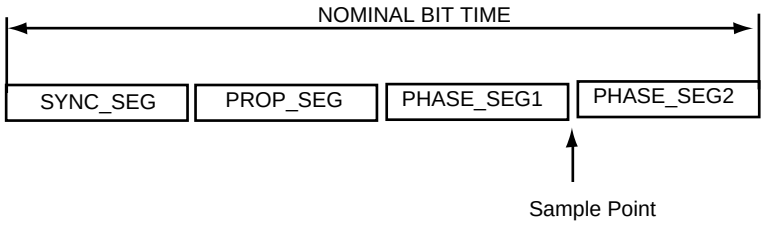
40.7.4 CAN 2.0 Standard Features

40.7.4.1 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 40-4. 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: PROPagation 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).

$$t_{BIT} = t_{CSC} + t_{PRS} + t_{PHS1} + t_{PHS2}$$

The time quantum is calculated as follows:

$$t_{CSC} = (BRP + 1)/MCK$$

Note: The BRP field must be within the range [1, 0x7F], i.e., BRP = 0 is not authorized.

$$t_{PRS} = t_{CSC} \times (PROPAG + 1)$$

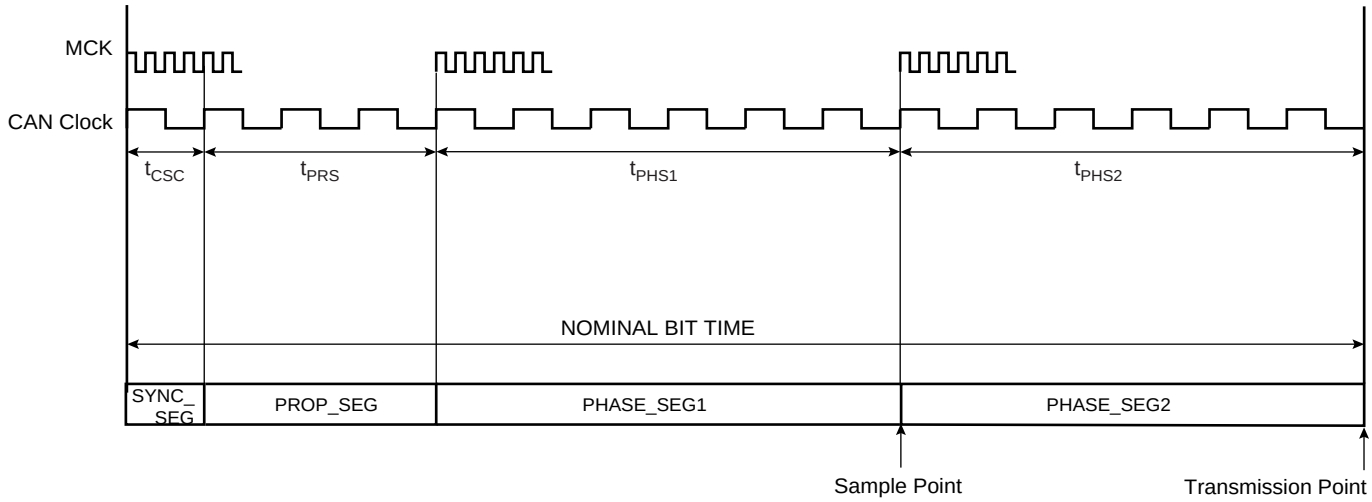
$$t_{PHS1} = t_{CSC} \times (PHASE1 + 1)$$

$$t_{PHS2} = t_{CSC} \times (PHASE2 + 1)$$

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.

$$t_{SJW} = t_{CSC} \times (SJW + 1)$$

Figure 40-5. CAN Bit Timing



Example of bit timing determination for CAN baudrate of 500 Kbit/s:

```
MCK = 48MHz
CAN baudrate= 500kbit/s => 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:
Tcsc = 1 time quanta = bit time / 16 = 125 ns
=> BRP = (Tcsc x MCK) - 1 = 5

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:
Tprs = 2 * (50+30+110) ns = 380 ns = 3 Tcsc
=> PROPAG = Tprs/Tcsc - 1 = 2
The remaining time for the two phase segments is:
Tphs1 + Tphs2 = bit time - Tcsc - Tprs = (16 - 1 - 3)Tcsc
Tphs1 + Tphs2 = 12 Tcsc

Because this number is even, we choose Tphs2 = Tphs1 (else we would choose Tphs2
= Tphs1 + Tcsc)
Tphs1 = Tphs2 = (12/2) Tcsc = 6 Tcsc
=> PHASE1 = PHASE2 = Tphs1/Tcsc - 1 = 5

The resynchronization jump width must be comprised between 1 Tcsc and the
minimum of 4 Tcsc and Tphs1. We choose its maximum value:
Tsjw = Min(4 Tcsc,Tphs1) = 4 Tcsc
=> SJW = Tsjw/Tcsc - 1 = 3

Finally: CAN_BR = 0x00053255
```

CAN Bus Synchronization

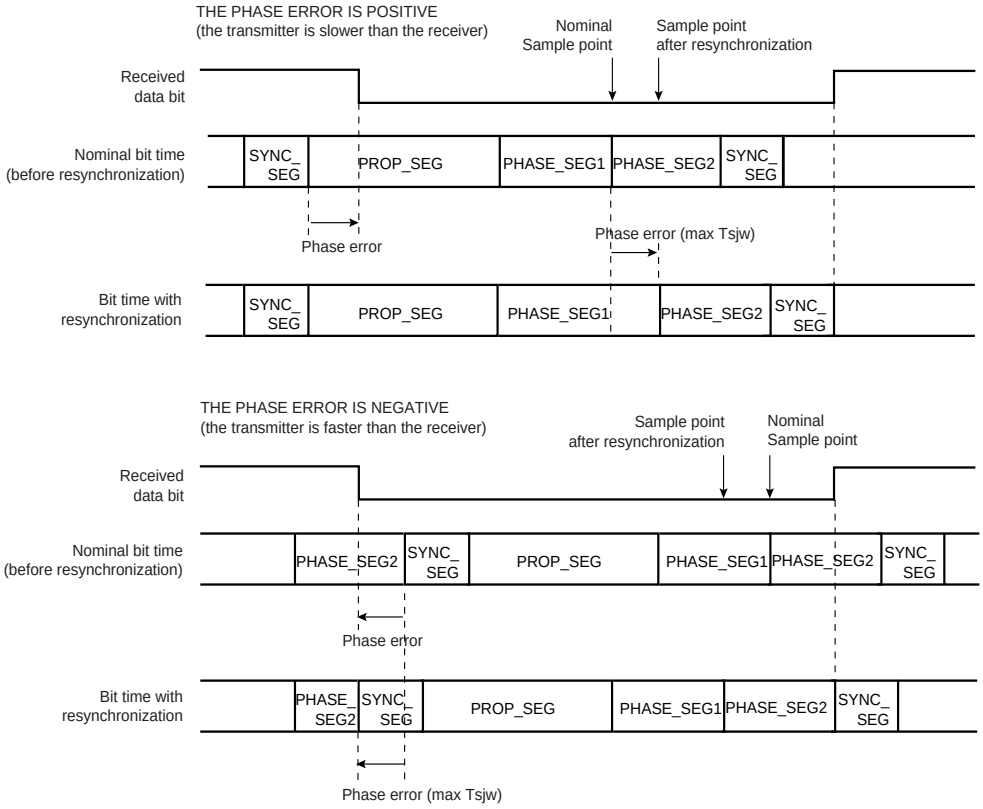
Two types of synchronization are distinguished: “hard synchronization” at the start of a frame and “resynchronization” inside a frame. After a hard synchronization, the bit time is restarted with the end of the SYNC_SEG segment, regardless of the phase error. Resynchronization causes a reduction or increase 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 40-6. 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.

40.7.4.2 Error Detection

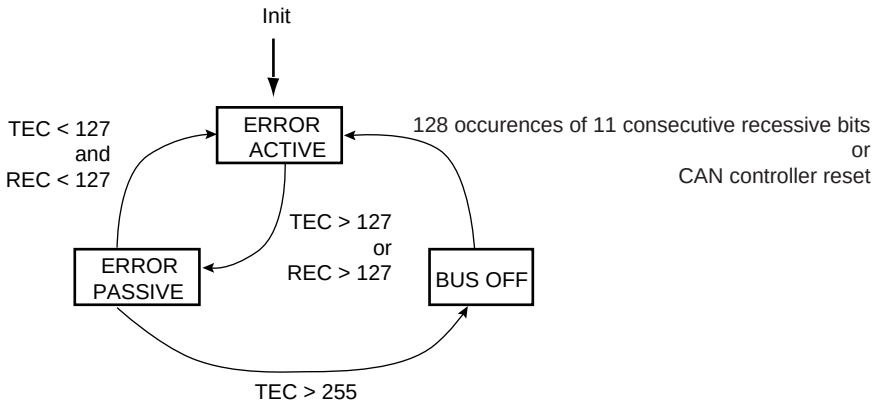
There are five different error types that are not mutually exclusive. Each error concerns only specific fields of the CAN data frame (refer to the Bosch CAN specification for their correspondence):

- 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 40-7. 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 is in 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 is in 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 is in 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

WARN, BOFF, ERRA and ERRP (CAN_SR) represent the current status of the CAN bus and are not latched. They reflect the current TEC and REC (CAN_ECR) values as described in Section “Fault Confinement” on page 1196.

Based on that, if these bits are used as an interrupt, the user can enter into an interrupt and not see the corresponding status register if the TEC and REC counter have changed their state. When entering Bus Off Mode, the only way to exit from this state is 128 occurrences of 11 consecutive recessive bits or a CAN controller reset.

In Error Active Mode, the user reads:

- ERRA =1
- ERRP = 0
- BOFF = 0

In Error Passive Mode, the user reads:

- ERRA = 0
- ERRP =1
- BOFF = 0

In Bus Off Mode, the user reads:

- ERRA = 0
- ERRP =1
- BOFF =1

The CAN interrupt handler should do the following:

- Only enable one error mode interrupt at a time.
- Look at and check the REC and TEC values in the interrupt handler to determine the current state.

40.7.4.3 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.

40.7.5 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.

40.7.5.1 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.

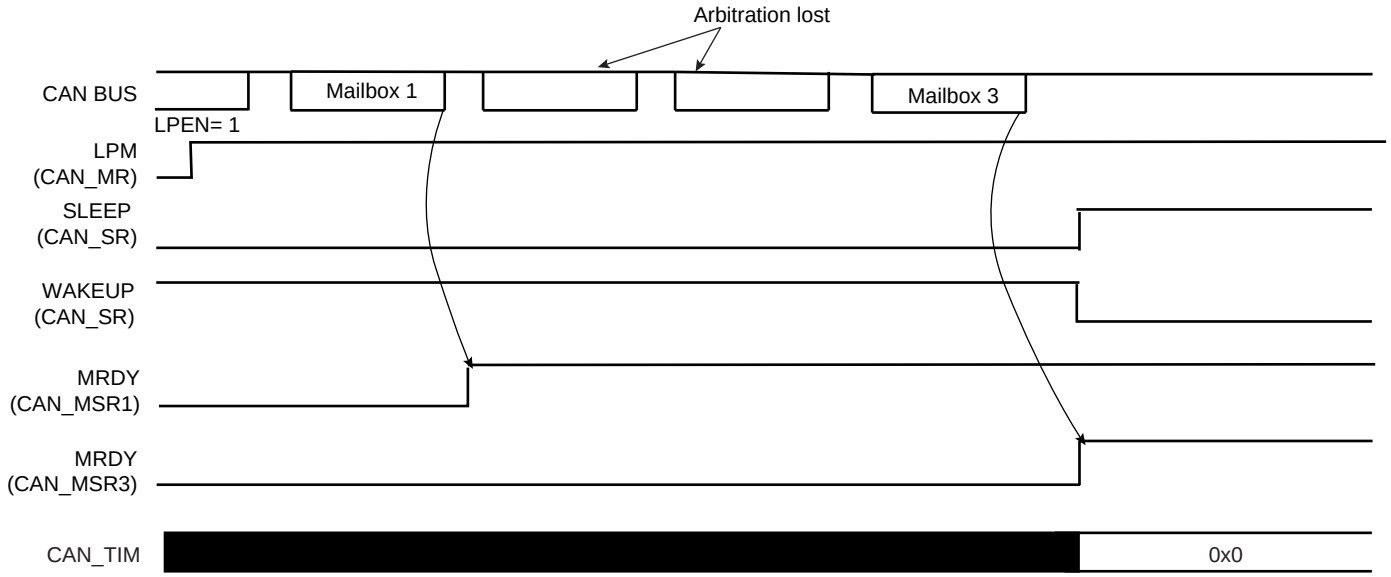
Error counters are disabled while the SLEEP signal is set to one.

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 40-8. Enabling Low-power Mode



40.7.5.2 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

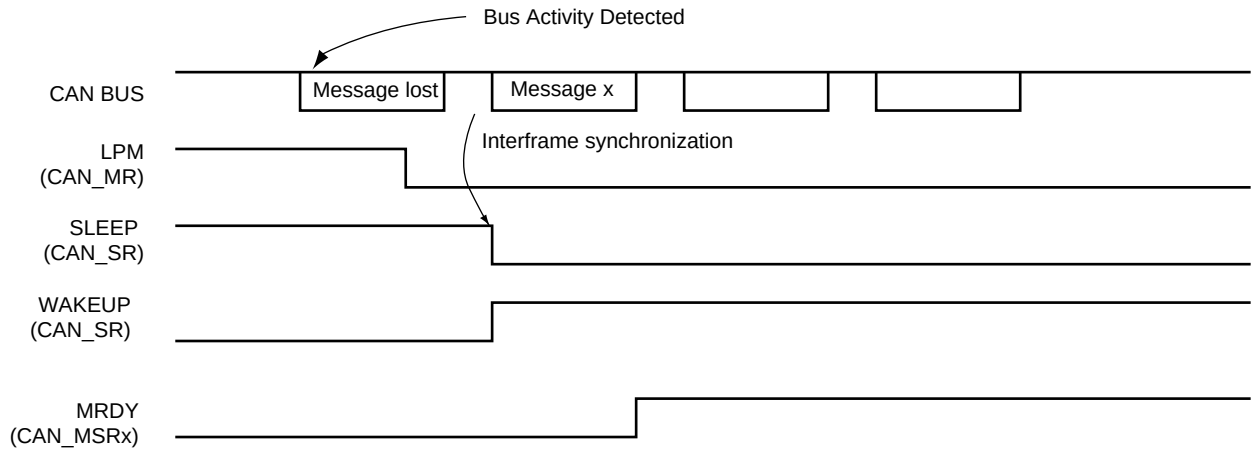
The CAN controller synchronizes itself with the bus activity by checking for eleven consecutive "recessive" bits. Once synchronized, the WAKEUP signal in the CAN_SR register is set.

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 40-9).

Figure 40-9. Disabling Low-power Mode



40.8 Functional Description

40.8.1 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).

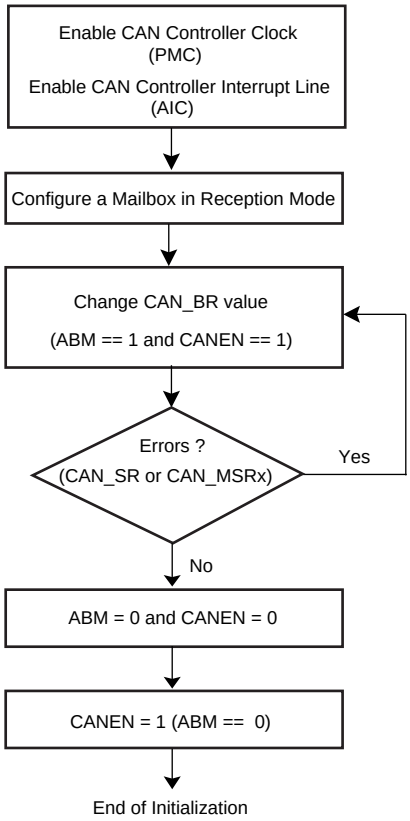
The CAN controller must be initialized with the CAN network parameters. The CAN_BR register defines the sampling point in the bit time period. CAN_BR must be set before the CAN controller is enabled by setting the CANEN field in the CAN_MR register.

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 synchronized 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 40-10. Possible Initialization Procedure



40.8.2 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_TIMESTP 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.

40.8.3 CAN Controller Message Handling

40.8.3.1 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

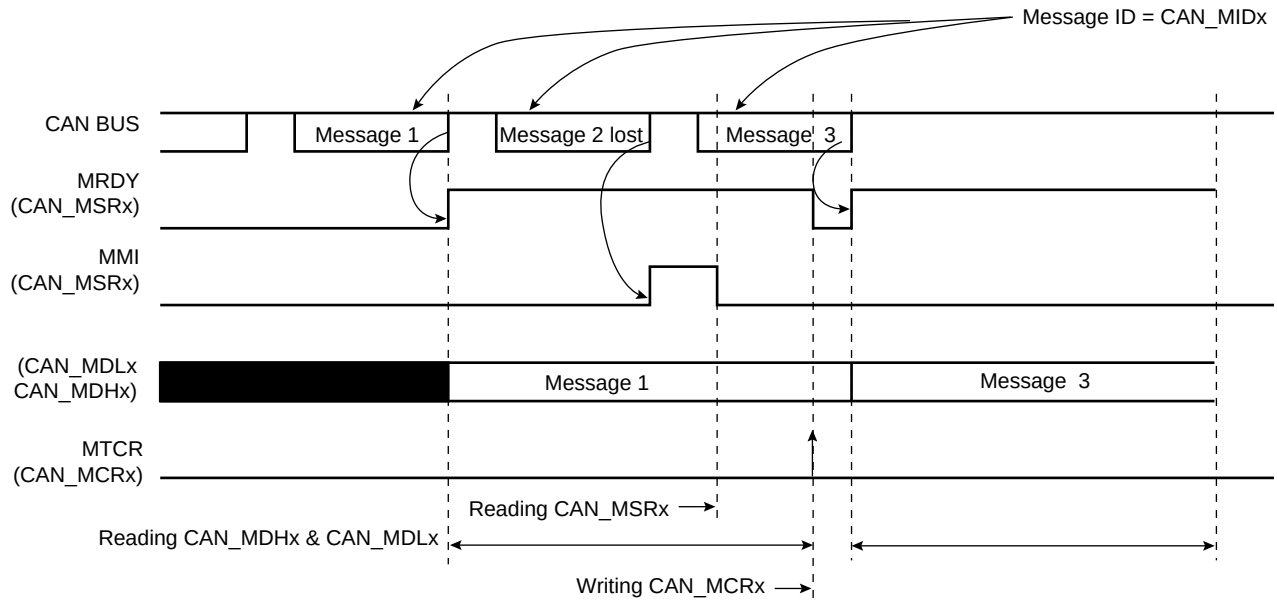
A mailbox is in Receive Mode once the MOT field in the CAN_MMRx register has been configured. Message ID and Message Acceptance Mask must be set before the Receive Mode is enabled.

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 40-11.

Figure 40-11. 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

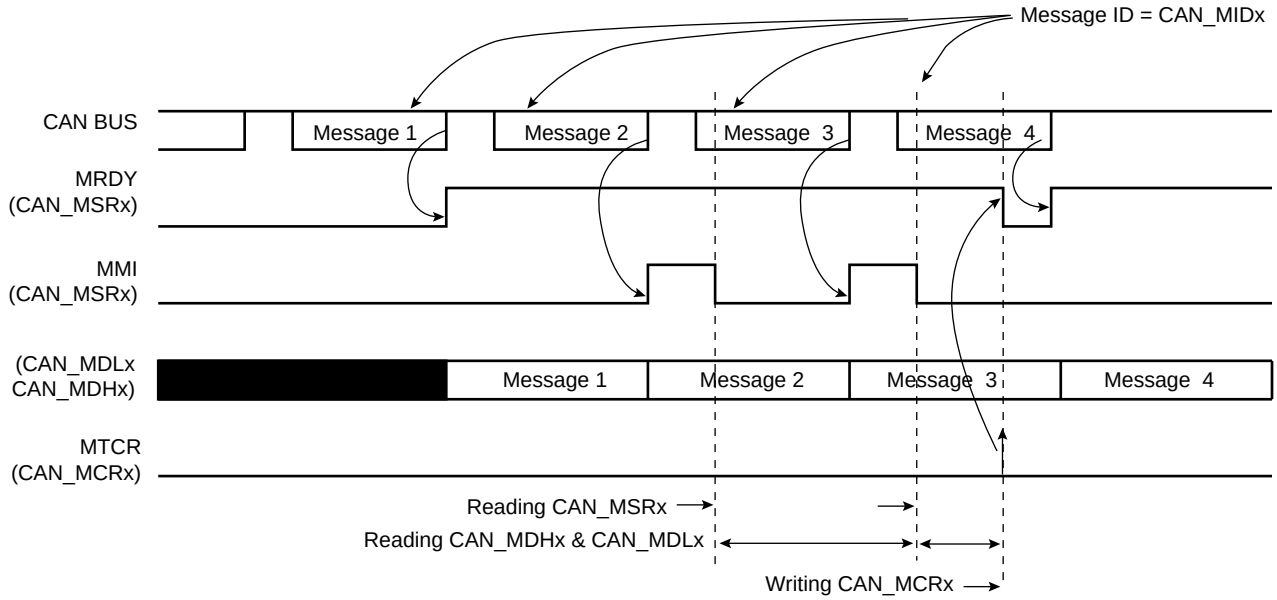
A mailbox is in Receive with Overwrite Mode once the MOT field in the CAN_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

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_MSRx register notifies the software that a message has been dropped by the mailbox. This flag is cleared when reading the CAN_MSRx 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_MSRx 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 40-12).

Figure 40-12. Receive with Overwrite Mailbox

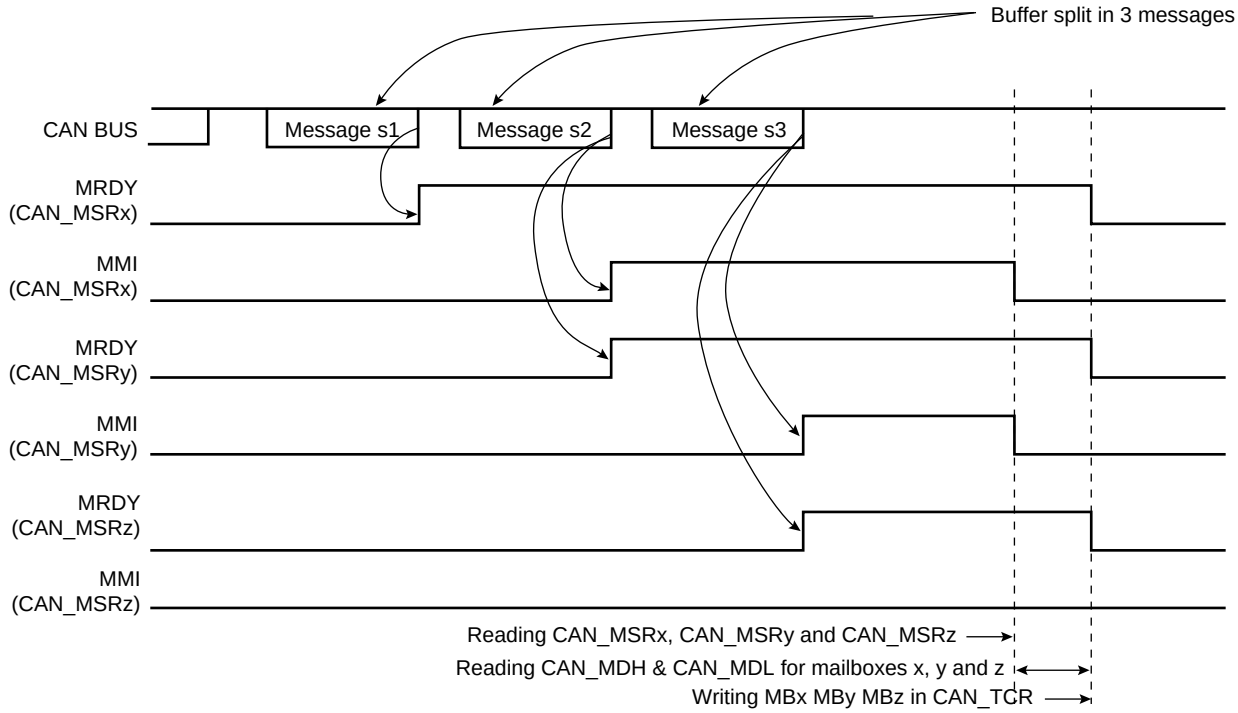


Chaining Mailboxes

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.

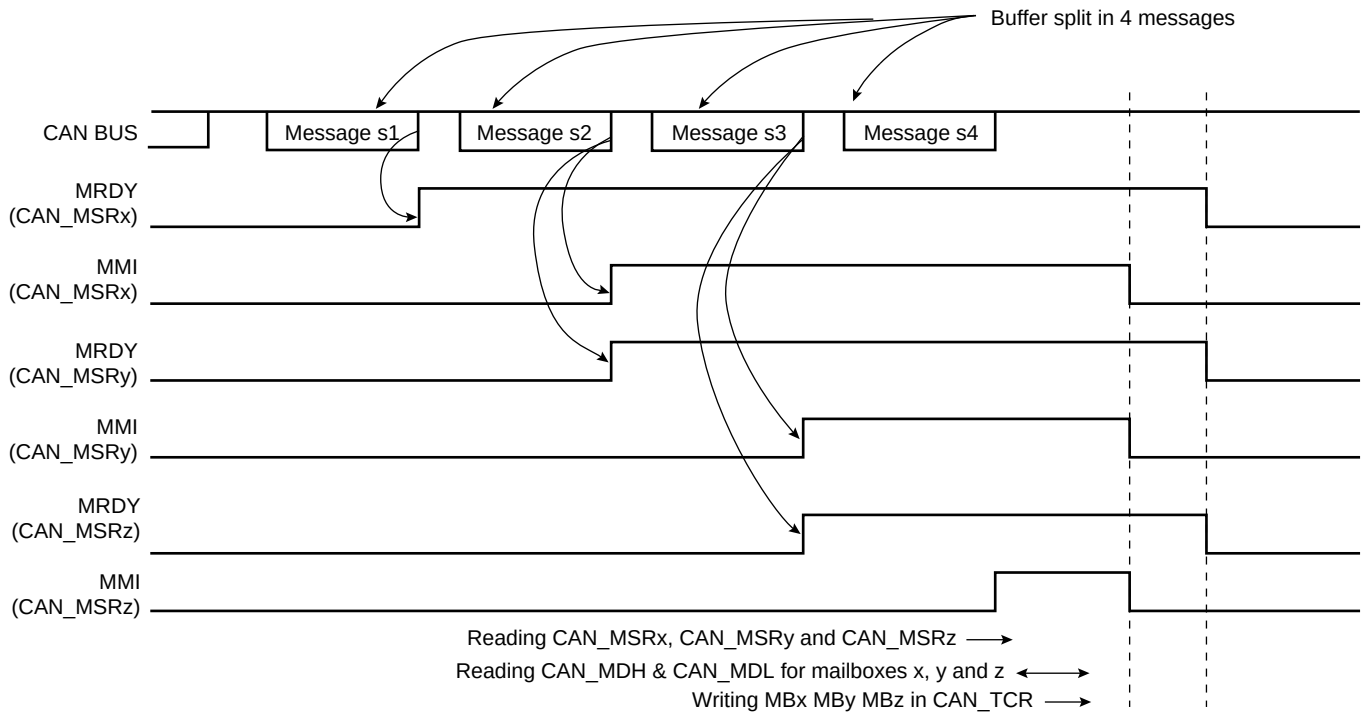
If several mailboxes are chained to receive a buffer split into several messages, all mailboxes except the last one (with the highest number) must be configured in Receive Mode. The first message received is handled by the first mailbox, the second one is refused by the first mailbox and accepted by the second mailbox, the last message is accepted by the last mailbox and refused by previous ones (see Figure 40-13).

Figure 40-13. 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 40-14).

Figure 40-14. Chaining Three Mailboxes to Receive a Buffer Split into Four Messages



40.8.3.2 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 1205.

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.

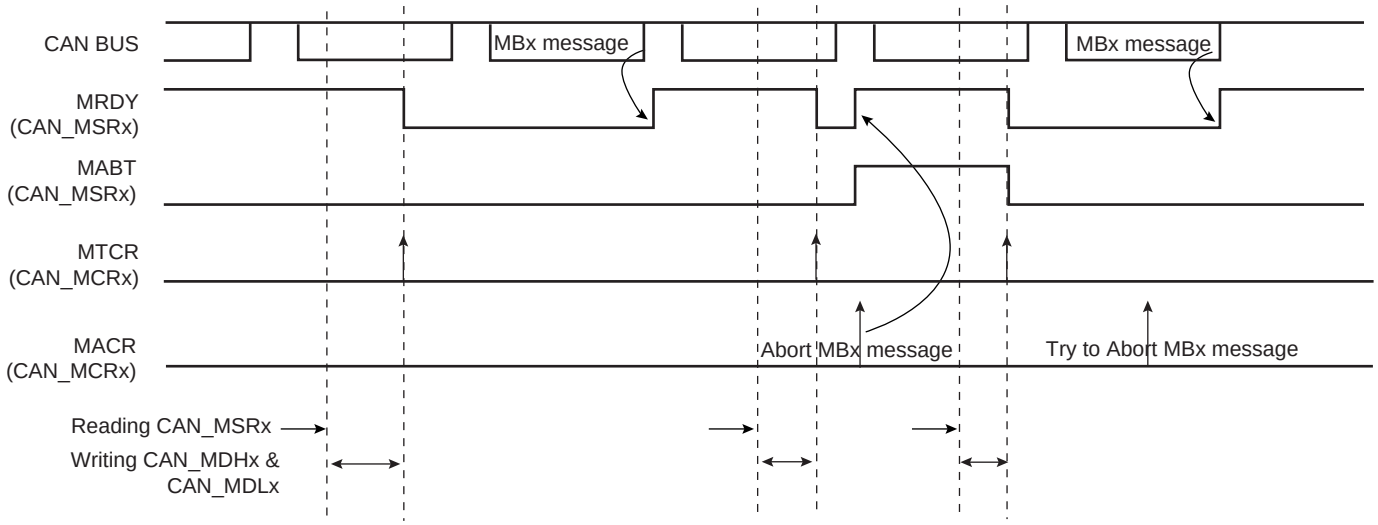
Setting the MACR bit in the CAN_MCRx register aborts the transmission. Transmission for several mailboxes can be aborted by writing MBx fields in the CAN_MACR register. If the message is being sent when the abort command is set, then the application is notified by the MRDY bit set and not the MABT in 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 40-15 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.

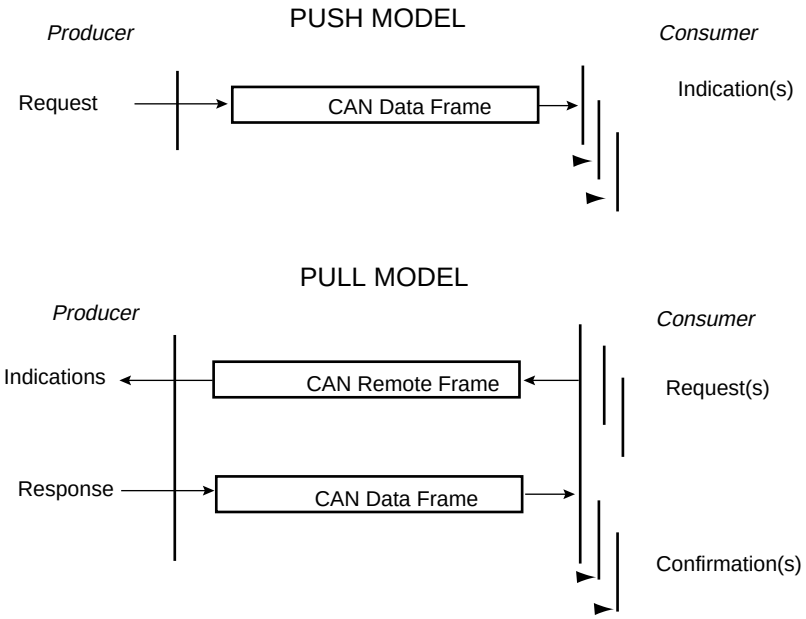
Figure 40-15. Transmitting Messages



40.8.3.3 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 40-16. 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.

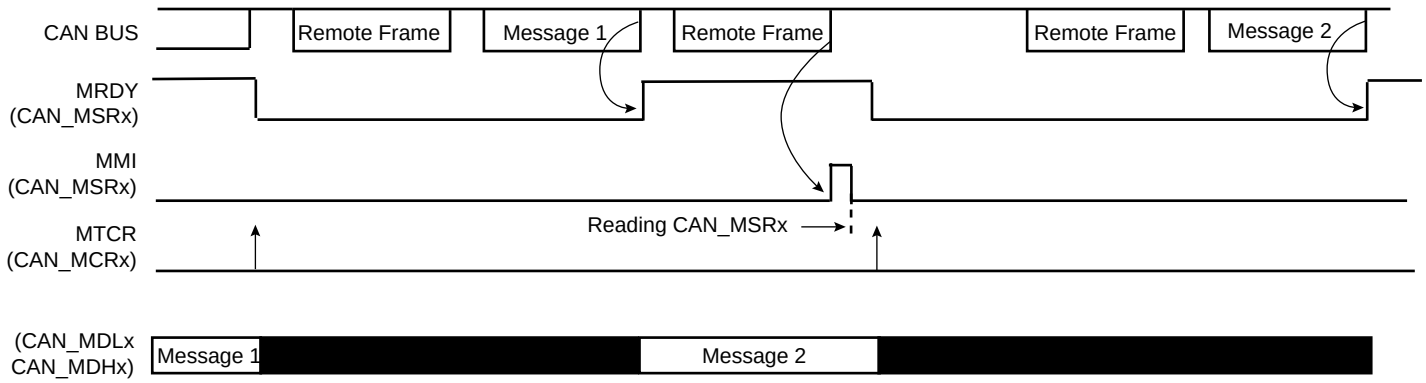
The MRDY flag remains at zero as long as the message has not been sent or aborted. No access to the mailbox data register can be done while MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked according to the mailbox flag in the CAN_IMR 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 1204.

Figure 40-17. 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.

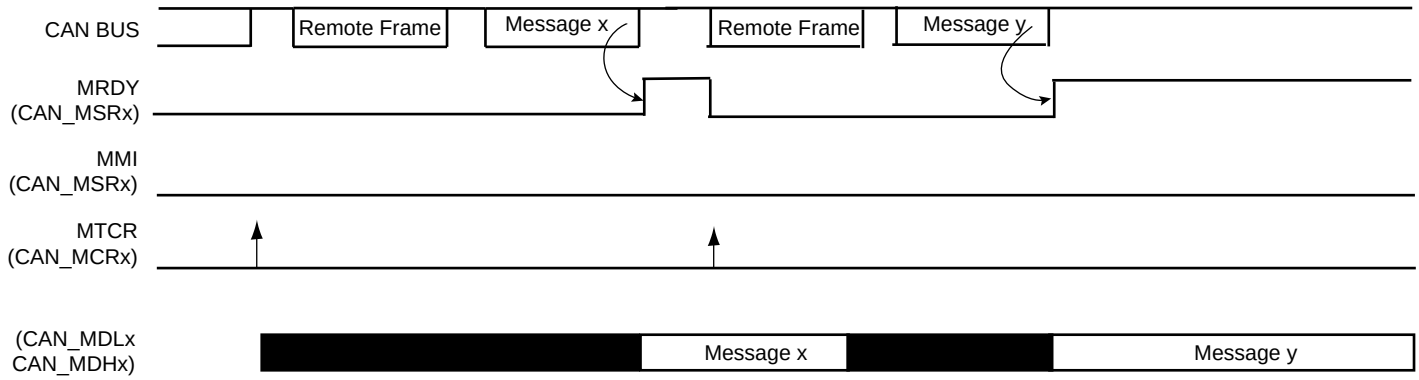
The MRTR bit in the CAN_MCRx register has no effect. This field is used only when using Transmit Mode.

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 40-18. Consumer Handling



40.8.4 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:

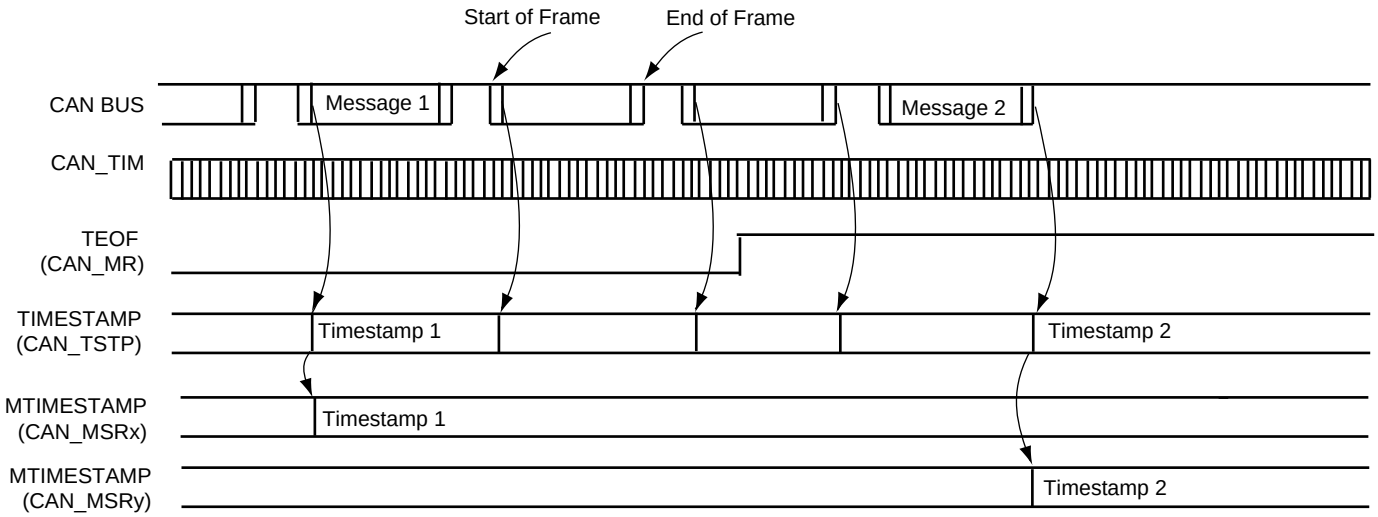
- Timestamping Mode: The value of the internal timer is captured at each Start Of Frame or each End Of Frame.
- Time Triggered Mode: The mailbox transfer operation is triggered when the internal timer reaches the mailbox trigger.

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.

40.8.4.1 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.

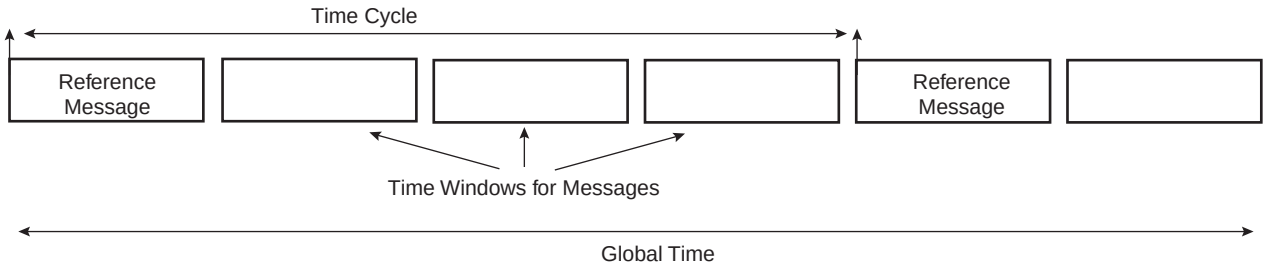
Figure 40-19. Mailbox Timestamp



40.8.4.2 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 40-20. 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.

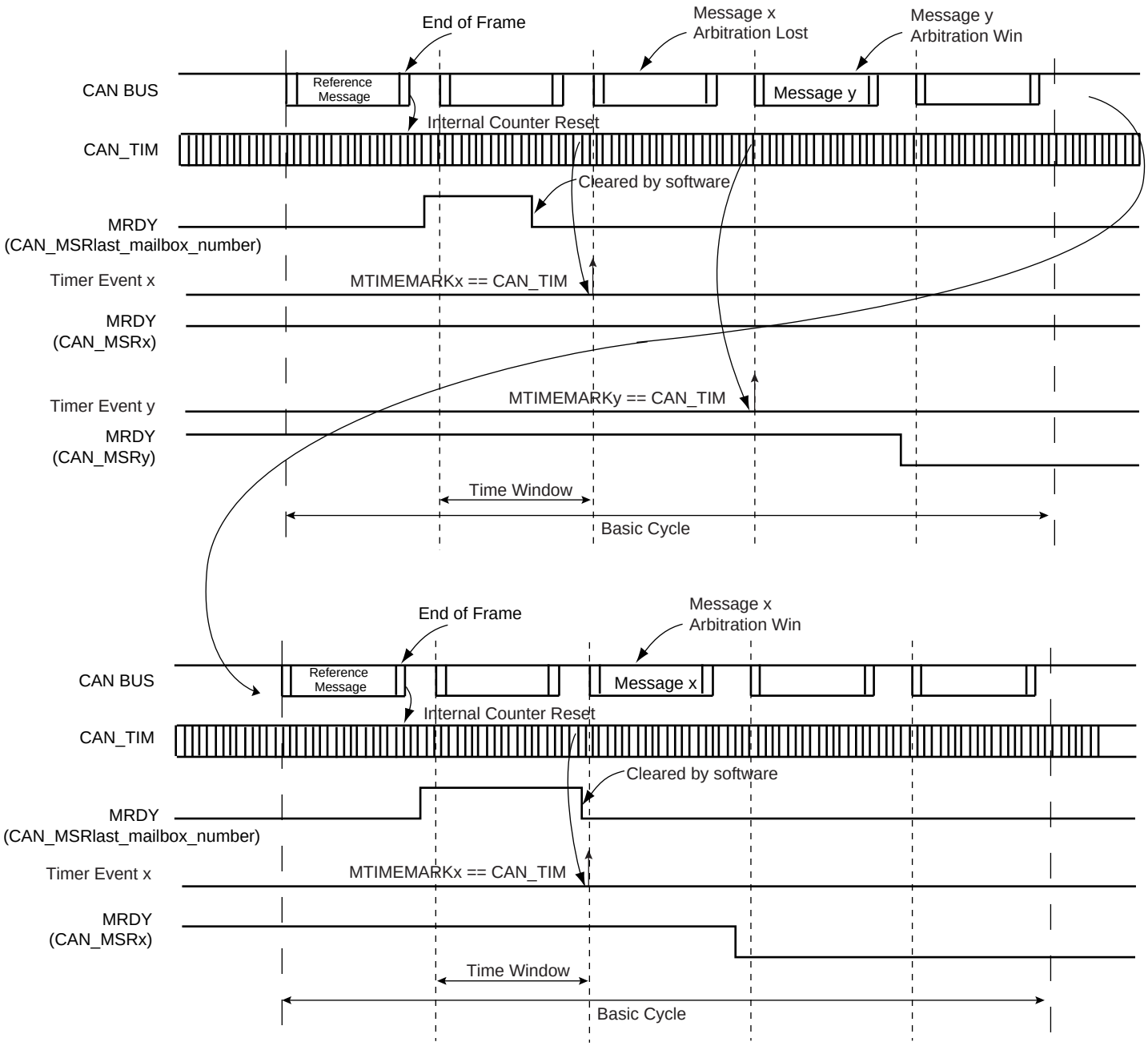
In Time Triggered Mode, transmit operations are delayed until the internal timer event for the mailbox. The application prepares a message to be sent by setting the MTCR in the CAN_MCRx register. The message is not sent until the CAN_TIM value is less than the MTIMEMARK value defined in the CAN_MMRx 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 40-21. Time Triggered Operations



40.8.5 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 WPVSRC 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).

List of the write-protected registers:

Section 40.9.1 “CAN Mode Register” on page 1212

Section 40.9.6 “CAN Baudrate Register” on page 1223

Section 40.9.14 “CAN Message Mode Register” on page 1231

Section 40.9.15 “CAN Message Acceptance Mask Register” on page 1232

Section 40.9.16 “CAN Message ID Register” on page 1233

40.9 Controller Area Network (CAN) User Interface

Table 40-6. 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_TIMESTP	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(3)	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	-

3. Mailbox number ranges from 0 to 7.

40.9.1 CAN Mode Register

Name: CAN_MR

Address: 0x400B4000 (0), 0x400B8000 (1)

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	RXSYNC		
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_TIMESTP register at each Start Of Frame.

1: The value of CAN_TIM is captured in the CAN_TIMESTP 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 1208.

- **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.

- **RXSYNC: Reception Synchronization Stage (not readable)**

This field allows configuration of the reception stage of the macrocell (for debug purposes only)

Value	Name	Description
0	DOUBLE_PP	Rx Signal with Double Synchro Stages (2 Positive Edges)
1	DOUBLE_PN	Rx Signal with Double Synchro Stages (One Positive Edge and One Negative Edge)
2	SINGLE_P	Rx Signal with Single Synchro Stage (Positive Edge)
3	NONE	Rx Signal with No Synchro Stage

40.9.2 CAN Interrupt Enable Register

Name: CAN_IER

Address: 0x400B4004 (0), 0x400B8004 (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: Wakeup Interrupt Enable**

0: No effect.

1: Enable SLEEP 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.

40.9.3 CAN Interrupt Disable Register

Name: CAN_IDR

Address: 0x400B4008 (0), 0x400B8008 (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 ERRP 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: Wakeup 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.

40.9.4 CAN Interrupt Mask Register

Name: CAN_IMR

Address: 0x400B400C (0), 0x400B800C (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: ERRP interrupt is disabled.

1: ERRP 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: Wakeup 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.

40.9.5 CAN Status Register

Name: CAN_SR

Address: 0x400B4010 (0), 0x400B8010 (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 is not in Error Active Mode.

1: CAN controller is in Error Active Mode.

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.

This flag is automatically reset when the above condition is not satisfied. Refer to Section “Error Interrupt Handler” on page 1196 for more information.

- **WARN: Warning Limit**

0: CAN controller Warning Limit is not reached.

1: CAN controller Warning Limit is reached.

This flag is set depending on TEC and REC counter values. It is set when at least one of the counter values exceeds 96.

This flag is automatically reset when above the condition is not satisfied. Refer to Section “Error Interrupt Handler” on page 1196 for more information.

- **ERRP: Error Passive Mode**

0: CAN controller is not in Error Passive Mode.

1: CAN controller is in Error Passive Mode.

This flag is set depending on TEC and REC counters values.

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).

This flag is automatically reset when the above condition is not satisfied. Refer to Section “Error Interrupt Handler” on page 1196 for more information.

- **BOFF: Bus Off Mode**

0: CAN controller is not in Bus Off Mode.

1: CAN controller is in Bus Off Mode.

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).

This flag is automatically reset when the above condition is not satisfied. Refer to Section "Error Interrupt Handler" on page 1196 for more information.

- **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.

40.9.6 CAN Baudrate Register

Name: CAN_BR

Address: 0x400B4014 (0), 0x400B8014 (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 40.7.4.1 "CAN Bit Timing Configuration" on page 1192.

- **PHASE2: Phase 2 segment**

This phase is used to compensate the edge phase error.

$$t_{PHS2} = t_{CSC} \times (PHASE2 + 1)$$

Warning: PHASE2 value must be different from 0.

- **PHASE1: Phase 1 segment**

This phase is used to compensate for edge phase error.

$$t_{PHS1} = t_{CSC} \times (PHASE1 + 1)$$

- **PROPAG: Programming time segment**

This part of the bit time is used to compensate for the physical delay times within the network.

$$t_{PRS} = t_{CSC} \times (PROPAG + 1)$$

- **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.

$$t_{SJW} = t_{CSC} \times (SJW + 1)$$

- **BRP: Baudrate Prescaler.**

This field allows user to program the period of the CAN system clock to determine the individual bit timing.

$$t_{CSC} = (BRP + 1) / MCK$$

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.

40.9.7 CAN Timer Register

Name: CAN_TIM

Address: 0x400B4018 (0), 0x400B8018 (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.

40.9.8 CAN Timestamp Register

Name: CAN_TIMESTP

Address: 0x400B401C (0), 0x400B801C (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_TIMESTP register is reset when the CAN is disabled then enabled thanks to the CANEN bit in the CAN_MR.

40.9.9 CAN Error Counter Register

Name: CAN_ECR

Address: 0x400B4020 (0), 0x400B8020 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
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 OVER-LOAD 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 OVER-LOAD 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.

40.9.10 CAN Transfer Command Register

Name: CAN_TCR

Address: 0x400B4024 (0), 0x400B8024 (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.

40.9.11 CAN Abort Command Register

Name: CAN_ACR

Address: 0x400B4028 (0), 0x400B8028 (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.

40.9.12 CAN Write Protection Mode Register

Name: CAN_WPMR

Address: 0x400B40E4 (0), 0x400B80E4 (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, ie 0x43414E in hexadecimal).

Protects the registers:

Section 40.9.1 “CAN Mode Register” on page 1212

Section 40.9.6 “CAN Baudrate Register” on page 1223

Section 40.9.14 “CAN Message Mode Register” on page 1231

Section 40.9.15 “CAN Message Acceptance Mask Register” on page 1232

Section 40.9.16 “CAN Message ID Register” on page 1233

40.9.13 CAN Write Protection Status Register

Name: CAN_WPSR

Address: 0x400B40E8 (0), 0x400B80E8 (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

40.9.14 CAN Message Mode Register

Name: CAN_MMRx [x=0..7]

Address: 0x400B4200 (0)[0], 0x400B4220 (0)[1], 0x400B4240 (0)[2], 0x400B4260 (0)[3], 0x400B4280 (0)[4], 0x400B42A0 (0)[5], 0x400B42C0 (0)[6], 0x400B42E0 (0)[7], 0x400B8200 (1)[0], 0x400B8220 (1)[1], 0x400B8240 (1)[2], 0x400B8260 (1)[3], 0x400B8280 (1)[4], 0x400B82A0 (1)[5], 0x400B82C0 (1)[6], 0x400B82E0 (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 1208.

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

40.9.15 CAN Message Acceptance Mask Register

Name: CAN_MAMx [x=0..7]

Address: 0x400B4204 (0)[0], 0x400B4224 (0)[1], 0x400B4244 (0)[2], 0x400B4264 (0)[3], 0x400B4284 (0)[4], 0x400B42A4 (0)[5], 0x400B42C4 (0)[6], 0x400B42E4 (0)[7], 0x400B8204 (1)[0], 0x400B8224 (1)[1], 0x400B8244 (1)[2], 0x400B8264 (1)[3], 0x400B8284 (1)[4], 0x400B82A4 (1)[5], 0x400B82C4 (1)[6], 0x400B82E4 (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.

40.9.16 CAN Message ID Register

Name: CAN_MIDx [x=0..7]

Address: 0x400B4208 (0)[0], 0x400B4228 (0)[1], 0x400B4248 (0)[2], 0x400B4268 (0)[3], 0x400B4288 (0)[4], 0x400B42A8 (0)[5], 0x400B42C8 (0)[6], 0x400B42E8 (0)[7], 0x400B8208 (1)[0], 0x400B8228 (1)[1], 0x400B8248 (1)[2], 0x400B8268 (1)[3], 0x400B8288 (1)[4], 0x400B82A8 (1)[5], 0x400B82C8 (1)[6], 0x400B82E8 (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**

40.9.17 CAN Message Family ID Register

Name: CAN_MFIDx [x=0..7]

Address: 0x400B420C (0)[0], 0x400B422C (0)[1], 0x400B424C (0)[2], 0x400B426C (0)[3], 0x400B428C (0)[4], 0x400B42AC (0)[5], 0x400B42CC (0)[6], 0x400B42EC (0)[7], 0x400B820C (1)[0], 0x400B822C (1)[1], 0x400B824C (1)[2], 0x400B826C (1)[3], 0x400B828C (1)[4], 0x400B82AC (1)[5], 0x400B82CC (1)[6], 0x400B82EC (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
```

40.9.18 CAN Message Status Register

Name: CAN_MSRx [x=0..7]

Address: 0x400B4210 (0)[0], 0x400B4230 (0)[1], 0x400B4250 (0)[2], 0x400B4270 (0)[3], 0x400B4290 (0)[4], 0x400B42B0 (0)[5], 0x400B42D0 (0)[6], 0x400B42F0 (0)[7], 0x400B8210 (1)[0], 0x400B8230 (1)[1], 0x400B8250 (1)[2], 0x400B8270 (1)[3], 0x400B8290 (1)[4], 0x400B82B0 (1)[5], 0x400B82D0 (1)[6], 0x400B82F0 (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.

40.9.19 CAN Message Data Low Register

Name: CAN_MDLx [x=0..7]

Address: 0x400B4214 (0)[0], 0x400B4234 (0)[1], 0x400B4254 (0)[2], 0x400B4274 (0)[3], 0x400B4294 (0)[4], 0x400B42B4 (0)[5], 0x400B42D4 (0)[6], 0x400B42F4 (0)[7], 0x400B8214 (1)[0], 0x400B8234 (1)[1], 0x400B8254 (1)[2], 0x400B8274 (1)[3], 0x400B8294 (1)[4], 0x400B82B4 (1)[5], 0x400B82D4 (1)[6], 0x400B82F4 (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]

40.9.20 CAN Message Data High Register

Name: CAN_MDHx [x=0..7]

Address: 0x400B4218 (0)[0], 0x400B4238 (0)[1], 0x400B4258 (0)[2], 0x400B4278 (0)[3], 0x400B4298 (0)[4], 0x400B42B8 (0)[5], 0x400B42D8 (0)[6], 0x400B42F8 (0)[7], 0x400B8218 (1)[0], 0x400B8238 (1)[1], 0x400B8258 (1)[2], 0x400B8278 (1)[3], 0x400B8298 (1)[4], 0x400B82B8 (1)[5], 0x400B82D8 (1)[6], 0x400B82F8 (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]

40.9.21 CAN Message Control Register

Name: CAN_MCRx [x=0..7]

Address: 0x400B421C (0)[0], 0x400B423C (0)[1], 0x400B425C (0)[2], 0x400B427C (0)[3], 0x400B429C (0)[4], 0x400B42BC (0)[5], 0x400B42DC (0)[6], 0x400B42FC (0)[7], 0x400B821C (1)[0], 0x400B823C (1)[1], 0x400B825C (1)[2], 0x400B827C (1)[3], 0x400B829C (1)[4], 0x400B82BC (1)[5], 0x400B82DC (1)[6], 0x400B82FC (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.

41. Ethernet MAC 10/100 (EMAC)

41.1 Description

The EMAC module implements a 10/100 Ethernet MAC compatible with the IEEE 802.3 standard using an address checker, statistics and control registers, receive and transmit blocks, and a DMA interface.

The address checker recognizes four specific 48-bit addresses and contains a 64-bit hash register for matching multicast and unicast addresses. It can recognize the broadcast address of all ones, copy all frames, and act on an external address match signal.

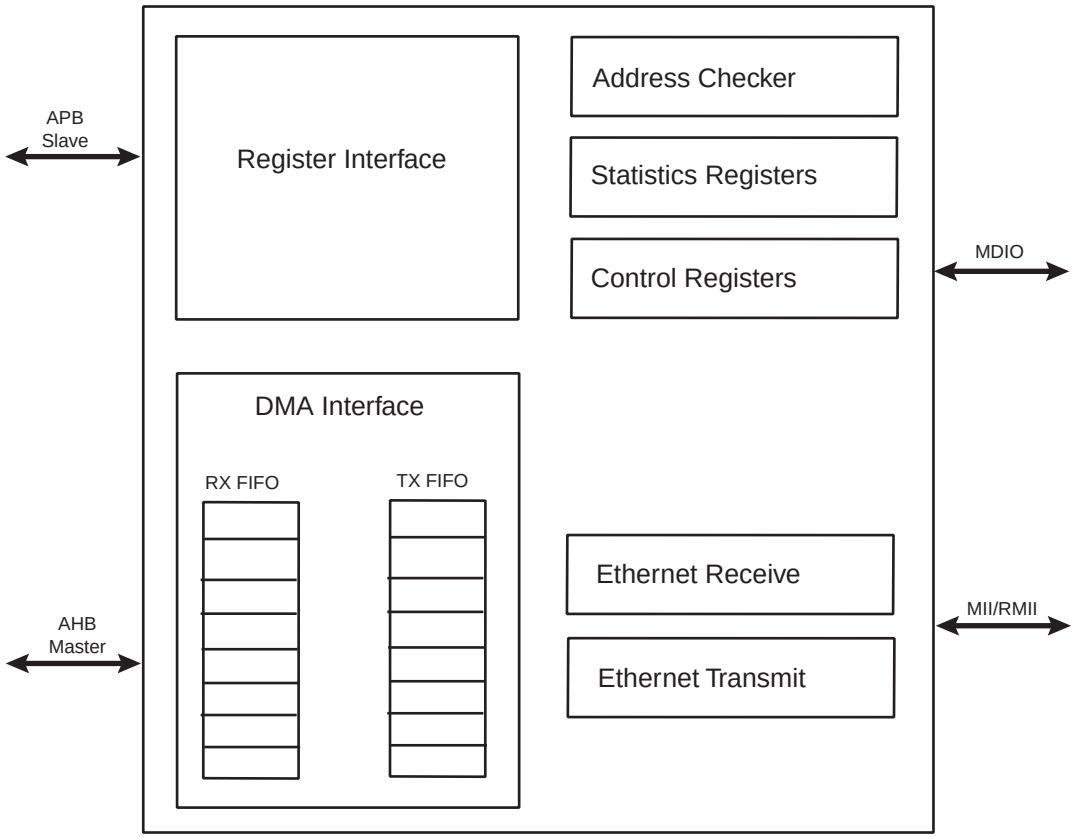
41.2 Embedded Characteristics

- EMAC0 supports MII interface to the physical layer, EMAC1 supports RMII interface to the physical layer
- Compatible with IEEE Standard 802.3
- 10 and 100 Mbit/s Operation
- Full- and Half-duplex Operation
- Statistics Counter Registers
- Interrupt Generation to Signal Receive and Transmit Completion
- DMA Master on Receive and Transmit Channels
- Transmit and Receive FIFOs
- Automatic Pad and CRC Generation on Transmitted Frames
- Automatic Discard of Frames Received with Errors
- Address Checking Logic Supports Up to Four Specific 48-bit Addresses
- Supports Promiscuous Mode Where All Valid Received Frames are Copied to Memory
- Hash Matching of Unicast and Multicast Destination Addresses
- Physical Layer Management through MDIO Interface
- Half-duplex Flow Control by Forcing Collisions on Incoming Frames
- Full-duplex Flow Control with Recognition of Incoming Pause Frames
- Support for 802.1Q VLAN Tagging with Recognition of Incoming VLAN and Priority Tagged Frames
- Multiple Buffers per Receive and Transmit Frame
- Jumbo Frames Up to 10240 bytes Supported

The statistics register block contains registers for counting various types of event associated with transmit and receive operations. These registers, along with the status words stored in the receive buffer list, enable software to generate network management statistics compatible with IEEE 802.3.

41.3 Block Diagram

Figure 41-1. EMAC Block Diagram



41.4 Functional Description

The MACB has several clock domains:

- System bus clock (AHB and APB): DMA and register blocks
- Transmit clock: transmit block
- Receive clock: receive and address checker block

The system bus clock must run at least as fast as the receive clock and transmit clock (25 MHz at 100 Mbps, and 2.5 MHZ at 10 Mbps).

Figure 41-1 illustrates the different blocks of the EMAC module.

The control registers drive the MDIO interface, setup up DMA activity, start frame transmission and select modes of operation such as full- or half-duplex.

The receive block checks for valid preamble, FCS, alignment and length, and presents received frames to the address checking block and DMA interface.

The transmit block takes data from the DMA interface, adds preamble and, if necessary, pad and FCS, and transmits data according to the CSMA/CD (carrier sense multiple access with collision detect) protocol. The start of transmission is deferred if CRS (carrier sense) is active.

If COL (collision) becomes active during transmission, a jam sequence is asserted and the transmission is retried after a random back off. CRS and COL have no effect in full duplex mode.

The DMA block connects to external memory through its AHB bus interface. It contains receive and transmit FIFOs for buffering frame data. It loads the transmit FIFO and empties the receive FIFO using AHB bus master operations. Receive data is not sent to memory until the address checking logic has determined that the frame should be copied. Receive or transmit frames are stored in one or more buffers. Receive buffers have a fixed length of 128 bytes. Transmit buffers range in length between 0 and 2047 bytes, and up to 128 buffers are permitted per frame. The DMA block manages the transmit and receive framebuffer queues. These queues can hold multiple frames.

41.4.1 Clock

Synchronization module in the EMAC requires that the bus clock (hclk) runs at the speed of the macb_tx/rx_clk at least, which is 25 MHz at 100 Mbps, and 2.5 MHz at 10 Mbps.

41.4.2 Memory Interface

Frame data is transferred to and from the EMAC through the DMA interface. All transfers are 32-bit words and may be single accesses or bursts of 2, 3 or 4 words. Burst accesses do not cross sixteen-byte boundaries. Bursts of 4 words are the default data transfer; single accesses or bursts of less than four words may be used to transfer data at the beginning or the end of a buffer.

The DMA controller performs six types of operation on the bus. In order of priority, these are:

1. Receive buffer manager write
2. Receive buffer manager read
3. Transmit data DMA read
4. Receive data DMA write
5. Transmit buffer manager read
6. Transmit buffer manager write

41.4.2.1 FIFO

The FIFO depths are 128 bytes for receive and 128 bytes for transmit and are a function of the system clock speed, memory latency and network speed.

Data is typically transferred into and out of the FIFOs in bursts of four words. For receive, a bus request is asserted when the FIFO contains four words and has space for 28 more. For transmit, a bus request is generated when there is space for four words, or when there is space for 27 words if the next transfer is to be only one or two words.

Thus the bus latency must be less than the time it takes to load the FIFO and transmit or receive three words (112 bytes) of data.

At 100 Mbit/s, it takes 8960 ns to transmit or receive 112 bytes of data. In addition, six master clock cycles should be allowed for data to be loaded from the bus and to propagate through the FIFOs. For a 133 MHz master clock this takes 45 ns, making the bus latency requirement 8915 ns.

41.4.2.2 Receive Buffers

Received frames, including CRC/FCS optionally, are written to receive buffers stored in memory. Each receive buffer is 128 bytes long. The start location for each receive buffer is stored in memory in a list of receive buffer descriptors at a location pointed to by the receive buffer queue pointer register. The receive buffer start location is a word address. For the first buffer of a frame, the start location 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 buffer is offset the available length of the first buffer of a frame is reduced by the corresponding number of bytes.

Each list entry consists of two words, the first being the address of the receive buffer and the second being the receive status. If the length of a receive frame exceeds the buffer length, the status word for the used buffer is written with zeroes except for the “start of frame” bit and the offset bits, if appropriate. Bit zero of the address field is written to one to show the buffer has been used. The receive buffer manager then reads the location of the next receive buffer and fills that with receive frame data. The final buffer descriptor status word contains the complete frame status. Refer to Table 41-1 for details of the receive buffer descriptor list.

Table 41-1. Receive Buffer Descriptor Entry

Bit	Function
Word 0	
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 EMAC to write data to the receive buffer. The EMAC 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.
Word 1	
31	Global all ones broadcast address detected
30	Multicast hash match
29	Unicast hash match
28	External address match
27	Reserved for future use
26	Specific address register 1 match
25	Specific address register 2 match
24	Specific address register 3 match
23	Specific address register 4 match
22	Type ID match
21	VLAN tag detected (i.e., type id of 0x8100)
20	Priority tag detected (i.e., type id of 0x8100 and null VLAN identifier)

Table 41-1. Receive Buffer Descriptor Entry (Continued)

Bit	Function
19:17	VLAN priority (only valid if bit 21 is set)
16	Concatenation 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 other valid status are bits 12, 13 and 14.
14	Start of frame - when set the buffer contains the start of a frame. If both bits 15 and 14 are set, then the buffer contains a whole frame.
13:12	Receive buffer offset - indicates the number of bytes by which the data in the first buffer is offset from the word address. Updated with the current values of the network configuration register. If jumbo frame mode is enabled through bit 3 of the network configuration register, then bits 13:12 of the receive buffer descriptor entry are used to indicate bits 13:12 of the frame length.
11:0	Length of frame including FCS (if selected). Bits 13:12 are also used if jumbo frame mode is selected.

To receive frames, the buffer descriptors must be initialized by writing an appropriate address to bits 31 to 2 in the first word of each list entry. Bit zero must be written with zero. Bit one is the wrap bit and indicates the last entry in the list.

The start location of the receive buffer descriptor list must be written to the receive buffer queue pointer register before setting the receive enable bit in the network control register to enable receive. As soon as the receive block starts writing received frame data to the receive FIFO, the receive buffer manager reads the first receive buffer location pointed to by the receive buffer queue pointer register.

If the filter block then indicates that the 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. If the current buffer pointer has its wrap bit set or is the 1024th descriptor, the next receive buffer location is read from the beginning of the receive descriptor list. Otherwise, the next receive buffer location is read from the next word in memory.

There is an 11-bit counter to count out the 2048 word locations of a maximum length, receive buffer descriptor list. This is added with the value originally written to the receive buffer queue pointer register to produce a pointer into the list. A read of the receive buffer queue pointer register returns the pointer value, which is the queue entry currently being accessed. The counter is reset after receive status is written to a descriptor that has its wrap bit set or rolls over to zero after 1024 descriptors have been accessed. The value written to the receive buffer pointer register may be any word-aligned address, provided that there are at least 2048 word locations available between the pointer and the top of the memory.

Section 3.6 of the AMBA 2.0 specification states that bursts should not cross 1K boundaries. As receive buffer manager writes are bursts of two words, to ensure that this does not occur, it is best to write the pointer register with the least three significant bits set to zero. As receive buffers are used, the receive buffer manager sets bit zero of the first word of the descriptor to indicate *used*. If a receive error is detected the receive buffer currently being written is recovered. Previous buffers are not recovered. Software should search through the *used* bits in the buffer descriptors to find out how many frames have been received. It should be checking the start-of-frame and end-of-frame bits, and not rely on the value returned by the receive buffer queue pointer register which changes continuously as more buffers are used.

For CRC errored frames, excessive length frames or length field mismatched frames, all of which are counted in the statistics registers, it is possible that a frame fragment might be stored in a sequence of receive buffers. Software can detect this by looking for start of frame bit set in a buffer following a buffer with no end of frame bit set.

For a properly working Ethernet system, there should be no excessively long frames or frames greater than 128 bytes with CRC/FCS errors. Collision fragments are less than 128 bytes long. Therefore, it is a rare occurrence to find a frame fragment in a receive buffer.

If bit zero is set when the receive buffer manager reads the location of the receive buffer, then the buffer has already been used and cannot be used again until software has processed the frame and cleared bit zero. In this case, the DMA block sets the buffer not available bit in the receive status register and triggers an interrupt.

If bit zero is set when the receive buffer manager reads the location of the receive buffer and a frame is being received, the frame is discarded and the receive resource error statistics register is incremented.

A receive overrun condition occurs when bus was not granted in time or because HRESP was not OK (bus error). In a receive overrun condition, the receive overrun interrupt is asserted and the buffer currently being written is recovered. The next frame received with an address that is recognized reuses the buffer.

If bit 17 of the network configuration register is set, the FCS of received frames shall not be copied to memory. The frame length indicated in the receive status field shall be reduced by four bytes in this case.

41.4.2.3 Transmit Buffer

Frames to be transmitted are stored in one or more transmit buffers. Transmit buffers can be between 0 and 2047 bytes long, so it is possible to transmit frames longer than the maximum length specified in IEEE Standard 802.3. Zero length buffers are allowed. The maximum number of buffers permitted for each transmit frame is 128.

The start location for each transmit buffer is stored in memory in a list of transmit buffer descriptors at a location pointed to by the transmit buffer queue pointer register. Each list entry consists of two words, the first being the byte address of the transmit buffer and the second containing the transmit control and status. Frames can be transmitted with or without automatic CRC generation. If CRC is automatically generated, pad is also automatically generated to take frames to a minimum length of 64 bytes. Table 41-2 on page 1248 defines an entry in the transmit buffer descriptor list. To transmit frames, the buffer descriptors must be initialized by writing an appropriate byte address to bits 31 to 0 in the first word of each list entry. The second transmit buffer descriptor is initialized with control information that indicates the length of the buffer, whether or not it is to be transmitted with CRC and whether the buffer is the last buffer in the frame.

After transmission, the control bits are written back to the second word of the first buffer along with the “used” bit and other status information. Bit 31 is the “used” bit which must be zero when the control word is read if transmission is to happen. It is written to one when a frame has been transmitted. Bits 27, 28 and 29 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 after 1024 descriptors, the queue pointer rolls over to the start in a similar fashion to the receive queue.

The transmit buffer queue pointer register must not be written while transmit is active. If a new value is written to the transmit buffer queue pointer register, the queue pointer resets itself to point to the beginning of the new queue. If transmit is disabled by writing to bit 3 of the network control, the transmit buffer queue pointer register resets to point to the beginning of the transmit queue. Note that disabling receive does not have the same effect on the receive queue pointer.

Once the transmit queue is initialized, transmit is activated by writing to bit 9, the *Transmit Start* bit of the network control register. Transmit is halted when a buffer descriptor with its *used* bit set is read, or if 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.

Transmission control is implemented with a Tx_go variable which is readable in the transmit status register at bit location 3. The Tx_go variable is reset when:

- transmit is disabled
- a buffer descriptor with its ownership bit set is read
- a new value is written to the transmit buffer queue pointer register
- bit 10, tx_halt, of the network control register is written
- there is a transmit error such as too many retries or a transmit underrun.

To set tx_go, write to bit 9, tx_start, of the network control register. Transmit halt does not take effect until any ongoing transmit finishes. If a collision occurs during transmission of a multi-buffer frame, transmission automatically restarts from the first buffer of the frame. If a “used” bit is read midway through transmission of a multi-buffer frame, this is treated as a transmit error. Transmission stops, tx_er is asserted and the FCS is bad.

If transmission stops due to a transmit error, the transmit queue pointer resets to point to the beginning of the transmit queue. Software needs to re-initialize the transmit queue after a transmit error.

If transmission stops due to a “used” bit being read at the start of the frame, the transmission queue pointer is not reset and transmit starts from the same transmit buffer descriptor when the transmit start bit is written

Table 41-2. Transmit Buffer Descriptor Entry

Bit	Function
Word 0	
31:0	**Byte Address of buffer**
Word 1	
31	Used. Needs to be zero for the EMAC to read data from the transmit buffer. The EMAC sets this to one for the first buffer of a frame once it has been successfully transmitted. Software has to clear this bit before the buffer can be used again. Note: This bit is only set for the first buffer in a frame unlike receive where all buffers have the Used bit set once used.
30	Wrap. Marks last descriptor in transmit buffer descriptor list.
29	Retry limit exceeded, transmit error detected
28	Transmit underrun, occurs either when hresp is not OK (bus error) or the transmit data could not be fetched in time or when buffers are exhausted in mid frame.
27	Buffers exhausted in mid frame
26:17	Reserved
16	No CRC. When set, no CRC is appended to the current frame. This bit only needs to be set for the last buffer of a frame.
15	Last buffer. When set, this bit indicates the last buffer in the current frame has been reached.
14:11	Reserved
10:0	Length of buffer

41.4.3 Transmit Block

This block transmits frames in accordance with the Ethernet IEEE 802.3 CSMA/CD protocol. Frame assembly starts by adding preamble and the start frame delimiter. Data is taken from the transmit FIFO a word at a time. Data is transmitted least significant nibble first. If necessary, padding is added to increase the frame length to 60 bytes. CRC is calculated as a 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.

In full-duplex mode, 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, it waits for it to de-assert and then starts transmission after the interframe gap of 96 bit times. If the collision signal is asserted during transmission, the transmitter transmits a jam sequence of 32 bits taken from the data register and then retry transmission after the back off time has elapsed.

The back-off time is based on an XOR of the 10 least significant bits of the data coming from the transmit 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, after the second 2, and so on up to 10. Above 10, all 10 bits are used. An error is indicated and no further attempts are made if 16 attempts cause collisions.

If transmit DMA underruns, bad CRC is automatically appended using the same mechanism as jam insertion and the tx_er signal is asserted. For a properly configured system, this should never happen.

If the back pressure bit is set in the network control register in 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.

41.4.4 Pause Frame Support

The start of an 802.3 pause frame is as follows:

Table 41-3. 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 network configuration register contains a receive pause enable bit (13). If a valid pause frame is received, the pause time register is updated with the frame's pause time, regardless of its current contents and regardless of the state of the configuration register bit 13. An interrupt (12) is triggered when a pause frame is received, assuming it is enabled in the interrupt mask register. If bit 13 is set in the network configuration register and the value of the pause time register is non-zero, no new frame is transmitted until the pause time register has decremented to zero.

The loading of a new pause time, and hence the pausing of transmission, only occurs when the EMAC is configured for full-duplex operation. If the EMAC is configured for half-duplex, there is no transmission pause, but the pause frame received interrupt is still triggered.

A valid pause frame is defined as having a destination address that matches either the address stored in specific address register 1 or matches 0x0180C2000001 and has the MAC control frame type ID of 0x8808 and the pause opcode of 0x0001. Pause frames that have FCS or other errors are treated as invalid and are discarded. Valid pause frames received increment the Pause Frame Received statistic register.

The pause time register decrements every 512 bit times (i.e., 128 `rx_clks` in nibble mode) once transmission has stopped. For test purposes, the register decrements every `rx_clk` cycle once transmission has stopped if bit 12 (retry test) is set in the network configuration register. If the pause enable bit (13) is not set in the network configuration register, then the decrementing occurs regardless of whether transmission has stopped or not.

An interrupt (13) is asserted whenever the pause time register decrements to zero (assuming it is enabled in the interrupt mask register).

41.4.5 Receive Block

The receive block checks for valid preamble, FCS, alignment and length, presents received frames to the DMA block and stores the frames destination address for use by the address checking block. If, during frame reception, the frame is found to be too long or rx_er is asserted, a bad frame indication is sent to the DMA block. The DMA block then ceases sending data to memory. At the end of frame reception, the receive block indicates to the DMA block whether the frame is good or bad. The DMA block recovers the current receive buffer if the frame was bad. The receive block signals the register block to increment the alignment error, the CRC (FCS) error, the short frame, long frame, jabber error, the receive symbol error statistics and the length field mismatch statistics.

The enable bit for jumbo frames in the network configuration register allows the EMAC to receive jumbo frames of up to 10240 bytes in size. This operation does not form part of the IEEE802.3 specification and is disabled by default. When jumbo frames are enabled, frames received with a frame size greater than 10240 bytes are discarded.

41.4.6 Address Checking Block

The address checking (or filter) block indicates to the DMA block which receive frames should be copied to memory. Whether a frame is copied depends on what is enabled in the network configuration register, the state of the external match pin, the contents of the specific address and hash registers and the frame's destination address. In this implementation of the EMAC, the frame's source address is not checked. Provided that bit 18 of the Network Configuration register is not set, a frame is not copied to memory if the EMAC is transmitting in half duplex mode at the time a destination address is received. If bit 18 of the Network Configuration register is set, frames can be received while transmitting in half-duplex mode.

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, the LSB of the first byte of the frame, is the group/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 EMAC 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 copied to memory.

The following example illustrates the use of the address match registers for a MAC address of 21:43:65:87:A9:CB.

Preamble 55

SFD D5

DA (Octet0 - LSB) 21

DA(Octet 1) 43

DA(Octet 2) 65

DA(Octet 3) 87

DA(Octet 4) A9

DA (Octet5 - MSB) CB

SA (LSB) 00

SA 00

SA 00

SA 00

SA 00

SA (MSB) 43

SA (LSB) 21

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:

- Base address + 0x98 0x87654321 (Bottom)
- Base address + 0x9C 0x0000CBA9 (Top)

And for a successful match to the Type ID register, the following should be set up:

- Base address + 0xB8 0x00004321

41.4.7 Broadcast Address

The broadcast address of 0xFFFFFFFFFFFF is recognized if the ‘no broadcast’ bit in the network configuration register is zero.

41.4.8 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 *exclusive or* of every sixth bit of the destination address.

hash_index[5] = da[5] ^ da[11] ^ da[17] ^ da[23] ^ da[29] ^ da[35] ^ da[41] ^ da[47]

hash_index[4] = da[4] ^ da[10] ^ da[16] ^ da[22] ^ da[28] ^ da[34] ^ da[40] ^ da[46]

hash_index[3] = da[3] ^ da[09] ^ da[15] ^ da[21] ^ da[27] ^ da[33] ^ da[39] ^ da[45]

hash_index[2] = da[2] ^ da[08] ^ da[14] ^ da[20] ^ da[26] ^ da[32] ^ da[38] ^ da[44]

hash_index[1] = da[1] ^ da[07] ^ da[13] ^ da[19] ^ da[25] ^ da[31] ^ da[37] ^ da[43]

hash_index[0] = da[0] ^ 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 is matched according to whether the frame is multicast or unicast.

A multicast match is signalled if the multicast hash enable bit is set. da[0] is 1 and the hash index points to a bit set in the hash register.

A unicast match is signalled if the unicast hash enable bit is set. da[0] is 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.

41.4.9 Copy All Frames (or Promiscuous Mode)

If the copy all frames bit is set in the network configuration register, then all non-errored frames are copied to memory. For example, frames that are too long, too short, or have FCS errors or rx_er asserted during reception are discarded and all others are received. Frames with FCS errors are copied to memory if bit 19 in the network configuration register is set.

41.4.10 Type ID Checking

The contents of the type_id register are compared against the length/type ID of received frames (i.e., bytes 13 and 14). Bit 22 in the receive buffer descriptor status is set if there is a match. The reset state of this register is zero which is unlikely to match the length/type ID of any valid Ethernet frame.

Note: A type ID match does not affect whether a frame is copied to memory.

41.4.11 VLAN Support

An Ethernet encoded 802.1Q VLAN tag looks like this:

Table 41-4. 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. If the VID (VLAN identifier) is null (0x000), this indicates a priority-tagged frame. The MAC can support frame lengths up to 1536 bytes, 18 bytes more than the original Ethernet maximum frame length of 1518 bytes. This is achieved by setting bit 8 in the network configuration register.

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 is 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

41.4.12 PHY Maintenance

The register EMAC_MAN enables the EMAC to communicate with a PHY by means of the MDIO interface. It is used during auto-negotiation to ensure that the EMAC and the PHY are configured for the same speed and duplex configuration.

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 bit ten is set to zero, and bit eleven is set to one 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 MDC cycle. This causes transmission of a PHY management frame on MDIO.

Reading during the shift operation returns the current contents of the shift register. At the end of management operation, the bits 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 MDIO interface can read IEEE 802.3 clause 45 PHYs as well as clause 22 PHYs. To read clause 45 PHYs, bits[31:28] should be written as 0x0011. For a description of MDC generation, see the network configuration register in the “Network Control Register” on page 1259.

41.4.13 Physical Interface

Depending on products, the Ethernet MAC is capable of interfacing to RMII or MII Interface. The RMII bit in the EMAC_USRIO register controls the interface that is selected. When this bit is set, the RMII interface is selected, else the MII interface is selected.

The MII and RMII interfaces are capable of both 10 Mb/s and 100 Mb/s data rates as described in the IEEE 802.3u standard. The signals used by the MII and RMII interfaces are described in Table 41-5.

Table 41-5. Pin Configuration, Depending on Products

Pin Name	MII	RMII
ETXCK_EREFCK	ETXCK: Transmit Clock	EREFCK: Reference Clock
ECRS	ECRS: Carrier Sense	

Table 41-5. Pin Configuration, Depending on Products

ECOL	ECOL: Collision Detect	
ERXDV	ERXDV: Data Valid	ECRSDV: Carrier Sense/Data Valid
ERX0 - ERX3	ERX0 - ERX3: 4-bit Receive Data	ERX0 - ERX1: 2-bit Receive Data
ERXER	ERXER: Receive Error	ERXER: Receive Error
ERXCK	ERXCK: Receive Clock	
ETXEN	ETXEN: Transmit Enable	ETXEN: Transmit Enable
ETX0-ETX3	ETX0 - ETX3: 4-bit Transmit Data	ETX0 - ETX1: 2-bit Transmit Data
ETXER	ETXER: Transmit Error	

The intent of the RMII is to provide a reduced pin count alternative to the IEEE 802.3u MII. It uses 2 bits for transmit (ETX0 and ETX1) and two bits for receive (ERX0 and ERX1). There is a Transmit Enable (ETXEN), a Receive Error (ERXER), a Carrier Sense (ECRS_DV), and a 50 MHz Reference Clock (ETXCK_EREFCK) for 100Mb/s data rate.

41.4.13.1RMII Transmit and Receive Operation

The same signals are used internally for both the RMII and the MII operations. The RMII maps these signals in a more pin-efficient manner. The transmit and receive bits are converted from a 4-bit parallel format to a 2-bit parallel scheme that is clocked at twice the rate. The carrier sense and data valid signals are combined into the ECRSDV signal. This signal contains information on carrier sense, FIFO status, and validity of the data. Transmit error bit (ETXER) and collision detect (ECOL) are not used in RMII mode.

41.5 Programming Interface

41.5.1 Initialization

41.5.1.1 Configuration

Initialization of the EMAC 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:

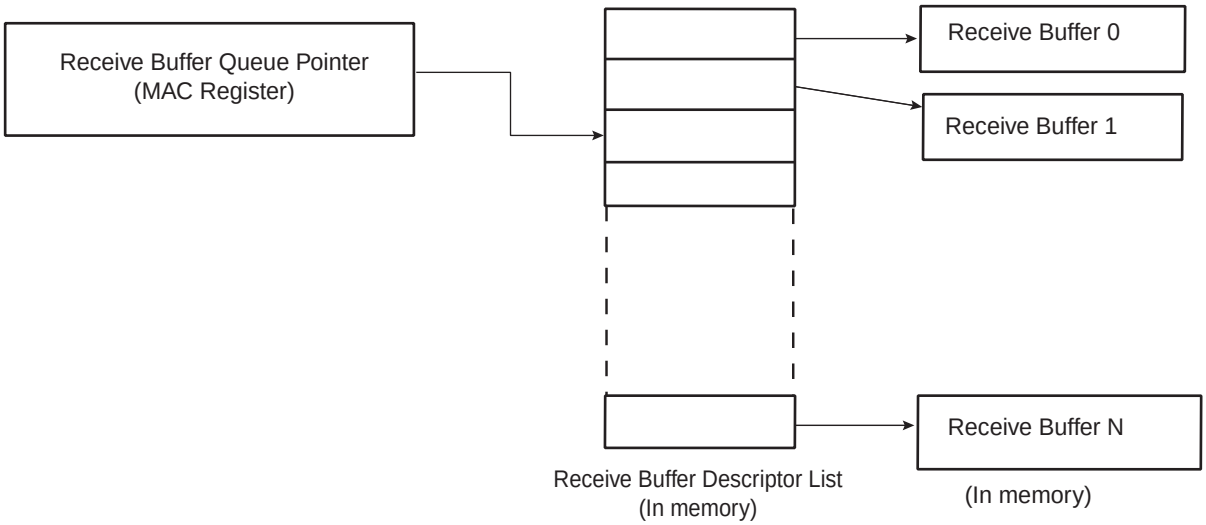
1. Write to network control register to disable transmit and receive circuits.
2. Write to network control register to change loop-back mode.
3. Write to network control register to re-enable transmit or receive circuits.

Note: These writes to network control register cannot be combined in any way.

41.5.1.2 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 “Receive Buffer Descriptor Entry” on page 1245. It points to this data structure.

Figure 41-2. Receive Buffer List



To create the list of buffers:

1. Allocate a number (*n*) of buffers of 128 bytes in system memory.
2. Allocate an area 2*n* words for the receive buffer descriptor entry in system memory and create *n* entries in this list. Mark all entries in this list as owned by EMAC, i.e., bit 0 of word 0 set to 0.
3. If less than 1024 buffers are defined, the last descriptor must be marked with the wrap bit (bit 1 in word 0 set to 1).
4. Write address of receive buffer descriptor entry to EMAC register `receive_buffer` queue pointer.
5. The receive circuits can then be enabled by writing to the address recognition registers and then to the network control register.

41.5.1.3 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 41-2 on page 1248) that 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 2*n* words for the transmit buffer descriptor entry in system memory and create N entries in this list. Mark all entries in this list as owned by EMAC, i.e. bit 31 of word 1 set to 0.
3. If fewer than 1024 buffers are defined, the last descriptor must be marked with the wrap bit — bit 30 in word 1 set to 1.
4. Write address of transmit buffer descriptor entry to EMAC register transmit_buffer queue pointer.
5. The transmit circuits can then be enabled by writing to the network control register.

41.5.1.4 Address Matching

The EMAC register-pair hash address and the four specific address register-pairs must be written with the required values. Each register-pair comprises 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. See “Address Checking Block” on page 1250. for details of address matching. Each register-pair may be written at any time, regardless of whether the receive circuits are enabled or disabled.

41.5.1.5 Interrupts

There are 14 interrupt conditions that are detected within the EMAC. 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 Interrupt Controller). To ascertain which interrupt has been generated, read the interrupt status register. Note that this register clears itself when read. 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.

41.5.1.6 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 as long as they conclude on byte borders.
3. Set-up the transmit buffer list.
4. Set the network control register to enable transmission and enable interrupts.
5. Write data for transmission into these buffers.
6. Write the address to transmit buffer descriptor queue pointer.
7. Write control and length to word one of the transmit buffer descriptor entry.
8. Write to the transmit start bit in the network control register.

41.5.1.7 Receiving Frames

When a frame is received and the receive circuits are enabled, the EMAC 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 the hash address function.
- if it is a broadcast address (0xFFFFFFFFFFFF) and broadcasts are allowed.
- if the EMAC is configured to copy all frames.

The register receive buffer queue pointer points to the next entry (see Table 41-1 on page 1245) and the EMAC 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 EMAC then updates the receive buffer descriptor entry with the reason for the address match and marks the area as being owned by software. Once this is complete an interrupt receive complete is set. Software is then responsible for handling the data in the buffer and then releasing the buffer by writing the ownership bit back to 0.

If the EMAC is unable to write the data at a rate to match the incoming frame, then an interrupt receive overrun is set. If there is no receive buffer available, i.e., the next buffer is still owned by software, the interrupt receive buffer not available is set. If the frame is not successfully received, a statistic register is incremented and the frame is discarded without informing software.

41.6 Ethernet MAC 10/100 (EMAC) User Interface

Table 41-6. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Network Control Register	EMAC_NCR	Read-write	0
0x04	Network Configuration Register	EMAC_NCFGR	Read-write	0x800
0x08	Network Status Register	EMAC_NSR	Read-only	-
0x0C	Reserved			
0x10	Reserved			
0x14	Transmit Status Register	EMAC_TSR	Read-write	0x0000_0000
0x18	Receive Buffer Queue Pointer Register	EMAC_RBQP	Read-write	0x0000_0000
0x1C	Transmit Buffer Queue Pointer Register	EMAC_TBQP	Read-write	0x0000_0000
0x20	Receive Status Register	EMAC_RSR	Read-write	0x0000_0000
0x24	Interrupt Status Register	EMAC_ISR	Read-write	0x0000_0000
0x28	Interrupt Enable Register	EMAC_IER	Write-only	-
0x2C	Interrupt Disable Register	EMAC_IDR	Write-only	-
0x30	Interrupt Mask Register	EMAC_IMR	Read-only	0x0000_3FFF
0x34	Phy Maintenance Register	EMAC_MAN	Read-write	0x0000_0000
0x38	Pause Time Register	EMAC_PTR	Read-write	0x0000_0000
0x3C	Pause Frames Received Register	EMAC_PFR	Read-write	0x0000_0000
0x40	Frames Transmitted Ok Register	EMAC_FTO	Read-write	0x0000_0000
0x44	Single Collision Frames Register	EMAC_SCF	Read-write	0x0000_0000
0x48	Multiple Collision Frames Register	EMAC_MCF	Read-write	0x0000_0000
0x4C	Frames Received Ok Register	EMAC_FRO	Read-write	0x0000_0000
0x50	Frame Check Sequence Errors Register	EMAC_FCSE	Read-write	0x0000_0000
0x54	Alignment Errors Register	EMAC_ALE	Read-write	0x0000_0000
0x58	Deferred Transmission Frames Register	EMAC_DTF	Read-write	0x0000_0000
0x5C	Late Collisions Register	EMAC_LCOL	Read-write	0x0000_0000
0x60	Excessive Collisions Register	EMAC_ECOL	Read-write	0x0000_0000
0x64	Transmit Underrun Errors Register	EMAC_TUND	Read-write	0x0000_0000
0x68	Carrier Sense Errors Register	EMAC_CSE	Read-write	0x0000_0000
0x6C	Receive Resource Errors Register	EMAC_RRE	Read-write	0x0000_0000
0x70	Receive Overrun Errors Register	EMAC_ROV	Read-write	0x0000_0000
0x74	Receive Symbol Errors Register	EMAC_RSE	Read-write	0x0000_0000
0x78	Excessive Length Errors Register	EMAC_ELE	Read-write	0x0000_0000
0x7C	Receive Jabbers Register	EMAC_RJA	Read-write	0x0000_0000
0x80	Undersize Frames Register	EMAC_USF	Read-write	0x0000_0000
0x84	SQE Test Errors Register	EMAC_STE	Read-write	0x0000_0000
0x88	Received Length Field Mismatch Register	EMAC_RLE	Read-write	0x0000_0000

Table 41-6. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x90	Hash Register Bottom [31:0] Register	EMAC_HRB	Read-write	0x0000_0000
0x94	Hash Register Top [63:32] Register	EMAC_HRT	Read-write	0x0000_0000
0x98	Specific Address 1 Bottom Register	EMAC_SA1B	Read-write	0x0000_0000
0x9C	Specific Address 1 Top Register	EMAC_SA1T	Read-write	0x0000_0000
0xA0	Specific Address 2 Bottom Register	EMAC_SA2B	Read-write	0x0000_0000
0xA4	Specific Address 2 Top Register	EMAC_SA2T	Read-write	0x0000_0000
0xA8	Specific Address 3 Bottom Register	EMAC_SA3B	Read-write	0x0000_0000
0xAC	Specific Address 3 Top Register	EMAC_SA3T	Read-write	0x0000_0000
0xB0	Specific Address 4 Bottom Register	EMAC_SA4B	Read-write	0x0000_0000
0xB4	Specific Address 4 Top Register	EMAC_SA4T	Read-write	0x0000_0000
0xB8	Type ID Checking Register	EMAC_TID	Read-write	0x0000_0000
0xC0	User Input/Output Register	EMAC_USRIO	Read-write	0x0000_0000
0xC8 - 0xFC	Reserved	–	–	–

41.6.1 Network Control Register

Name: EMAC_NCR

Address: 0x400B0000

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
–	–	–	–	–	THALT	TSTART	BP
7	6	5	4	3	2	1	0
WESTAT	INCSTAT	CLRSTAT	MPE	TE	RE	LLB	LB

- **LB: LoopBack**

Asserts the loopback signal to the PHY.

- **LLB: Loopback local**

Connects `txd` to `rxd`, `tx_en` to `rx_dv`, forces full duplex and drives `rx_clk` and `tx_clk` with `pclk` divided by 4. `rx_clk` and `tx_clk` may glitch as the EMAC 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.

- **RE: Receive enable**

When set, enables the EMAC to receive data. When reset, frame reception stops immediately and the receive FIFO is cleared. The receive queue pointer register is unaffected.

- **TE: Transmit enable**

When set, enables the Ethernet transmitter to send data. When reset transmission, stops immediately, the transmit FIFO and control registers are cleared and the transmit queue pointer register resets 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 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.

41.6.2 Network Configuration Register

Name: EMAC_NCFGR

Address: 0x400B0004

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	IRXFCS	EFRHD	DRFCS	RLCE

15	14	13	12	11	10	9	8
RBOF		PAE	RTY	CLK		–	BIG

7	6	5	4	3	2	1	0
UNI	MTI	NBC	CAF	JFRAME	–	FD	SPD

- **SPD: Speed**

Set to 1 to indicate 100 Mbit/s operation, 0 for 10 Mbit/s. The value of this pin is reflected on the `speed` pin.

- **FD: Full Duplex**

If set to 1, the transmit block ignores the state of collision and carrier sense and allows receive while transmitting. Also controls the `half_duplex` pin.

- **CAF: Copy All Frames**

When set to 1, all valid frames are received.

- **JFRAME: Jumbo Frames**

Set to one to enable jumbo frames of up to 10240 bytes to be accepted.

- **NBC: No Broadcast**

When set to 1, frames addressed to the broadcast address of all ones are not received.

- **MTI: Multicast Hash Enable**

When set, multicast frames are received when the 6-bit hash function of the destination address points to a bit that is set in the hash register.

- **UNI: Unicast Hash Enable**

When set, unicast frames are received when the 6-bit hash function of the destination address points to a bit that is set in the hash register.

- **BIG: Receive 1536 bytes frames**

Setting this bit means the EMAC receives frames up to 1536 bytes in length. Normally, the EMAC would reject any frame above 1518 bytes.

- **CLK: MDC clock divider**

Set according to system clock speed. This determines by what number system clock is divided to generate MDC. For conformance with 802.3, MDC must not exceed 2.5MHz (MDC is only active during MDIO read and write operations).

Value	Name	Description
0	HCLK_8	MCK divided by 8 (MCK up to 20 MHz).
1	HCLK_16	MCK divided by 16 (MCK up to 40 MHz).
2	HCLK_32	MCK divided by 32 (MCK up to 80 MHz).
3	HCLK_64	MCK divided by 64 (MCK up to 160 MHz).

- **RTY: Retry test**

Must be set to zero for normal operation. If set to one, the back off between collisions is always one slot time. Setting this bit to one helps testing the too many retries condition. Also used in the pause frame tests to reduce the pause counters decrement time from 512 bit times, to every `rx_clk` cycle.

- **PAE: Pause Enable**

When set, transmission pauses when a valid pause frame is received.

- **RBOF: Receive Buffer Offset**

Indicates the number of bytes by which the received data is offset from the start of the first receive buffer.

Value	Name	Description
0	OFFSET_0	No offset from start of receive buffer.
1	OFFSET_1	One-byte offset from start of receive buffer.
2	OFFSET_2	Two-byte offset from start of receive buffer.
3	OFFSET_3	Three-byte offset from start of receive buffer.

- **RLCE: Receive Length field Checking Enable**

When set, frames with measured lengths shorter than their length fields are discarded. Frames containing a type ID in bytes 13 and 14 — length/type ID = 0600 — are not be counted as length errors.

- **DRFCS: Discard Receive FCS**

When set, the FCS field of received frames are not be copied to memory.

- **EFRHD:**

Enable Frames to be received in half-duplex mode while transmitting.

- **IRXFCS: Ignore RX FCS**

When set, frames with FCS/CRC errors are not rejected and no FCS error statistics are counted. For normal operation, this bit must be set to 0.

41.6.3 Network Status Register

Name: EMAC_NSR

Address: 0x400B0008

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**

Returns status of the mdio_in pin. Use the PHY maintenance register for reading managed frames rather than this bit.

- **IDLE**

0: The PHY logic is running.

1: The PHY management logic is idle (i.e., has completed).

41.6.4 Transmit Status Register

Name: EMAC_TSR

Address: 0x400B0014

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	COMP	BEX	TGO	RLES	COL	UBR

This register, when read, provides details of the status of a transmit. 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.

- **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.

- **RLES: Retry Limit exceeded**

Cleared by writing a one to this bit.

- **TGO: Transmit Go**

If high transmit is active.

- **BEX: Buffers exhausted mid frame**

If the buffers run out during transmission of a frame, then transmission stops, FCS shall be bad and tx_er asserted. Cleared by writing a one to this bit.

- **COMP: Transmit Complete**

Set when a frame has been transmitted. Cleared by writing a one to this bit.

- **UND: Transmit Underrun**

Set when transmit DMA was not able to read data from memory, either because the bus was not granted in time, because a not OK `hresp(bus error)` was returned or because a used bit was read midway through frame transmission. If this occurs, the transmitter forces `bad` CRC. Cleared by writing a one to this bit.

41.6.5 Receive Buffer Queue Pointer Register

Name: EMAC_RBQP

Address: 0x400B0018

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 points to the entry in the receive buffer queue (descriptor list) currently being used. It is written with the start location of the receive buffer descriptor list. The lower order bits increment as buffers are used up and wrap to their original values after either 1024 buffers or when the wrap bit of the entry is set.

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.

Receive buffer writes also comprise bursts of two words and, as with transmit buffer reads, it is recommended that bit 2 is always written with zero to prevent a burst crossing a 1K boundary, in violation of section 3.6 of the AMBA specification.

- **ADDR: Receive buffer queue pointer address**

Written with the address of the start of the receive queue, reads as a pointer to the current buffer being used.

41.6.6 Transmit Buffer Queue Pointer Register

Name: EMAC_TBQP

Address: 0x400B001C

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 points to the entry in the transmit buffer queue (descriptor list) currently being used. It is written with the start location of the transmit buffer descriptor list. The lower order bits increment as buffers are used up and wrap to their original values after either 1024 buffers or when the wrap bit of the entry is set. This register can only be written when bit 3 in the transmit status register is low.

As transmit buffer reads consist of bursts of two words, it is recommended that bit 2 is always written with zero to prevent a burst crossing a 1K boundary, in violation of section 3.6 of the AMBA specification.

- **ADDR: Transmit buffer queue pointer address**

Written with the address of the start of the transmit queue, reads as a pointer to the first buffer of the frame being transmitted or about to be transmitted.

41.6.7 Receive Status Register

Name: EMAC_RSR

Address: 0x400B0020

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
–	–	–	–	–	OVR	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 rereads the pointer each time a new frame starts until a valid pointer is found. This bit is set at each attempt that fails even if it has not had a successful pointer read since it has been cleared.

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.

- **OVR: Receive Overrun**

The DMA block was unable to store the receive frame to memory, either because the bus was not granted in time or because a not OK `hresp(bus error)` was returned. The buffer is recovered if this happens.

Cleared by writing a one to this bit.

41.6.8 Interrupt Status Register

Name: EMAC_ISR

Address: 0x400B0024

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
–	–	PTZ	PFRE	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TXERR	RLEX	TUND	TXUBR	RXUBR	RCOMP	MFD

- **MFD: Management Frame Done**

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: Receive Used Bit Read**

Set when a receive buffer descriptor is read with its used bit set. Cleared on read.

- **TXUBR: Transmit Used Bit Read**

Set when a transmit buffer descriptor is read with its used bit set. Cleared on read.

- **TUND: Ethernet Transmit Buffer Underrun**

The transmit DMA did not fetch frame data in time for it to be transmitted or `hresp` returned not OK. Also set if a used bit is read mid-frame or when a new transmit queue pointer is written. Cleared on read.

- **RLEX: Retry Limit Exceeded**

Cleared on read.

- **TXERR: Transmit Error**

Transmit buffers exhausted in mid-frame - transmit error. Cleared on read.

- **TCOMP: Transmit Complete**

Set when a frame has been transmitted. Cleared on read.

- **ROVR: Receive Overrun**

Set when the receive overrun status bit gets set. Cleared on read.

- **HRESP: Hresp not OK**

Set when the DMA block sees a `bus error`. Cleared on read.

- **PFRE: Pause Frame Received**

Indicates a valid pause has been received. Cleared on a read.

- **PTZ: Pause Time Zero**

Set when the pause time register, 0x38 decrements to zero. Cleared on a read.

41.6.9 Interrupt Enable Register

Name: EMAC_IER

Address: 0x400B0028

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
–	–	PTZ	PFR	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TXERR	RLE	TUND	TXUBR	RXUBR	RCOMP	MFD

- **MFD: Management Frame sent**

Enable management done interrupt.

- **RCOMP: Receive Complete**

Enable receive complete interrupt.

- **RXUBR: Receive Used Bit Read**

Enable receive used bit read interrupt.

- **TXUBR: Transmit Used Bit Read**

Enable transmit used bit read interrupt.

- **TUND: Ethernet Transmit Buffer Underrun**

Enable transmit underrun interrupt.

- **RLE: Retry Limit Exceeded**

Enable retry limit exceeded interrupt.

- **TXERR**

Enable transmit buffers exhausted in mid-frame interrupt.

- **TCOMP: Transmit Complete**

Enable transmit complete interrupt.

- **ROVR: Receive Overrun**

Enable receive overrun interrupt.

- **HRESP: Hresp not OK**

Enable Hresp not OK interrupt.

- **PFR: Pause Frame Received**

Enable pause frame received interrupt.

- **PTZ: Pause Time Zero**

Enable pause time zero interrupt.

41.6.10 Interrupt Disable Register

Name: EMAC_IDR

Address: 0x400B002C

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
–	–	PTZ	PFR	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TXERR	RLE	TUND	TXUBR	RXUBR	RCOMP	MFD

- **MFD: Management Frame sent**

Disable management done interrupt.

- **RCOMP: Receive Complete**

Disable receive complete interrupt.

- **RXUBR: Receive Used Bit Read**

Disable receive used bit read interrupt.

- **TXUBR: Transmit Used Bit Read**

Disable transmit used bit read interrupt.

- **TUND: Ethernet Transmit Buffer Underrun**

Disable transmit underrun interrupt.

- **RLE: Retry Limit Exceeded**

Disable retry limit exceeded interrupt.

- **TXERR**

Disable transmit buffers exhausted in mid-frame interrupt.

- **TCOMP: Transmit Complete**

Disable transmit complete interrupt.

- **ROVR: Receive Overrun**

Disable receive overrun interrupt.

- **HRESP: Hresp not OK**

Disable Hresp not OK interrupt.

- **PFR: Pause Frame Received**

Disable pause frame received interrupt.

- **PTZ: Pause Time Zero**

Disable pause time zero interrupt.

41.6.11 Interrupt Mask Register

Name: EMAC_IMR

Address: 0x400B0030

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
–	–	PTZ	PFR	HRESP	ROVR	–	–

7	6	5	4	3	2	1	0
TCOMP	TXERR	RLE	TUND	TXUBR	RXUBR	RCOMP	MFD

- **MFD: Management Frame sent**

Management done interrupt masked.

- **RCOMP: Receive Complete**

Receive complete interrupt masked.

- **RXUBR: Receive Used Bit Read**

Receive used bit read interrupt masked.

- **TXUBR: Transmit Used Bit Read**

Transmit used bit read interrupt masked.

- **TUND: Ethernet Transmit Buffer Underrun**

Transmit underrun interrupt masked.

- **RLE: Retry Limit Exceeded**

Retry limit exceeded interrupt masked.

- **TXERR**

Transmit buffers exhausted in mid-frame interrupt masked.

- **TCOMP: Transmit Complete**

Transmit complete interrupt masked.

- **ROVR: Receive Overrun**

Receive overrun interrupt masked.

- **HRESP: Hresp not OK**

Hresp not OK interrupt masked.

- **PFR: Pause Frame Received**

Pause frame received interrupt masked.

- **PTZ: Pause Time Zero**

Pause time zero interrupt masked.

41.6.12 PHY Maintenance Register

Name: EMAC_MAN

Address: 0x400B0034

Access: Read-write

31	30	29	28	27	26	25	24
SOF		RW		PHYA			

23	22	21	20	19	18	17	16
PHYA		REGA				CODE	

15	14	13	12	11	10	9	8
DATA							

7	6	5	4	3	2	1	0
DATA							

- **DATA**

For a write operation this is written with the data to be written to the PHY.

After a read operation this contains the data read from the PHY.

- **CODE:**

Must be written to 10. Reads as written.

- **REGA: Register Address**

Specifies the register in the PHY to access.

- **PHYA: PHY Address**

- **RW: Read-write**

10 is read; 01 is write. Any other value is an invalid PHY management frame

- **SOF: Start of frame**

Must be written 01 for a valid frame.

41.6.13 Pause Time Register

Name: EMAC_PTR

Address: 0x400B0038

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
PTIME							

7	6	5	4	3	2	1	0
PTIME							

- **PTIME: Pause Time**

Stores the current value of the pause time register which is decremented every 512 bit times.

41.6.14 Hash Register Bottom

Name: EMAC_HRB

Address: 0x400B0090

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:**

Bits 31:0 of the hash address register. See “Hash Addressing” on page 1251.

41.6.15 Hash Register Top

Name: EMAC_HRT

Address: 0x400B0094

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:**

Bits 63:32 of the hash address register. See “Hash Addressing” on page 1251.

41.6.16 Specific Address 1 Bottom Register

Name: EMAC_SA1B

Address: 0x400B0098

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**

Least significant bits of the destination address. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

41.6.17 Specific Address 1 Top Register

Name: EMAC_SA1T

Address: 0x400B009C

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**

The most significant bits of the destination address, that is bits 47 to 32.

41.6.18 Specific Address 2 Bottom Register

Name: EMAC_SA2B

Address: 0x400B00A0

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**

Least significant bits of the destination address. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

41.6.19 Specific Address 2 Top Register

Name: EMAC_SA2T

Address: 0x400B00A4

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**

The most significant bits of the destination address, that is bits 47 to 32.

41.6.20 Specific Address 3 Bottom Register

Name: EMAC_SA3B

Address: 0x400B00A8

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**

Least significant bits of the destination address. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

41.6.21 Specific Address 3 Top Register

Name: EMAC_SA3T

Address: 0x400B00AC

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**

The most significant bits of the destination address, that is bits 47 to 32.

41.6.22 Specific Address 4 Bottom Register

Name: EMAC_SA4B

Address: 0x400B00B0

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**

Least significant bits of the destination address. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

41.6.23 Specific Address 4 Top Register

Name: EMAC_SA4T

Address: 0x400B00B4

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**

The most significant bits of the destination address, that is bits 47 to 32.

41.6.24 Type ID Checking Register

Name: EMAC_TID

Address: 0x400B00B8

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 checking**

For use in comparisons with received frames TypeID/Length field.

41.6.25 User Input/Output Register

Name: EMAC_USRIO

Address: 0x400B00C0

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
–	–	–	–	–	–	CLKEN	RMII

- **RMII: Reduce MII**

When set, this bit enables the RMII operation mode. When reset, it selects the MII mode.

- **CLKEN: Clock Enable**

When set, this bit enables the transceiver input clock.

Setting this bit to 0 reduces power consumption when the treasurer is not used.

41.6.26 EMAC Statistic Registers

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. The receive statistics registers are only incremented when the receive enable bit is set in the network control register. To write to these registers, bit 7 must be set in the network control register. The statistics register block contains the following registers.

41.6.26.1Pause Frames Received Register

Name: EMAC_PFR

Address: 0x400B003C

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
FROK							

7	6	5	4	3	2	1	0
FROK							

- **FROK: Pause Frames received OK**

A 16-bit register counting the number of good pause frames received. A good frame has a length of 64 to 1518 (1536 if bit 8 set in network configuration register) and has no FCS, alignment or receive symbol errors.

41.6.26.2Frames Transmitted OK Register

Name: EMAC_FTO

Address: 0x400B0040

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
FTOK							

15	14	13	12	11	10	9	8
FTOK							

7	6	5	4	3	2	1	0
FTOK							

- **FTOK: Frames Transmitted OK**

A 24-bit register counting the number of frames successfully transmitted, i.e., no underrun and not too many retries.

41.6.26.3Single Collision Frames Register

Name: EMAC_SCF

Address: 0x400B0044

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
SCF							

7	6	5	4	3	2	1	0
SCF							

- **SCF: Single Collision Frames**

A 16-bit register counting the number of frames experiencing a single collision before being successfully transmitted, i.e., no underrun.

41.6.26.4Multicollision Frames Register

Name: EMAC_MCF

Address: 0x400B0048

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
MCF							

7	6	5	4	3	2	1	0
MCF							

- **MCF: Multicollision Frames**

A 16-bit register counting the number of frames experiencing between two and fifteen collisions prior to being successfully transmitted, i.e., no underrun and not too many retries.

41.6.26.5Frames Received OK Register

Name: EMAC_FRO

Address: 0x400B004C

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
FROK							

15	14	13	12	11	10	9	8
FROK							

7	6	5	4	3	2	1	0
FROK							

- **FROK: Frames Received OK**

A 24-bit register counting the number of good frames received, i.e., address recognized and successfully copied to memory. A good frame is of length 64 to 1518 bytes (1536 if bit 8 set in network configuration register) and has no FCS, alignment or receive symbol errors.

41.6.26.6Frames Check Sequence Errors Register

Name: EMAC_FCSE

Address: 0x400B0050

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
FCSE							

- **FCSE: Frame Check Sequence Errors**

An 8-bit register counting frames that are an integral number of bytes, have bad CRC and are between 64 and 1518 bytes in length (1536 if bit 8 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.

41.6.26.7Alignment Errors Register

Name: EMAC_ALE

Address: 0x400B0054

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
ALE							

- **ALE: Alignment Errors**

An 8-bit register counting 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 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.

41.6.26.8Deferred Transmission Frames Register

Name: EMAC_DTF

Address: 0x400B0058

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
DTF							

7	6	5	4	3	2	1	0
DTF							

- **DTF: Deferred Transmission Frames**

A 16-bit register counting 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 underrun.

41.6.26.9Late Collisions Register

Name: EMAC_LCOL

Address: 0x400B005C

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
LCOL							

- **LCOL: Late Collisions**

An 8-bit register counting the number of frames that experience a collision after the slot time (512 bits) has expired. A late collision is counted twice; i.e., both as a collision and a late collision.

41.6.26.10 Excessive Collisions Register

Name: EMAC_ECOL

Address: 0x400B0060

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
EXCOL							

- **EXCOL: Excessive Collisions**

An 8-bit register counting the number of frames that failed to be transmitted because they experienced 16 collisions.

41.6.26.11 Transmit Underrun Errors Register

Name: EMAC_TUND

Address: 0x400B0064

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
TUND							

- **TUND: Transmit Underruns**

An 8-bit register counting the number of frames not transmitted due to a transmit DMA underrun. If this register is incremented, then no other statistics register is incremented.

41.6.26.12Carrier Sense Errors Register

Name: EMAC_CSE

Address: 0x400B0068

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
CSE							

- **CSE: Carrier Sense Errors**

An 8-bit register counting 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 underrun). 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.

41.6.26.13 Receive Resource Errors Register

Name: EMAC_RRE

Address: 0x400B006C

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
RRE							

7	6	5	4	3	2	1	0
RRE							

- **RRE: Receive Resource Errors**

A 16-bit register counting the number of frames that were address matched but could not be copied to memory because no receive buffer was available.

41.6.26.14 Receive Overrun Errors Register

Name: EMAC_ROV

Address: 0x400B0070

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
ROVR							

- **ROVR: Receive Overrun**

An 8-bit register counting the number of frames that are address recognized but were not copied to memory due to a receive DMA overrun.

41.6.26.15Receive Symbol Errors Register

Name: EMAC_RSE

Address: 0x400B0074

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
RSE							

- **RSE: Receive Symbol Errors**

An 8-bit register counting the number of frames that had `rx_er` asserted during reception. Receive symbol errors are also counted as an FCS or alignment error if the frame is between 64 and 1518 bytes in length (1536 if bit 8 is set in the network configuration register). If the frame is larger, it is recorded as a jabber error.

41.6.26.16 Excessive Length Errors Register

Name: EMAC_ELE

Address: 0x400B0078

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
EXL							

- **EXL: Excessive Length Errors**

An 8-bit register counting the number of frames received exceeding 1518 bytes (1536 if bit 8 set in network configuration register) in length but do not have either a CRC error, an alignment error nor a receive symbol error.

41.6.26.17Receive Jabbers Register

Name: EMAC_RJA

Address: 0x400B007C

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
RJB							

- **RJB: Receive Jabbers**

An 8-bit register counting the number of frames received exceeding 1518 bytes (1536 if bit 8 set in network configuration register) in length and have either a CRC error, an alignment error or a receive symbol error.

41.6.26.18 Undersize Frames Register

Name: EMAC_USF

Address: 0x400B0080

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
USF							

- **USF: Undersize frames**

An 8-bit register counting the number of frames received less than 64 bytes in length but do not have either a CRC error, an alignment error or a receive symbol error.

41.6.26.19SQE Test Errors Register

Name: EMAC_STE

Address: 0x400B0084

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
SQER							

- **SQER: SQE test errors**

An 8-bit register counting the number of frames where `col` was not asserted within 96 bit times (an interframe gap) of `tx_en` being deasserted in half duplex mode.

41.6.26.20Received Length Field Mismatch Register

Name: EMAC_RLE

Address: 0x400B0088

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
RLFM							

- **RLFM: Receive Length Field Mismatch**

An 8-bit register counting the number of frames received that have a measured length shorter than that extracted from its length field. Checking is enabled through bit 16 of the network configuration register. Frames containing a type ID in bytes 13 and 14 (i.e., length/type ID = 0x0600) are not counted as length field errors, neither are excessive length frames.

42. True Random Number Generator (TRNG)

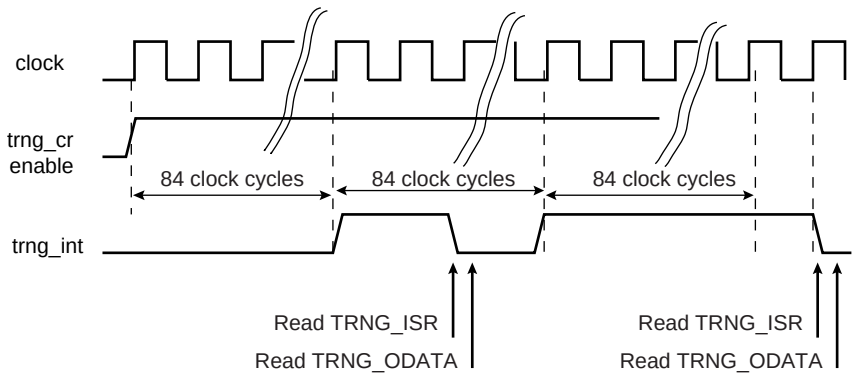
42.1 Description

The True Random Number Generator (TRNG) passes the American *NIST Special Publication 800-22 and Diehard Random Tests Suites.*

As soon as the TRNG is enabled (TRNG_CTRL register), the generator provides one 32-bit value every 84 clock cycles. Interrupt trng_int can be enabled through the TRNG_IER register (respectively disabled in TRNG_IDR). This interrupt is set when a new random value is available and is cleared when the status register is read (TRNG_SR register). The flag DATRDY of the status register (TRNG_ISR) is set when the random data is ready to be read out on the 32-bit output data register (TRNG_ODATA).

The normal mode of operation checks that the status register flag equals 1 before reading the output data register when a 32-bit random value is required by the software application.

Figure 42-1. TRNG Data Generation Sequence



42.2 Embedded Characteristics

- Passed NIST Special Publication 800-22 Tests Suite
- Passed Diehard Random Tests Suite
- Provides a 32-bit Random Number Every 84 Clock Cycles

42.3 True Random Number Generator (TRNG) User Interface

Table 42-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	TRNG_CR	Write-only	–
0x10	Interrupt Enable Register	TRNG_IER	Write-only	–
0x14	Interrupt Disable Register	TRNG_IDR	Write-only	–
0x18	Interrupt Mask Register	TRNG_IMR	Read-only	0x0000_0000
0x1C	Interrupt Status Register	TRNG_ISR	Read-only	0x0000_0000
0x50	Output Data Register	TRNG_ODATA	Read-only	0x0000_0000

42.3.1 TRNG Control Register

Name: TRNG_CR

Address: 0x400BC000

Access: Write-only

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

- **ENABLE: Enables the TRNG to provide random values**

0: Disables the TRNG.

1: Enables the TRNG.

- **KEY: Security Key**

KEY = 0x524e47 (RNG in ASCII)

This key is to be written when the ENABLE bit is set or cleared.

42.3.2 TRNG Interrupt Enable Register

Name: TRNG_IER

Address: 0x400BC010

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
–	–	–	–	–	–	–	DATRDY

- **DATRDY: Data Ready Interrupt Enable**

0: No effect.

1: Enables the corresponding interrupt.

42.3.3 TRNG Interrupt Disable Register

Name: TRNG_IDR

Address: 0x400BC014

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
–	–	–	–	–	–	–	DATRDY

- **DATRDY: Data Ready Interrupt Disable**

0: No effect.

1: Disables the corresponding interrupt.

42.3.4 TRNG Interrupt Mask Register

Name: TRNG_IMR

Address: 0x400BC018

Reset: 0x0000_0000

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
–	–	–	–	–	–	–	DATRDY

- **DATRDY: Data Ready Interrupt Mask**

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

42.3.5 TRNG Interrupt Status Register

Name: TRNG_ISR

Address: 0x400BC01C

Reset: 0x0000_0000

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
–	–	–	–	–	–	–	DATRDY

- **DATRDY: Data Ready**

0: Output data is not valid or TRNG is disabled.

1: New Random value is completed.

DATRDY is cleared when this register is read.

42.3.6 TRNG Output Data Register

Name: TRNG_ODATA

Address: 0x400BC050

Reset: 0x0000_0000

Access: Read-only

31	30	29	28	27	26	25	24
ODATA							

23	22	21	20	19	18	17	16
ODATA							

15	14	13	12	11	10	9	8
ODATA							

7	6	5	4	3	2	1	0
ODATA							

- **ODATA: Output Data**

The 32-bit Output Data register contains the 32-bit random data.

43. Analog-to-Digital Converter (ADC)

43.1 Description

The ADC is based on a 12-bit Analog-to-Digital Converter (ADC) managed by an ADC 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 ADC supports an 10-bit or 12-bit resolution mode, 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 ADC 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 ADC 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 ADC 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.

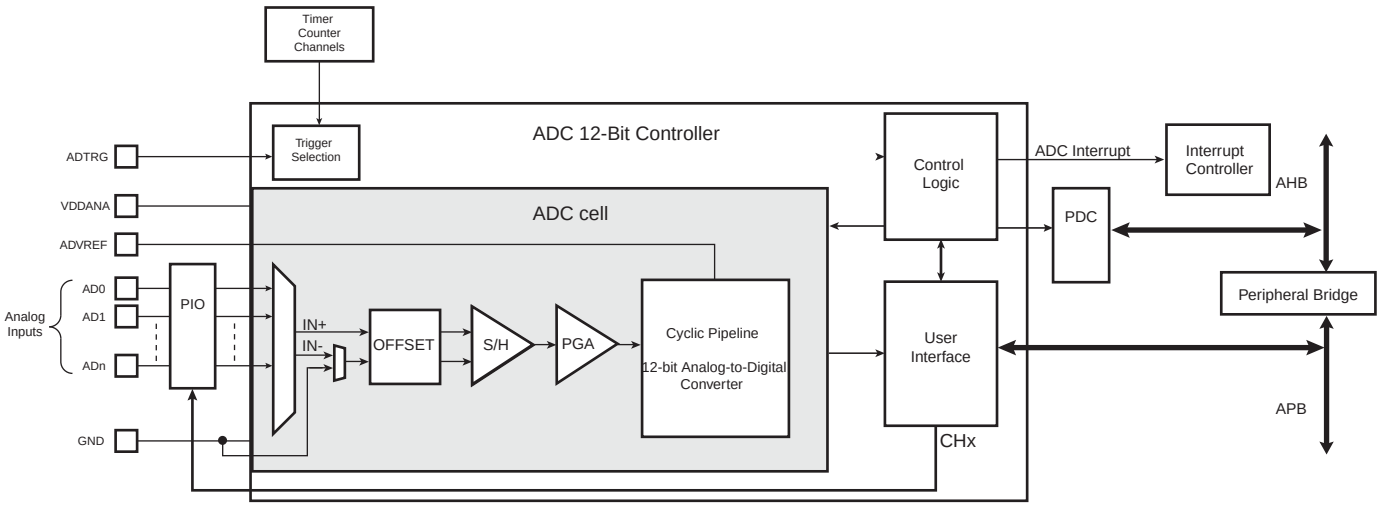
Finally, the user can configure ADC timings, such as Startup Time and Tracking Time.

43.2 Embedded Characteristics

- 12-bit Resolution
- 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
- 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 ADC Timings Configuration
- Two Sleep Modes and Conversion Sequencer
  - Automatic Wakeup on Trigger and Back to Sleep Mode after Conversions of all Enabled Channels
  - Possibility of Customized Channel Sequence
- Standby Mode for Fast Wakeup Time Response
  - Power Down Capability
- Automatic Window Comparison of Converted Values
- Write Protect Registers

43.3 Block Diagram

Figure 43-1. Analog-to-Digital Converter Block Diagram



43.4 Signal Description

Table 43-1. ADC Pin Description

Pin Name	Description
VDDANA	Analog power supply
ADVREF	Reference voltage[1]
AD0 - AD15	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 ADC Controller is not continuously clocked. The programmer must first enable the ADC Controller MCK in the Power Management Controller (PMC) before using the ADC Controller. However, if the application does not require ADC operations, the ADC Controller clock can be stopped when not needed and restarted when necessary. Configuring the ADC Controller does not require the ADC Controller clock to be enabled.

43.5.2 Interrupt Sources

The ADC interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the ADC interrupt requires the NVIC to be programmed first.

Table 43-2. Peripheral IDs

Instance	ID
ADC	37

43.5.3 Analog Inputs

The analog input pins can be multiplexed with PIO lines. In this case, the assignment of the ADC input is automatically done as soon as the corresponding channel is enabled by writing the register ADC_CHER. By default, after reset, the PIO line is configured as input with its pull-up enabled and the ADC input is connected to the GND.

43.5.4 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). To activate the temperature sensor, TSON bit (ADC_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 ADC function.

Table 43-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
ADC	ADTRG	PA11	B
ADC	AD0	PA2	X1
ADC	AD1/WKUP1	PA3	X1
ADC	AD2	PA4	X1
ADC	AD3	PA6	X1
ADC	AD4	PA22	X1
ADC	AD5	PA23	X1
ADC	AD6	PA24	X1
ADC	AD7	PA16	X1
ADC	AD8	PB12	X1
ADC	AD9	PB13	X1
ADC	AD10	PB17	X1
ADC	AD11	PB18	X1
ADC	AD12	PB19	X1
ADC	AD13	PB20	X1
ADC	AD14/WKUP13	PB21	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 ADC Controller has the FAULT output connected to the FAULT input of PWM. Please refer to Section 43.6.12 ”Fault Output” and implementation of the PWM in the product.

43.5.9 Conversion Performances

For performance and electrical characteristics of the ADC, see the product DC Characteristics section.

43.6 Functional Description

43.6.1 Analog-to-digital Conversion

The ADC uses the ADC 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 “ADC Mode Register” on page 1333 and Transfer Clock cycles as defined in the field TRANSFER of the same register. The ADC Clock frequency is selected in the PRESCAL field of the Mode Register (ADC_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 ADC clock range is between MCK/2, if PRESCAL is 0, and MCK/512, if PRESCAL is set to 255 (0xFF). PRESCAL must be programmed in order to provide an ADC clock frequency according to the parameters given in the product Electrical Characteristics section.

Figure 43-2. Sequence of ADC conversions when Tracking time > Conversion time

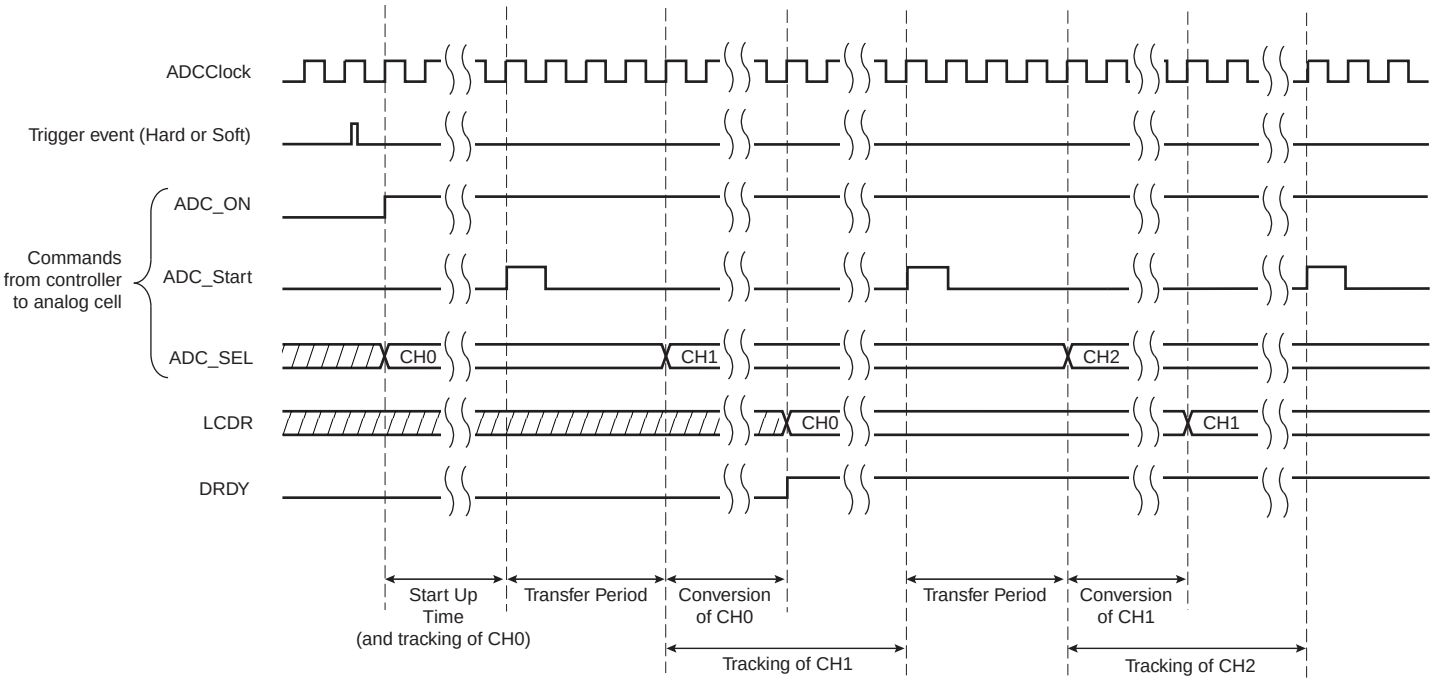
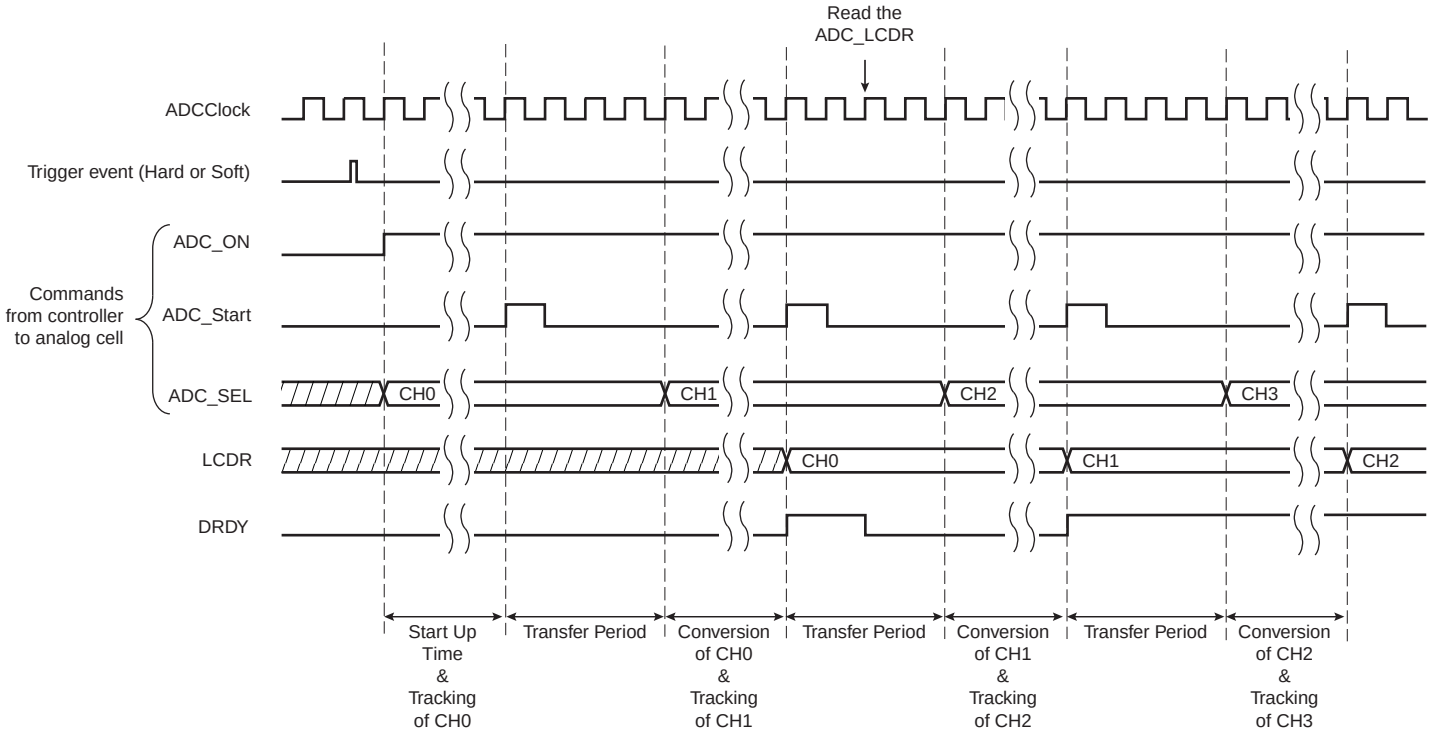


Figure 43-3. Sequence of ADC conversions when Tracking time < Conversion time



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 ADC supports 10-bit or 12-bit resolutions. The 10-bit selection is performed by setting the LOWRES bit in the ADC Mode Register (ADC_MR). By default, after a reset, the resolution is the highest and the DATA field in the data registers is fully used. By setting the LOWRES bit, the ADC switches to the lowest resolution and the conversion results can be read in the lowest significant bits of the data registers. The two highest bits of the DATA field in the corresponding ADC_CDR register and of the LDATA field in the ADC_LCDR register read 0.

Moreover, when a PDC channel is connected to the ADC, 12-bit or 10-bit resolution sets the transfer request size to 16 bits.

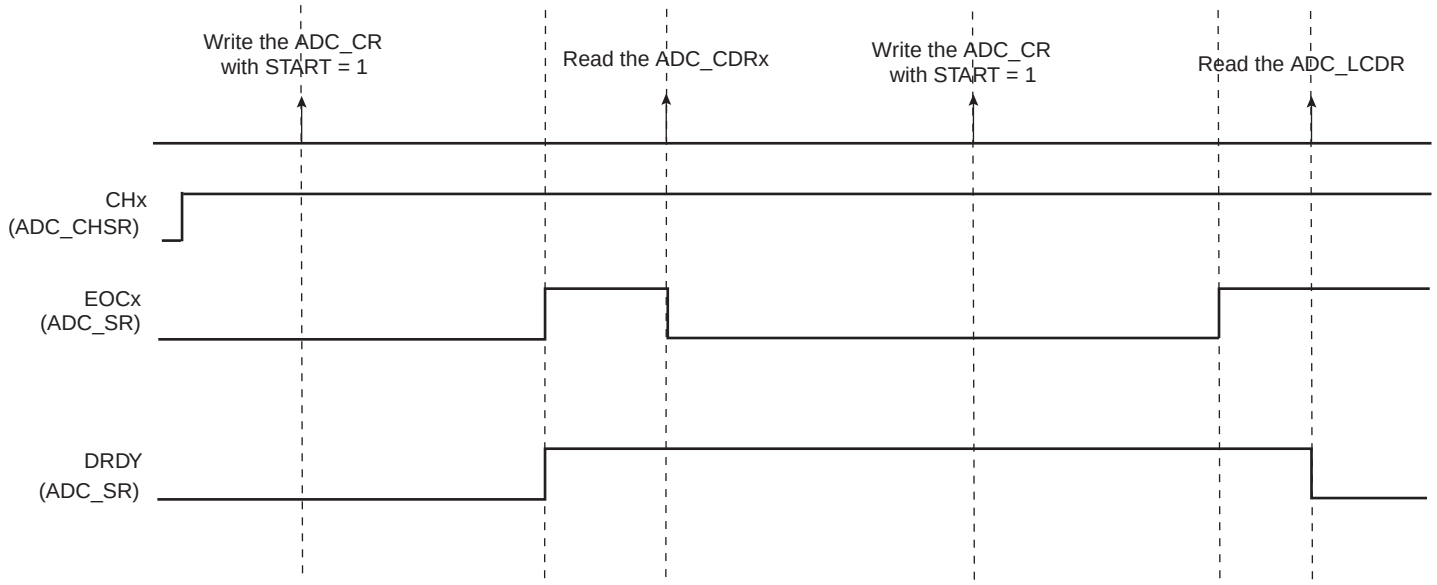
43.6.4 Conversion Results

When a conversion is completed, the resulting 12-bit digital value is stored in the Channel Data Register (ADC_CDRx) of the current channel and in the ADC Last Converted Data Register (ADC_LCDR). By setting the TAG option in the ADC_EMR, the ADC_LCDR presents the channel number associated to the last converted data in the CHNB field.

The channel EOC bit in the Status Register (ADC_SR) 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 one of the ADC_CDR registers clears the corresponding EOC bit. Reading ADC_LCDR clears the DRDY bit and EOC bit corresponding to the last converted channel.

Figure 43-4. EOCx and DRDY Flag Behavior

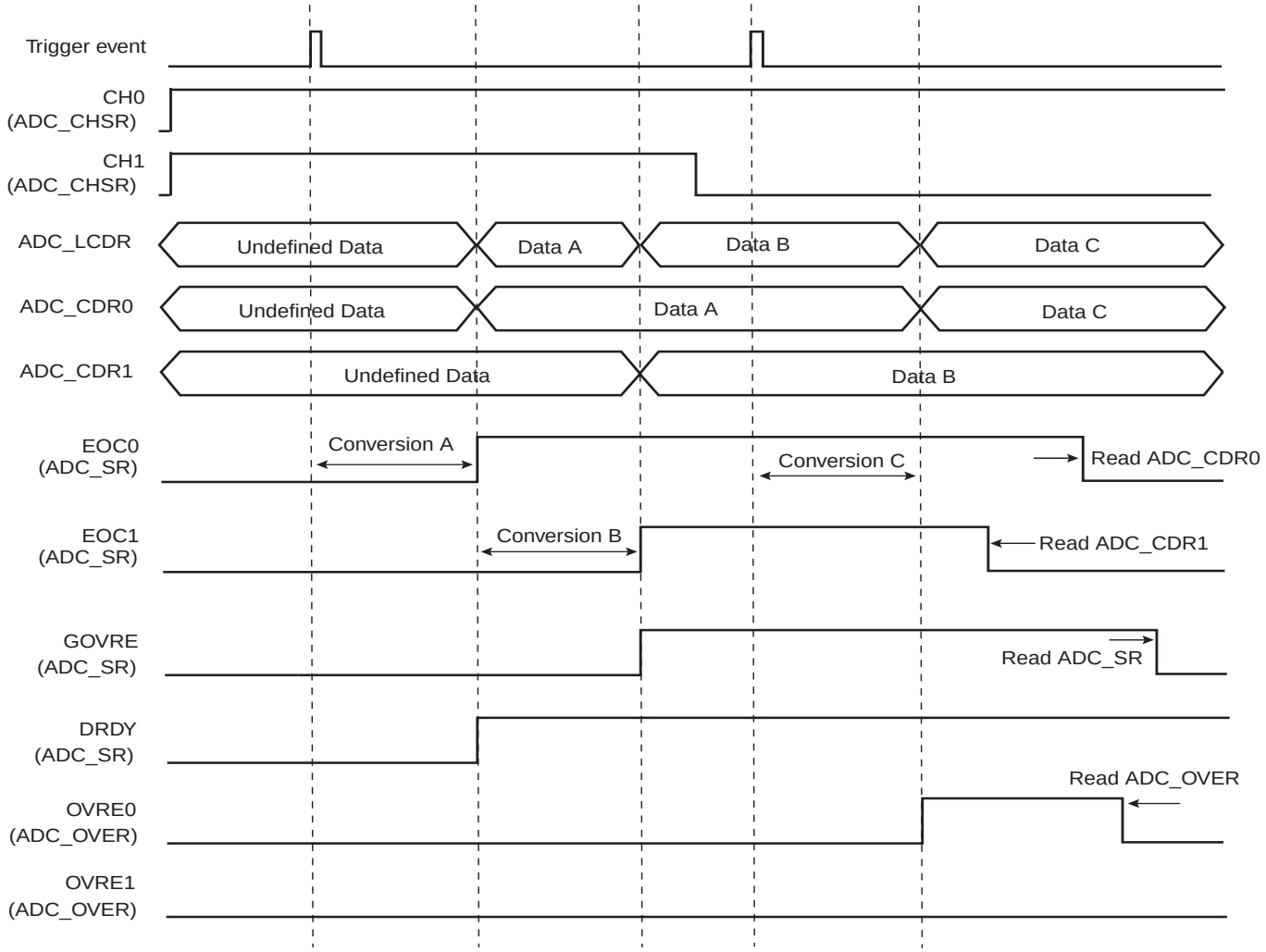


If the ADC_CDR is not read before further incoming data is converted, the corresponding Overrun Error (OVREx) flag is set in the Overrun Status Register (ADC_OVER).

Likewise, new data converted when DRDY is high sets the GOVRE bit (General Overrun Error) in ADC_SR.

The OVREx flag is automatically cleared when ADC_OVER is read, and GOVRE flag is automatically cleared when ADC_SR 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 ADC_SR are unpredictable.

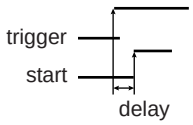
43.6.5 Conversion Triggers

Conversions of the active analog channels are started with a software or hardware trigger. The software trigger is provided by writing the Control Register (ADC_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 ADC (ADTRG). The hardware trigger is selected with the TRGSEL field in the Mode Register (ADC_MR). The selected hardware trigger is enabled with the TRGEN bit in the Mode Register (ADC_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 ADC_MR, ADC_CHSR, ADC_SEQR1, ADC_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 ADC clock period.



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 ADC hardware logic automatically performs the conversions on the active channels, then waits for a new request. The Channel Enable (ADC_CHER) and Channel Disable (ADC_CHDR) Registers permit the analog channels to be enabled or disabled independently.

If the ADC 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 ADC Sleep Mode maximizes power saving by automatically deactivating the ADC when it is not being used for conversions. Sleep Mode is selected by setting the SLEEP bit in the Mode Register ADC_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 ADC Characteristics section).

When a start conversion request occurs, the ADC 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 ADC 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 ADC Mode Register (ADC_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 ADC cell is not fully deactivated while no conversion is requested, thereby providing less power saving but faster wakeup.

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, ADC_SEQR1 and ADC_SEQR2 and setting to 1 the USEQ bit of the Mode Register (ADC_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 ADC_SEQR1 and ADC_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 ADC_CHSR[15], thus disabling the 16THCH field of ADC_SEQR2.

If all ADC channels (i.e. 16) are used on an application board, there is no restriction of usage of the user sequence. But as soon as some ADC channels are not enabled for conversion but rather used as pure digital inputs, the respective indexes of these channels cannot be used in the user sequence fields (ADC_SEQR1, ADC_SEQR2 bitfields). For example, if channel 4 is disabled (ADC_CSR[4] = 0), ADC_SEQR1, ADC_SEQR2 register bitfields USCH1 up to USCH16 must not contain the value 4. Thus the length of the user sequence may be limited by this behavior.

As an example, if only 4 channels over 16 (CH0 up to CH3) are selected for ADC conversions, the user sequence length cannot exceed 4 channels. Each trigger event may launch up to 4 successive conversions of any combination of channels 0 up to 3 but no more (i.e. in this case the sequence CH0, CH0, CH1, CH1, CH1 is impossible).

A sequence that repeats several times the same channel requires more enabled channels than channels actually used for conversion. For example, a sequence like CH0, CH0, CH1, CH1 requires 4 enabled channels (4 free channels on application boards) whereas only CH0, CH1 are really converted.

Note: The reference voltage pins always remain connected in normal mode as in sleep mode.

43.6.7 Comparison Window

The ADC 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 (ADC_EMR). The comparison can be done on all channels or only on the channel specified in CMPSEL field of ADC_EMR. To compare all channels the CMP_ALL parameter of ADC_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 CMPFILTER field of ADC_EMR.

The flag can be read on the COMPE bit of the Interrupt Status Register (ADC_ISR) and can trigger an interrupt.

The High Threshold and the Low Threshold can be read/write in the Comparison Window Register (ADC_CWR).

43.6.8 Differential Inputs

The ADC can be used either as a single ended ADC (DIFF bit equal to 0) or as a fully differential ADC (DIFF bit equal to 1) as shown in Figure 43-6. By default, after a reset, the ADC is in single ended mode.

If ANACH is set in ADC_MR the ADC 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 43-4. Input Pins and Channel Number in Single Ended Mode

Input Pins	Channel Number
AD0	CH0
AD1	CH1
AD2	CH2
AD3	CH3
AD4	CH4

Table 43-4. Input Pins and Channel Number in Single Ended Mode (Continued)

Input Pins	Channel Number
AD5	CH5
AD6	CH6
AD7	CH7
AD8	CH8
AD9	CH9
AD10	CH10
AD11	CH11
AD12	CH12
AD13	CH13
AD14	CH14
AD15	CH15

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

43.6.9 Input Gain and Offset

The ADC has a built in Programmable Gain Amplifier (PGA) and Programmable Offset.

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 ADC_MR the ADC 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 (ADC_CGR) as shown in Table 43-6.

Table 43-6. Gain of the Sample and Hold Unit: GAIN Bits and DIFF Bit.

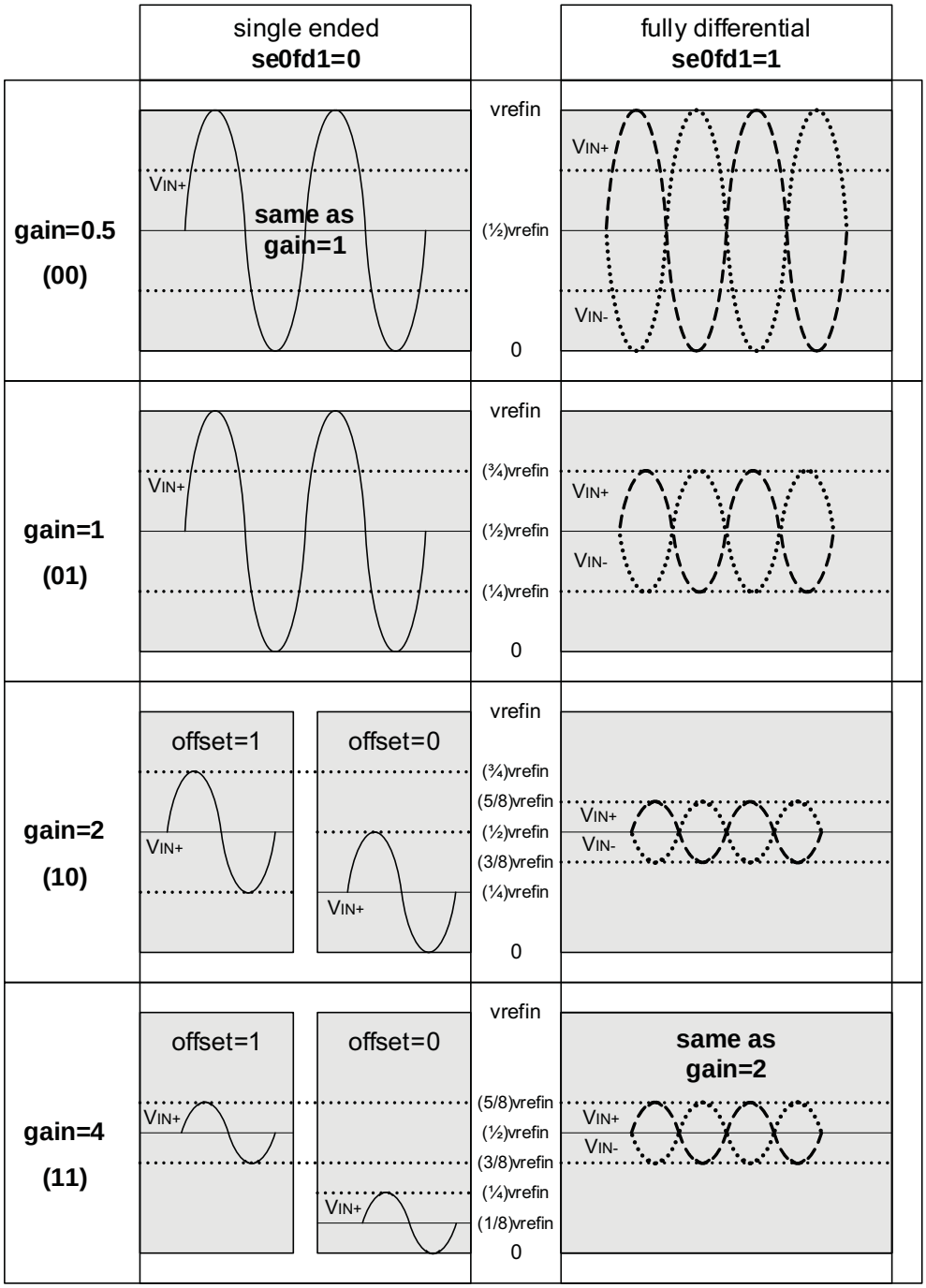
GAIN<0:1>	GAIN (DIFF = 0)	GAIN (DIFF = 1)
00	1	0.5
01	1	1
10	2	2
11	4	2

To allow full range, analog offset of the ADC can be configured by the OFFSET bit of the Channel Offset Register (ADC_COR). The Offset is only available in Single Ended Mode.

Table 43-7. Offset of the Sample and Hold Unit: OFFSET DIFF and Gain (G)

OFFSET Bit	OFFSET (DIFF = 0)	OFFSET (DIFF = 1)
0	0	0
1	(G-1)Vrefin/2	

Figure 43-6. Analog Full Scale Ranges in Single Ended/Differential Applications Versus Gain and Offset



43.6.10 ADC Timings

Each ADC has its own minimal Startup Time that is programmed through the field STARTUP in the Mode Register, ADC_MR.

A minimal Tracking Time is necessary for the ADC 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, ADC_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 “ADC 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 ADC. This must be taken into consideration to program a precise value in the TRACKTIM field. See the product ADC Characteristics section.

43.6.11 Buffer Structure

The PDC read channel is triggered each time a new data is stored in ADC_LCDR register. The same structure of data is repeatedly stored in ADC_LCDR register each time a trigger event occurs. Depending on user mode of operation (ADC_MR, ADC_CHSR, ADC_SEQR1, ADC_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 ADC_EMR register, the 4 most significant bits are carrying the channel number thus allowing an easier post-processing in the PDC buffer or better checking the PDC buffer integrity.

43.6.12 Fault Output

The ADC Controller internal fault output is directly connected to PWM fault input. Fault output may be asserted according to the configuration of ADC_EMR (Extended Mode Register) and ADC_CWR (Compare Window Register) and converted values. When the Compare occurs, the ADC 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 ADC Controller, the PWM outputs are immediately placed in a safe state (pure combinational path). Note that the ADC fault output connected to the PWM is not the COMPE bit. Thus the Fault Mode (FMOD) within the PWM configuration must be FMOD = 1.

43.6.13 Write Protection Registers

To prevent any single software error that may corrupt ADC behavior, certain address spaces can be write-protected by setting the WPEN bit in the "ADC Write Protect Mode Register" (ADC_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the ADC Write Protect Status Register (ADC_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the ADC Write Protect Mode Register (ADC_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- "ADC Mode Register" on page 1333
- "ADC Channel Sequence 1 Register" on page 1336
- "ADC Channel Sequence 2 Register" on page 1337
- "ADC Channel Enable Register" on page 1338
- "ADC Channel Disable Register" on page 1339
- "ADC Extended Mode Register" on page 1347
- "ADC Compare Window Register" on page 1348
- "ADC Channel Gain Register" on page 1349
- "ADC Channel Offset Register" on page 1350
- "ADC Analog Control Register" on page 1352

43.7 Analog-to-Digital Converter (ADC) User Interface

Any offset not listed in Table 43-8 must be considered as “reserved”.

Table 43-8. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	ADC_CR	Write-only	–
0x04	Mode Register	ADC_MR	Read-write	0x00000000
0x08	Channel Sequence Register 1	ADC_SEQR1	Read-write	0x00000000
0x0C	Channel Sequence Register 2	ADC_SEQR2	Read-write	0x00000000
0x10	Channel Enable Register	ADC_CHER	Write-only	–
0x14	Channel Disable Register	ADC_CHDR	Write-only	–
0x18	Channel Status Register	ADC_CHSR	Read-only	0x00000000
0x1C	Reserved	–	–	–
0x20	Last Converted Data Register	ADC_LCDR	Read-only	0x00000000
0x24	Interrupt Enable Register	ADC_IER	Write-only	–
0x28	Interrupt Disable Register	ADC_IDR	Write-only	–
0x2C	Interrupt Mask Register	ADC_IMR	Read-only	0x00000000
0x30	Interrupt Status Register	ADC_ISR	Read-only	0x00000000
0x34	Reserved	–	–	–
0x38	Reserved	–	–	–
0x3C	Overrun Status Register	ADC_OVER	Read-only	0x00000000
0x40	Extended Mode Register	ADC_EMR	Read-write	0x00000000
0x44	Compare Window Register	ADC_CWR	Read-write	0x00000000
0x48	Channel Gain Register	ADC_CGR	Read-write	0x00000000
0x4C	Channel Offset Register	ADC_COR	Read-write	0x00000000
0x50	Channel Data Register 0	ADC_CDR0	Read-only	0x00000000
0x54	Channel Data Register 1	ADC_CDR1	Read-only	0x00000000
...	...	...	...	...
0x8C	Channel Data Register 15	ADC_CDR15	Read-only	0x00000000
0x90 - 0x90	Reserved	–	–	–
0x94	Analog Control Register	ADC_ACR	Read-write	0x00000100
0x98 - 0xAC	Reserved	–	–	–
0xC4 - 0xE0	Reserved	–	–	–
0xE4	Write Protect Mode Register	ADC_WPMR	Read-write	0x00000000
0xE8	Write Protect Status Register	ADC_WPSR	Read-only	0x00000000
0xEC - 0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–

Note: If an offset is not listed in the table it must be considered as “reserved”.

43.7.1 ADC Control Register

Name: ADC_CR

Address: 0x400C0000

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
–	–	–	–	–	–	START	SWRST

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the ADC simulating a hardware reset.

- **START: Start Conversion**

0 = No effect.

1 = Begins analog-to-digital conversion.

43.7.2 ADC Mode Register

Name: ADC_MR

Address: 0x400C0004

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	LOWRES	TRGSEL			TRGEN

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 1353.

- **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	ADC_TRIG0	External : ADCTRG
1	ADC_TRIG1	TIOA Output of the Timer Counter Channel 0
2	ADC_TRIG2	TIOA Output of the Timer Counter Channel 1
3	ADC_TRIG3	TIOA Output of the Timer Counter Channel 2
4	ADC_TRIG4	PWM Event Line 0
5	ADC_TRIG5	PWM Event Line 0
6	ADC_TRIG6	Reserved
7	–	Reserved

- **LOWRES: Resolution**

Value	Name	Description
0	BITS_12	12-bit resolution
1	BITS_10	10-bit resolution

- **SLEEP: Sleep Mode**

Value	Name	Description
0	NORMAL	Normal Mode: The ADC Core and reference voltage circuitry are kept ON between conversions
1	SLEEP	Sleep Mode: The ADC 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 ADC 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**

  ADCClock = MCK / ((PRESCAL+1) * 2)

- **STARTUP: Start Up Time**

Value	Name	Description
0	SUT0	0 periods of ADCClock
1	SUT8	8 periods of ADCClock
2	SUT16	16 periods of ADCClock
3	SUT24	24 periods of ADCClock
4	SUT64	64 periods of ADCClock
5	SUT80	80 periods of ADCClock
6	SUT96	96 periods of ADCClock
7	SUT112	112 periods of ADCClock
8	SUT512	512 periods of ADCClock
9	SUT576	576 periods of ADCClock
10	SUT640	640 periods of ADCClock
11	SUT704	704 periods of ADCClock
12	SUT768	768 periods of ADCClock
13	SUT832	832 periods of ADCClock
14	SUT896	896 periods of ADCClock
15	SUT960	960 periods of ADCClock

- **SETTLING: Analog Settling Time**

Value	Name	Description
0	AST3	3 periods of ADCClock
1	AST5	5 periods of ADCClock
2	AST9	9 periods of ADCClock
3	AST17	17 periods of ADCClock

- **ANACH: Analog Change**

Value	Name	Description
0	NONE	No analog change on channel switching: DIFF0, GAIN0 and OFF0 are used for all channels
1	ALLOWED	Allows different analog settings for each channel. See ADC_CGR and ADC_COR Registers

- **TRACKTIM: Tracking Time**

  Tracking Time = (TRACKTIM + 1) * ADCClock periods.

- **TRANSFER: Transfer Period**

  Transfer Period = (TRANSFER * 2 + 3) ADCClock 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 ADC_SEQR1 and ADC_SEQR2 registers.

43.7.3 ADC Channel Sequence 1 Register

Name: ADC_SEQR1

Address: 0x400C0008

Access: Read-write

31	30	29	28	27	26	25	24
USCH8				USCH7			
23	22	21	20	19	18	17	16
USCH6				USCH5			
15	14	13	12	11	10	9	8
USCH4				USCH3			
7	6	5	4	3	2	1	0
USCH2				USCH1			

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 1353.

- **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 ADC_MR(USEQ) field is set to ‘1’.

Any USCHx field is taken into account only if ADC_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.

43.7.4 ADC Channel Sequence 2 Register

Name: ADC_SEQR2

Address: 0x400C000C

Access: Read-write

31	30	29	28	27	26	25	24
USCH16				USCH15			
23	22	21	20	19	18	17	16
USCH14				USCH13			
15	14	13	12	11	10	9	8
USCH12				USCH11			
7	6	5	4	3	2	1	0
USCH10				USCH9			

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 1353.

- **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 ADC_MR(USEQ) field is set to ‘1’.

Any USCHx field is taken into account only if ADC_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 ADC Channel Enable Register

Name: ADC_CHER

Address: 0x400C0010

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 “ADC Write Protect Mode Register” on page 1353.

- **CHx: Channel x Enable**

0 = No effect.

1 = Enables the corresponding channel.

Note: if USEQ = 1 in ADC_MR register, CHx corresponds to the xth channel of the sequence described in ADC_SEQR1 and ADC_SEQR2.

43.7.6 ADC Channel Disable Register

Name: ADC_CHDR

Address: 0x400C0014

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 “ADC Write Protect Mode Register” on page 1353.

- **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 ADC_SR are unpredictable.

43.7.7 ADC Channel Status Register

Name: ADC_CHSR

Address: 0x400C0018

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 = Corresponding channel is disabled.

1 = Corresponding channel is enabled.

43.7.8 ADC Last Converted Data Register

Name: ADC_LCDR

Address: 0x400C0020

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
CHNB				LDATA			
7	6	5	4	3	2	1	0
LDATA							

- **LDATA: Last Data Converted**

The analog-to-digital 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 ADC_EMR register. If TAG option is not set, CHNB = 0.

43.7.9 ADC Interrupt Enable Register

Name: ADC_IER

Address: 0x400C0024

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	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**

0 = No effect.

1 = Enables the corresponding interrupt.

43.7.10 ADC Interrupt Disable Register

Name: ADC_IDR

Address: 0x400C0028

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	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**

0 = No effect.

1 = Disables the corresponding interrupt.

43.7.11 ADC Interrupt Mask Register

Name: ADC_IMR

Address: 0x400C002C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	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**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

43.7.12 ADC Interrupt Status Register

Name: ADC_ISR

Address: 0x400C0030

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	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 = Corresponding analog channel is disabled, or the conversion is not finished. This flag is cleared when reading the corresponding ADC_CDRx registers.

1 = Corresponding analog channel is enabled and conversion is complete.

- **DRDY: Data Ready**

0 = No data has been converted since the last read of ADC_LCDR.

1 = At least one data has been converted and is available in ADC_LCDR.

- **GOVRE: General Overrun Error**

0 = No General Overrun Error occurred since the last read of ADC_ISR.

1 = At least one General Overrun Error has occurred since the last read of ADC_ISR.

- **COMPE: Comparison Error**

0 = No Comparison Error since the last read of ADC_ISR.

1 = At least one Comparison Error has occurred since the last read of ADC_ISR.

- **ENDRX: End of RX Buffer**

0 = The Receive Counter Register has not reached 0 since the last write in ADC_RCR or ADC_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in ADC_RCR or ADC_RNCR.

- **RXBUFF: RX Buffer Full**

0 = ADC_RCR or ADC_RNCR have a value other than 0.

1 = Both ADC_RCR and ADC_RNCR have a value of 0.

43.7.13 ADC Overrun Status Register

Name: ADC_OVER

Address: 0x400C003C

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

- **OVREx: Overrun Error x**

0 = No overrun error on the corresponding channel since the last read of ADC_OVER.

1 = There has been an overrun error on the corresponding channel since the last read of ADC_OVER.

43.7.14 ADC Extended Mode Register

Name: ADC_EMR

Address: 0x400C0040

Access: Read-write

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

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 1353.

- **CMPMODE: 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.

- **TAG: TAG of ADC_LDCR register**

0 = set CHNB to zero in ADC_LDCR.

1 = append the channel number to the conversion result in ADC_LDCR register.

43.7.15 ADC Compare Window Register

Name: ADC_CWR

Address: 0x400C0044

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 “ADC Write Protect Mode Register” on page 1353.

- **LOWTHRES: Low Threshold**

Low threshold associated to compare settings of ADC_EMR register.

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of ADC_EMR register.

43.7.16 ADC Channel Gain Register

Name: ADC_CGR

Address: 0x400C0048

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 “ADC Write Protect Mode Register” on page 1353.

- **GAINx: Gain for channel x**

Gain applied on input of analog-to-digital converter.

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 the following register, ADC_COR.

43.7.17 ADC Channel Offset Register

Name: ADC_COR

Address: 0x400C004C

Access: Read-write

31	30	29	28	27	26	25	24
DIFF15	DIFF14	DIFF13	DIFF12	DIFF11	DIFF10	DIFF9	DIFF8
23	22	21	20	19	18	17	16
DIFF7	DIFF6	DIFF5	DIFF4	DIFF3	DIFF2	DIFF1	DIFF0
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 “ADC Write Protect Mode Register” on page 1353.

- **OFFx: Offset for channel x**

0 = No Offset.

1 = center the analog signal on Vrefin/2 before the gain scaling. The Offset applied is: (G-1)Vrefin/2

where G is the gain applied (see description of ADC_CGR register).

- **DIFFx: Differential inputs for channel x**

0 = Single Ended Mode.

1 = Fully Differential Mode.

43.7.18 ADC Channel Data Register

Name: ADC_CDRx [x=0..15]

Address: 0x400C0050

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**

The analog-to-digital conversion data is placed into this register at the end of a conversion and remains until a new conversion is completed. The Convert Data Register (CDR) is only loaded if the corresponding analog channel is enabled.

43.7.19 ADC Analog Control Register

Name: ADC_ACR

Address: 0x400C0094

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
–	–	–	–	–	–	IBCTL	
7	6	5	4	3	2	1	0
–	–	–	TSON	–	–	–	–

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 1353.

- **TSON: Temperature Sensor On**

0 = temperature sensor is off.

1 = temperature sensor is on.

- **IBCTL: ADC Bias Current Control**

Allows to adapt performance versus power consumption. (See the product electrical characteristics for further details.)

43.7.20 ADC Write Protect Mode Register

Name: ADC_WPMR

Address: 0x400C00E4

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:

“ADC Mode Register” on page 1333

“ADC Channel Sequence 1 Register” on page 1336

“ADC Channel Sequence 2 Register” on page 1337

“ADC Channel Enable Register” on page 1338

“ADC Channel Disable Register” on page 1339

“ADC Extended Mode Register” on page 1347

“ADC Compare Window Register” on page 1348

“ADC Channel Gain Register” on page 1349

“ADC Channel Offset Register” on page 1350

“ADC Analog Control Register” on page 1352

- **WPKEY: Write Protect KEY**

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.

43.7.21 ADC Write Protect Status Register

Name: ADC_WPSR

Address: 0x400C00E8

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 ADC_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the ADC_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 ADC_WPSR automatically clears all fields.

44. Digital-to-Analog Converter Controller (DACC)

44.1 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.

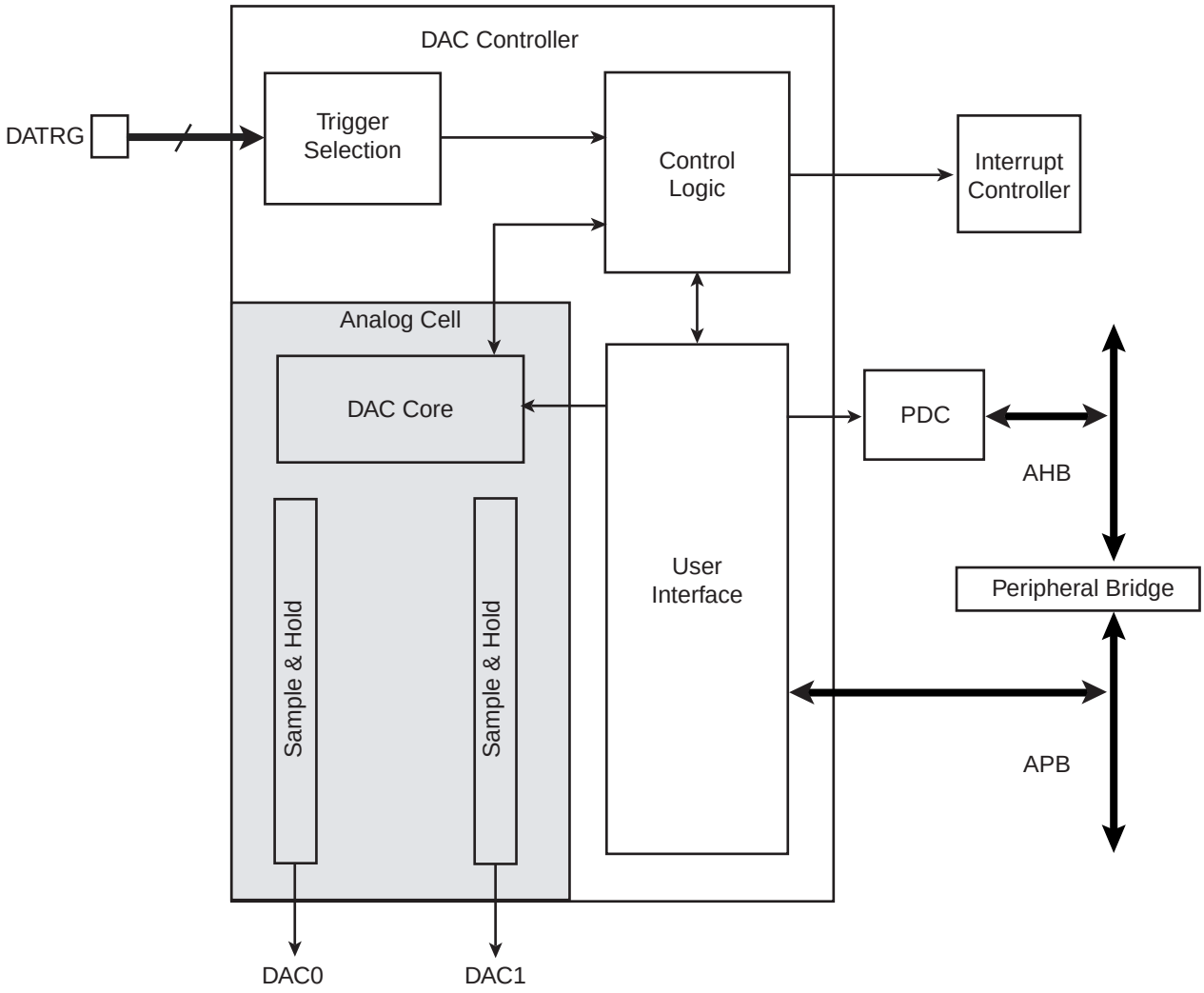
The user can configure DACC timings, such as Startup Time and Refresh Period.

44.2 Embedded Characteristics

- Up to Four 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

44.3 Block Diagram

Figure 44-1. Digital-to-Analog Converter Controller Block Diagram



44.4 Signal Description

Table 44-1. DACC Pin Description

Pin Name	Description
DAC0 - DAC1	Analog output channels
DATRG	External triggers

44.5 Product Dependencies

44.5.1 Power Management

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 44.6.6 ”Sleep Mode”.

44.5.2 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 44-2. Peripheral IDs

Instance	ID
DACC	38

44.5.3 Conversion Performances

For performance and electrical characteristics of the DACC, see the product DC Characteristics section.

44.6 Functional Description

44.6.1 Digital-to-Analog Conversion

The DACC uses the master clock (MCK) divided by two to perform conversions. This clock is named DACC Clock. Once a conversion starts the DACC takes 25 clock periods to provide the analog result on the selected analog output.

44.6.2 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.

44.6.3 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.

44.6.4 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.

44.6.5 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.

44.6.6 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).

44.6.7 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.

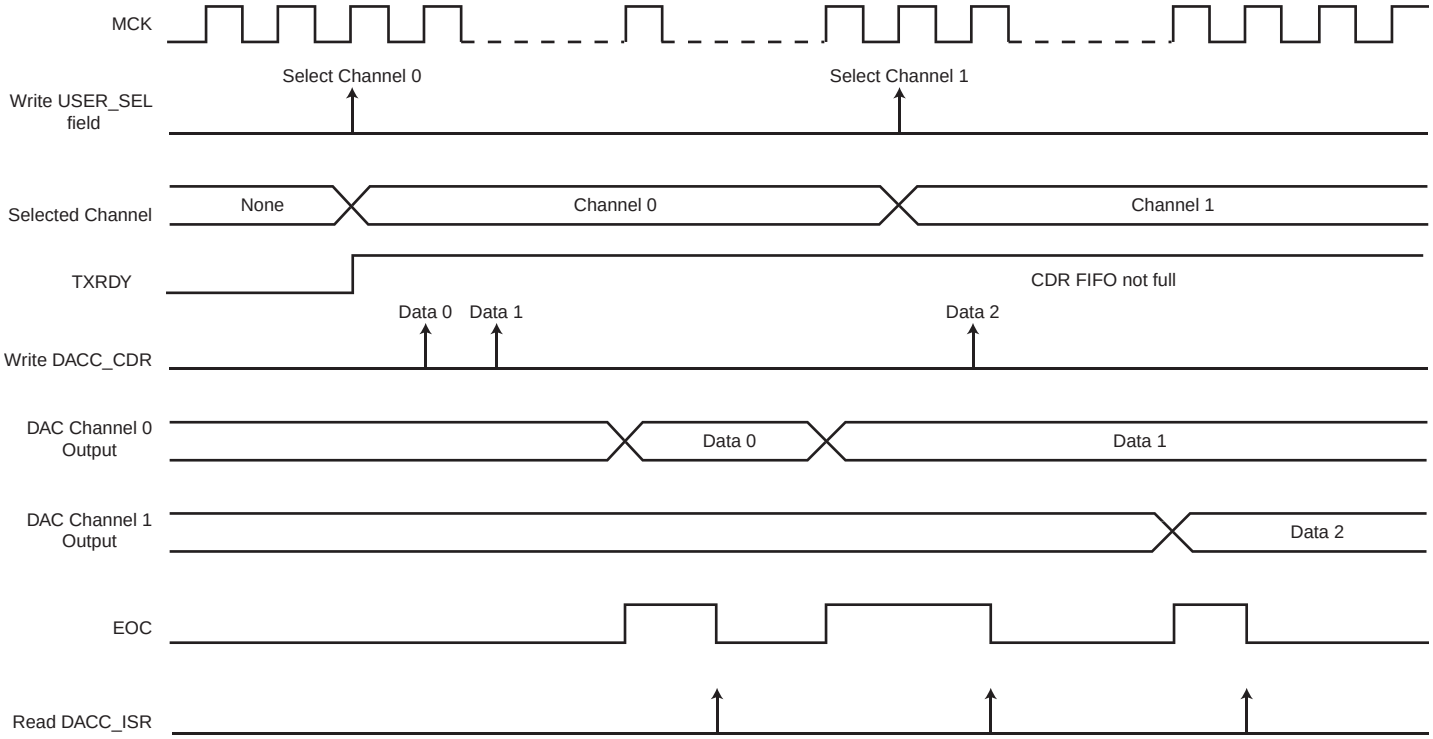
A max speed mode is available by setting the MAXS bit to 1 in the DACC_MR register. Using this mode, the DAC Controller no longer waits to sample the end of cycle signal coming from the DACC block to start the next conversion and uses an internal counter instead. This mode gains 2 DACC Clock periods 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 44-2. Conversion Sequence



44.6.8 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 WPROTADRR 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

44.7 Analog Converter Controller (DACC) User Interface

Table 44-3. 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	–	–	–

44.7.1 DACC Control Register

Name: DACC_CR

Address: 0x400C8000

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.

44.7.2 DACC Mode Register

Name: DACC_MR

Address: 0x400C8004

Access: Read-write

31	30	29	28	27	26	25	24
–	–	STARTUP					
23	22	21	20	19	18	17	16
–	–	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*REFRESH/DACC Clock

- **USER_SEL: User Channel Selection**

Value	Name	Description
0	CHANNEL0	Channel 0
1	CHANNEL1	Channel 1
2		Reserved
3		Reserved

- **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**

MAX SPEED	Selected Mode
0	Normal Mode
1	Max Speed Mode enabled

- **STARTUP: Startup Time Selection**

Value	Name	Description
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
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: Refer to the product DAC electrical characteristics section for Startup Time value.

44.7.3 DACC Channel Enable Register

Name: DACC_CHER

Address: 0x400C8010

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.

44.7.4 DACC Channel Disable Register

Name: DACC_CHDR

Address: 0x400C8014

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.

44.7.5 DACC Channel Status Register

Name: DACC_CHSR

Address: 0x400C8018

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: Corresponding channel is disabled.

1: Corresponding channel is enabled.

44.7.6 DACC Conversion Data Register

Name: DACC_CDR

Address: 0x400C8020

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.

44.7.7 DACC Interrupt Enable Register

Name: DACC_IER

Address: 0x400C8024

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**

44.7.8 DACC Interrupt Disable Register

Name: DACC_IDR

Address: 0x400C8028

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**

44.7.9 DACC Interrupt Mask Register

Name: DACC_IMR

Address: 0x400C802C

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**

44.7.10 DACC Interrupt Status Register

Name: DACC_ISR

Address: 0x400C8030

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.

44.7.11 DACC Analog Current Register

Name: DACC_ACR

Address: 0x400C8094

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.)

44.7.12 DACC Write Protect Mode Register

Name: DACC_WPMR

Address: 0x400C80E4

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 “” for details).

Must be filled with “DAC” ASCII code.

44.7.13 DACC Write Protect Status Register

Name: DACC_WPSR

Address: 0x400C80E8

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.

45. Electrical Characteristics

45.1 Absolute Maximum Ratings

Table 45-1. Absolute Maximum Ratings*

Operating Temperature (Industrial).....-40°C to +85°C
Storage Temperature.....-60°C to +150°C
Voltage on Input Pins with Respect to Ground (except VBUS).....-0.3V to +4.0V
Voltage on VBUS Pin with Respect to Ground.....-0.3V to +6.0V
Maximum Operating Voltage (VDDCORE).....2.0V
Maximum Operating Voltage (VDDIO).....4.0V
Total DC Output Current on all I/O lines 100-lead LQFP.....100 mA 144-lead LQFP.....130 mA 100-ball TFBGA.....150 mA 144-ball LFBGA.....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.**

DRAFT

45.2 DC Characteristics

The following characteristics are applicable to the operating temperature range T_A = -40°C to 85°C, unless otherwise specified.

Table 45-2. DC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDCORE}	DC Supply Core		1.62	1.8	1.95	V
V_{DDIO}	DC Supply I/Os		1.62	3.3	3.6	V
V_{DDBU}	Backup I/O Lines Power Supply		1.62		3.6	V
$V_{DDUTMII}$	USB UTMI+ Interface Power Supply		3.0		3.6	V
V_{DDPLL}	PLL A, UPLL and Main Oscillator Supply		1.62		1.95	V
V_{DDANA}	ADC Analog Power Supply		(1)		(1)	V
V_{IL}	Input Low-level Voltage	PIOA/B/C/D/E/F[0–31]	-0.3		$0.3 \times V_{DDIO}$	V
V_{IH}	Input High-level Voltage	PIOA/B/C/D/E/F[0–31]	$0.7 \times V_{DDIO}$		V_{DDIO} + 0.3V	V
V_{OH}	Output High-level Voltage	PIOA/B/C/D/E/F[0–31] I_{OH} ~ 0 I_{OH} > 0 (See I_{OH} details below)	 V_{DDIO} - 0.2V V_{DDIO} - 0.4V			V
V_{OL}	Output Low-level Voltage	PIOA/B/C/D/E/F[0–31] I_{OH} ~ 0 I_{OH} > 0 (See I_{OL} details below)			 0.2 0.4	V
V_{hys}	Hysteresis Voltage	PIOA/B/C/D/E/F[0–31] except PA0, PA9, PA26, PA29, PA30, PA31, PB14, PB22, PC[2–9], PC[15–24], PD[10–30], PE[0–4], PE15, PE17, PE19, PE21, PE23, PE25, PE29	150		500	mV
		ERASE, TST, FWUP, JTAGSEL	230		700	mV
I_{OH}	Source Current	1.62V < VDDIO < 1.95V; $V_{OH} = V_{DDIO}$ - 0.4 - Group 1(2) - Group 2(3)			 -8 -3	mA
		3.0V < VDDIO < 3.6V; $V_{OH} = V_{DDIO}$ - 0.4 - Group 1(2) - Group 2(3)			 -15 -3	
		1.62V < VDDIO < 3.6V; $V_{OH} = V_{DDIO}$ - 0.4 - NRST, TDO			 -2	
		Relaxed Mode: 3.0V < VDDIO < 3.6V; V_{OH} = 2.2V - Group 1(2) - Group 2(3)			 -24 -9	

DRAFT

Table 45-2. DC Characteristics (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{OL}	Sink Current	1.62V < VDDIO < 1.95V; V_{OL} = 0.4V - Group 1[2] - Group 2[3]			 8 4	mA
		3.0V < VDDIO < 3.6V; V_{OL} = 0.4V - Group 1[2] - Group 2[3]			 9 6	
		1.62V < VDDIO < 3.6V; V_{OL} = 0.4V - NRST, TDO			 2	
		Relaxed Mode: 3.0V < VDDIO < 3.6V; V_{OL} = 0.6V - Group 1[2] - Group 2[3]			 14 9	
I_{IL_lkg}	Input Low Leakage Current	V_{DDIO} powered pins[3] No pull-up or pull-down; V_{IN} = GND; V_{DDIO} Max. (Typ: T_A = 25°C, Max: T_A = 85°C)		5	30	nA
		V_{DDBU} powered pins[4] No pull-up or pull-down; V_{IN} = GND; V_{DDBU} Max. (Typ: T_A = 25°C, Max: T_A = 85°C)			1	µA
I_{IH_lkg}	Input High Leakage Current	V_{DDIO} powered pins[3] No pull-up or pull-down; V_{IN} = VDD; V_{DDIO} Max. (Typ: T_A = 25°C, Max: T_A = 85°C)		2	18	nA
		V_{DDBU} powered pins[4] No pull-up or pull-down; V_{IN} = VDD; V_{DDBU} Max. (Typ: T_A = 25°C, Max: T_A = 85°C)			1	µA
R_{PULLUP}	Pull-up Resistor	PA0–PA31, PB0–PB31, PC0–PC30, PD0–PD30, PE0–PE31, PF0–PF5	50	100	150	kΩ
		NRSTB	10		20	kΩ
$R_{PULLDOWN}$	Pull-down Resistor	TST, ERASE, JTAGSEL	10		20	kΩ
R_{ODT}	On-die Series Termination Resistor	PA0–PA31, PB0–PB31, PC0–PC30, PD0–PD30, PE0–PE31, PF0–PF5		36		Ω

Notes:
1. Refer to Section 45.7 “12-bit ADC Characteristics” and Section 45.9 “12-bit DAC Characteristics”
2. PA0, PA7, PA9, PA[14–15], PA[18–19], PA[25–31], PB[0–3], PB8, PB[10–11], PB14, PB[22–24], PC[0–30], PD[0–30], PE[0–9], PE15, PE[17–21], PE23, PE25, PF[0–2]
3. PA[1–6], PA8, PA[10–13], PA[16–17], PA[20–24], PB[4–7], PB9, PB[12–13], PB[15–21], PB[25–31], PE[10–14], PE16, PE22, PE24, PE26, PF[3–5]
4. FWUP, JTAGSEL, NRSTB, ERASE, TST

DRAFT

Table 45-3. 1.8V Voltage Regulator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDIN}	DC Input Voltage Range		1.8	3.3	3.6	V
V_{DDOUT}	DC Output Voltage	Normal Mode		1.8		V
		Standby Mode		0		
$V_{O(accuracy)}$	Output Voltage Accuracy	I_{LOAD} = 0.5–150 mA	-3		3	%
I_{LOAD}	Maximum DC Output Current	V_{DDIN} > 2.2V			150	mA
		$V_{DDIN} \leq$ 2.2V			60	
$I_{LOAD-START}$	Maximum Peak Current during startup	(3)			300	mA
$V_{DROPOUT}$	Dropout Voltage	V_{DDIN} = 1.8V I_{LOAD} = 60 mA		120	240	mV
V_{LINE}	Line Regulation	V_{DDIN} from 2.7V to 3.6V I_{LOAD} MAX		20	50	mV
$V_{LINE-TR}$	Transient Line regulation	V_{DDIN} from 2.7V to 3.6V $t_r = t_f$ = 5 µs I_{LOAD} Max CD_{OUT} = 4.7 µF		50	100	mV
V_{LOAD}	Load Regulation	$V_{DDIN} \geq$ 2.2V I_{LOAD} = 10% to 90% MAX		20	50	mV
$V_{LOAD-TR}$	Transient Load Regulation	$V_{DDIN} \geq$ 2.2V I_{LOAD} = 10% to 90% MAX $t_r = t_f$ = 5 µs CD_{OUT} = 4.7 µF		50	100	
I_Q	Quiescent Current	Normal Mode @ I_{LOAD} = 0 mA		7	10	µA
		Normal Mode @ I_{LOAD} = 150 mA		700	1200	
		Standby Mode			1	
CD_{IN}	Input Decoupling Capacitor	(1)		10		µF
CD_{OUT}	Output Decoupling Capacitor	(2)		4.7		µF
		ESR	0.5		10	Ω
t_{on}	Turn on Time	CD_{OUT} = 4.7 µF V_{DDOUT} reaches V_{T+} (core power brownout detector supply rising threshold)		120	250	µs
		CD_{OUT} = 4.7 µF V_{DDOUT} reaches 1.8V (± 3%)		200	400	

Notes:
1. A 10 µF 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 4.7 µF 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 0.5–10 Ω. 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 decreases output noise and improves the load transient response.
3. Defined as the current needed to charge external bypass/decoupling capacitor network.

DRAFT

Table 45-4. Core Power Supply Brownout Detector Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{T-}	Supply Falling Threshold(1)		1.52	1.55	1.58	V
V_{hys-}	Hysteresis V_{T-}			25	38	mV
V_{T+}	Supply Rising Threshold		1.35	1.50	1.59	V
t_{RST}	Reset Period		100		350	µs
V_{hys+}	Hysteresis V_{T+}		100		250	mV
I_{DDON}	Current Consumption on VDDCORE	Brownout Detector enabled		18		µA
I_{DDOFF}		Brownout Detector disabled			200	nA
t_{d-}	V_{T-} detection propagation time	VDDCORE = V_{T+} to (V_{T-} - 100mV)			200	ns
t_{d+}	V_{T+} detection propagation time		100	200	350	µs
t_{START}	Startup Time	From disabled state to enabled state		100	200	µs

Note: 1. The product is guaranteed to be functional at V_{T-}

Figure 45-1. Core Brownout Output Waveform

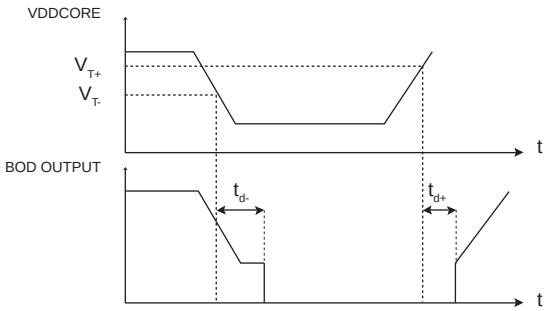


Table 45-5. VDDUTMI Supply Monitor

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_T	Supply Monitor Threshold	16 selectable steps of 100mV	1.9		3.4	V
$V_{T(accuracy)}$	Threshold Level Accuracy		-1.5		+1.5	%
V_{hys}	Hysteresis			20	30	mV
I_{DDON}	Current Consumption on VDDCORE	Enabled		18	28	µA
I_{DDOFF}		Disabled			1	
t_{START}	Startup Time	From disabled state to enabled state			140	µs

DRAFT

Figure 45-2. VDDUTMI Supply Monitor

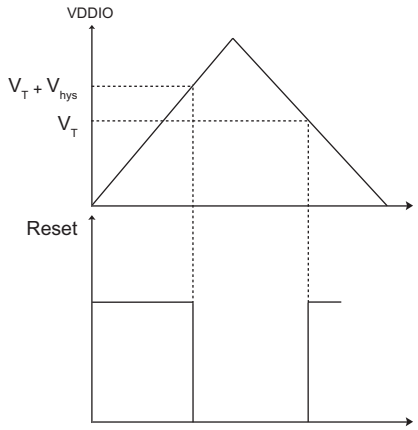


Table 45-6. Threshold Selection

Digital Code	Threshold Min (V)	Threshold Typ (V)	Threshold Max (V)
0000	1.881	1.9	1.919
0001	1.980	2.0	2.020
0010	2.079	2.1	2.121
0011	2.178	2.2	2.222
0100	2.277	2.3	2.323
0101	2.376	2.4	2.424
0110	2.475	2.5	2.525
0111	2.574	2.6	2.626
1000	2.673	2.7	2.727
1001	2.772	2.8	2.828
1010	2.871	2.9	2.929
1011	2.970	3.0	3.030
1100	3.069	3.1	3.131
1101	3.168	3.2	3.232
1110	3.267	3.3	3.333
1111	3.366	3.4	3.434

Table 45-7. Backup Power Supply Zero-Power-on Reset Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{T+}	Threshold Voltage Rising	At startup	1.50	1.55	1.60	V
V_{T-}	Threshold Voltage Falling		1.40	1.45	1.50	V
t_{RST}	Reset Period		40	90	150	µs

DRAFT

Figure 45-3. Zero-Power-on Reset Characteristics

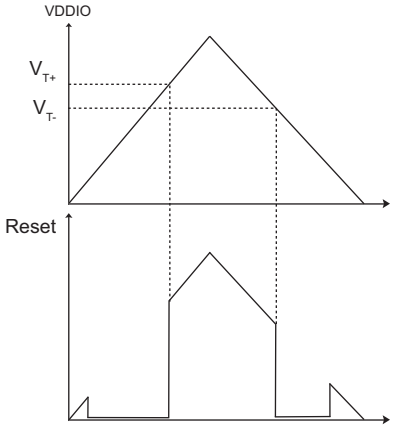


Table 45-8. DC Flash Characteristics

Symbol	Parameter	Conditions	Typ	Max	Unit
$I_{DD(standby)}$	Standby current	@ 25°C onto VDDCORE = 1.8V	< 1	1.5	µA
		@ 85°C onto VDDCORE = 1.8V	14	40	
		@ 25°C onto VDDCORE = 1.95V	< 1	1.8	
		@ 85°C onto VDDCORE = 1.95V	15	50	
I_{CC}	Active current	128-bit Mode Read Access:			mA
		Maximum Read Frequency onto VDDCORE = 1.8V @ 25 °C	15	20	
		Maximum Read Frequency onto VDDCORE = 1.95V @ 25 °C	20	25	
		64-bit Mode Read Access:			
		Maximum Read Frequency onto VDDCORE = 1.8V @ 25 °C	7.5	10	
		Maximum Read Frequency onto VDDCORE = 1.95V @ 25 °C	10	12.5	
		Write onto VDDCORE = 1.8V @ 25 °C	3.6	4.5	
		Write onto VDDCORE = 1.95V @ 25 °C	5.0	6.0	

DRAFT

45.3 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

45.3.1 Backup Mode Current Consumption

The Backup mode configuration and measurements are defined as follows.

45.3.1.1 Backup Power Configuration

- All power supplies OFF, except VDDBU
- Supply Monitor on VDDUTMI is disabled
- RTT and RTC not used
- Embedded RC oscillator used
- Wake-up pin FWUP = VDDBU
- Current measurement on AMP1

Figure 45-4. Measurement Setup

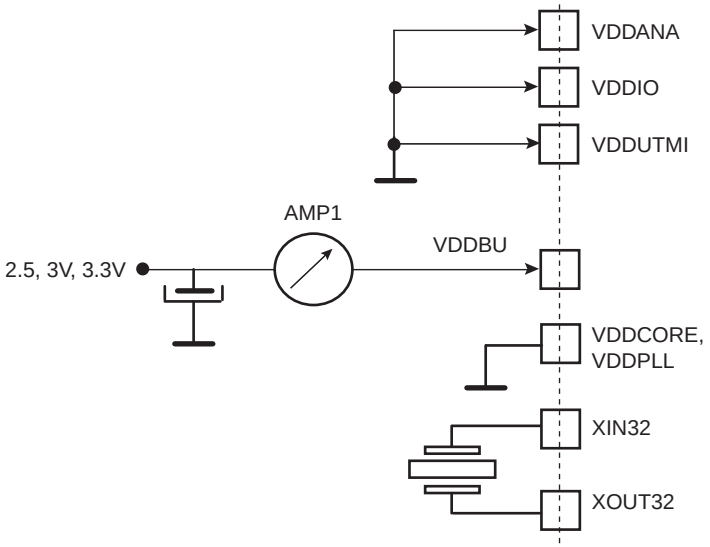


Table 45-9. Power Consumption for Backup Mode

Conditions		VDDBU Consumption (AMP1)	Unit
VDDBU	Temperature		
3.3V	25°C	3.1	µA
3.0V		2.8	
2.5V		2.3	
1.8V		1.7	
3.3V	85°C	4.1	
3.0V		3.7	
2.5V		3.1	
1.8V		2.4	

DRAFT

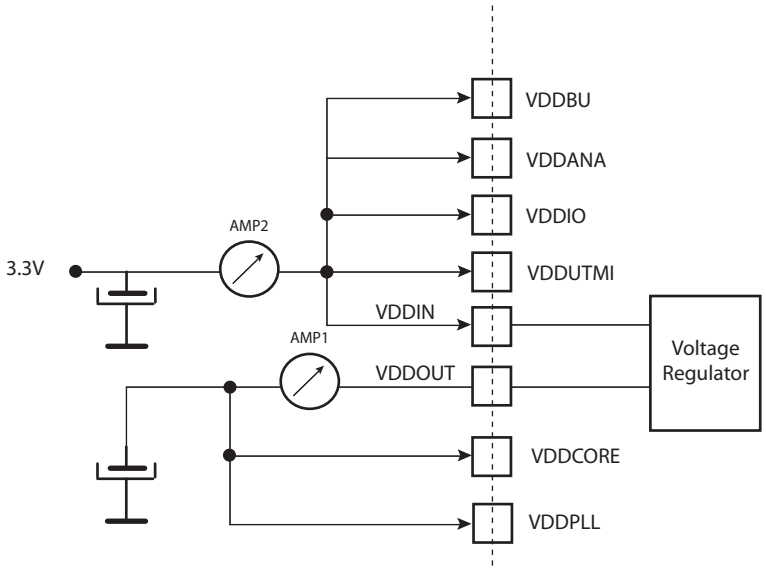
45.3.2 Wait and Sleep Mode Current Consumption

The Wait Mode and Sleep Mode configuration and measurements are defined below.

45.3.2.1 Sleep Mode

- All power supplies are powered
- Core clock OFF
- Master Clock (MCK) running at various frequencies with PLLA or the fast RC oscillator
- Fast startup through pins WKUP0–15
- Current measurement on AMP1 (VDDOUT = VDDCORE + VDDPLL)
- All peripheral clocks deactivated

Figure 45-5. Measurement Setup for Sleep Mode



DRAFT

Figure 45-6. Current Consumption in Sleep Mode (AMP1) versus Master Clock Ranges (refer to Table 45-10)

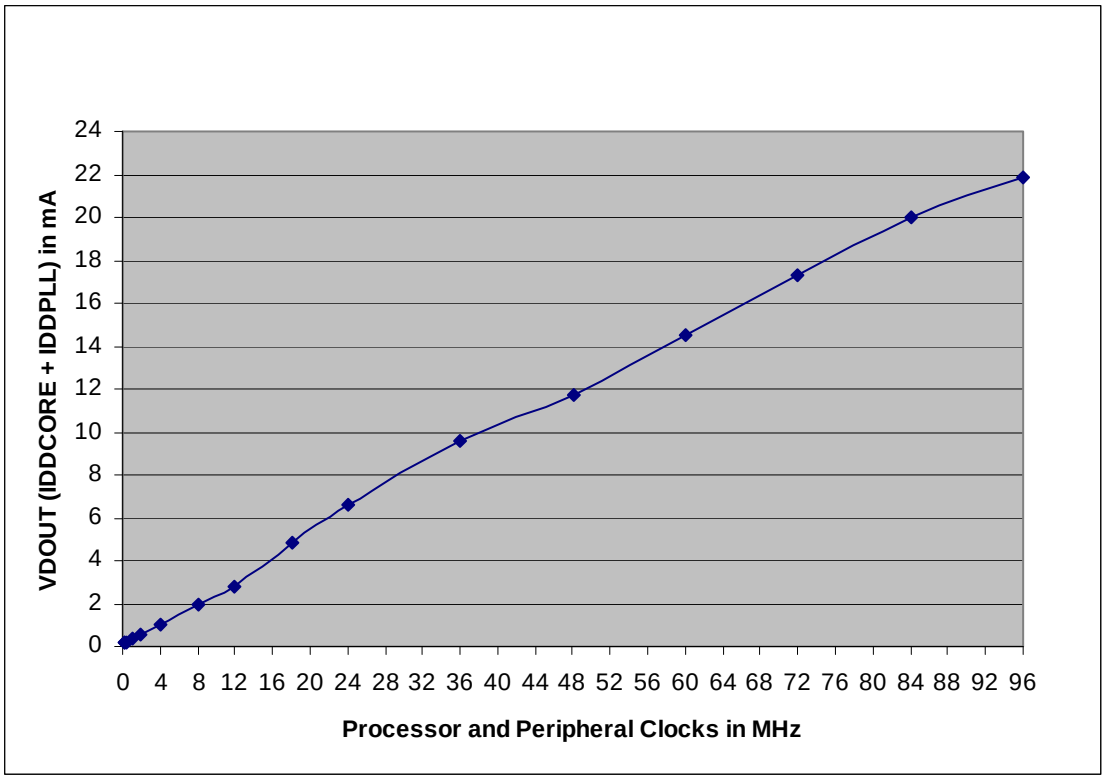


Table 45-10. Sleep Mode Current Consumption versus Master Clock (MCK) Variation

Core Clock/MCK (MHz)	AMP1 (IDDOUT) Consumption	Unit
96	45.9	mA
84	29.4	mA
72	25.6	mA
60	21.7	mA
48	17.8	mA
36	14.5	mA
24	10.0	mA
18	8.6	mA
12	5.9	mA
8	4.92	mA
4	3.52	mA
2	2.37	mA
1	1.31	mA
0.5	0.78	mA
0.25	0.51	mA
0.125	0.38	mA
0.032	0.025	mA

DRAFT

45.3.2.2 Wait Mode

- All power supplies are powered
- Core Clock and Master Clock Stopped
- Current measurement on AMP1, AMP2 and AMP3
- All peripheral clocks deactivated

Figure 45-7. Measurement Setup for Wait Mode

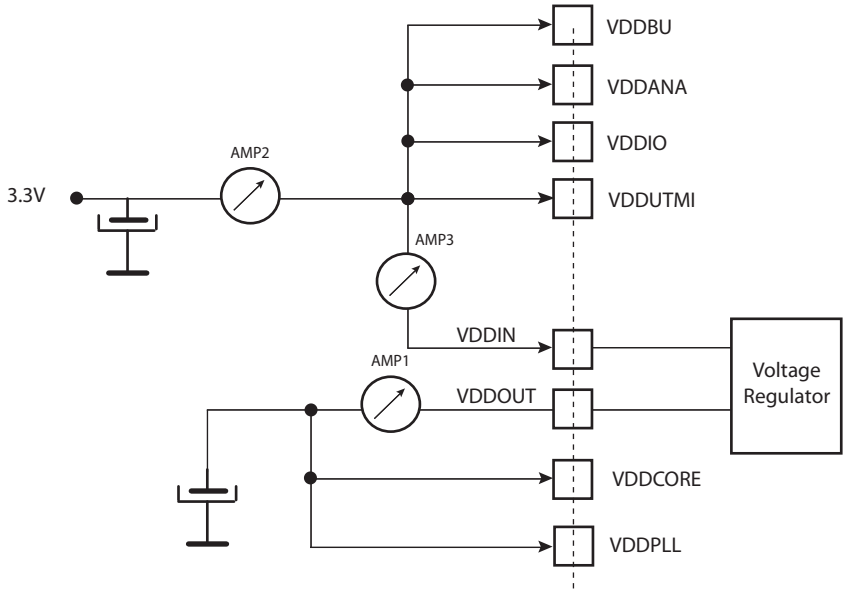


Table 45-11 gives current consumption in typical conditions.

Table 45-11. Typical Current Consumption in Wait Mode

Conditions	Consumption			Unit
	VDDOUT (AMP1)	Total (AMP2)	Regulator and Core (AMP3)	
See Figure 45-7 @ 25°C There is no activity on the I/Os of the device.	18.7	26.6	18.4	µA

45.3.3 Active Mode Power Consumption

The Active Mode configuration and measurements are defined as follows:

- VDDIO = VDDIN = VDDBU= VDDANA = VDUTMI = 3.3V for VDDCORE = 1.8V. VDDOUT not used for VDDCORE = 1.62V
- VDDCORE = 1.8V (internal voltage regulator used)
- T_A = 25°C
- CoreMark Algorithm running from Flash Memory or SRAM
- 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

DRAFT

Figure 45-8. Active Mode Measurement Setup for VDDCORE at 1.8V

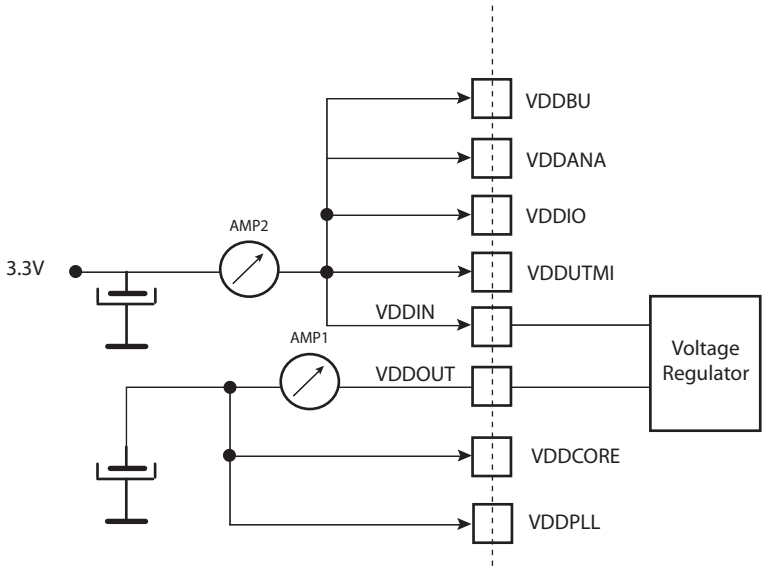
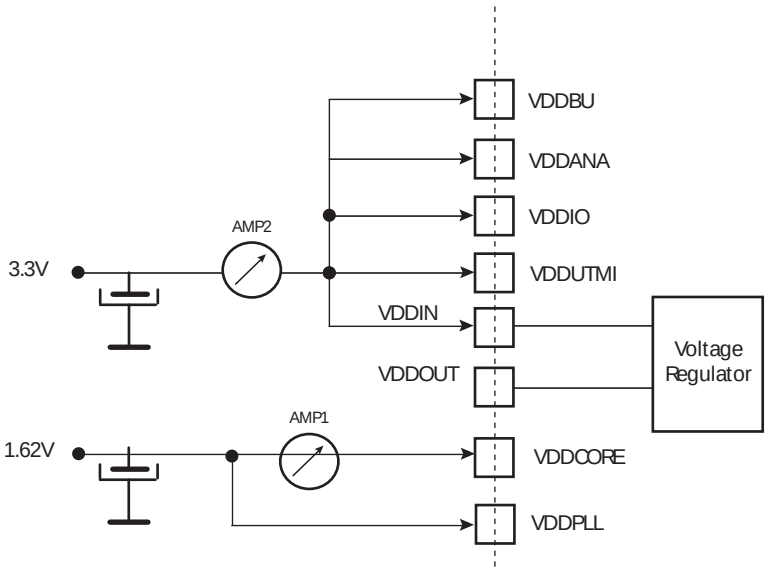


Figure 45-9. Active Mode Measurement Setup for VDDCORE at 1.62V



The following tables give Active mode current consumption in typical conditions.

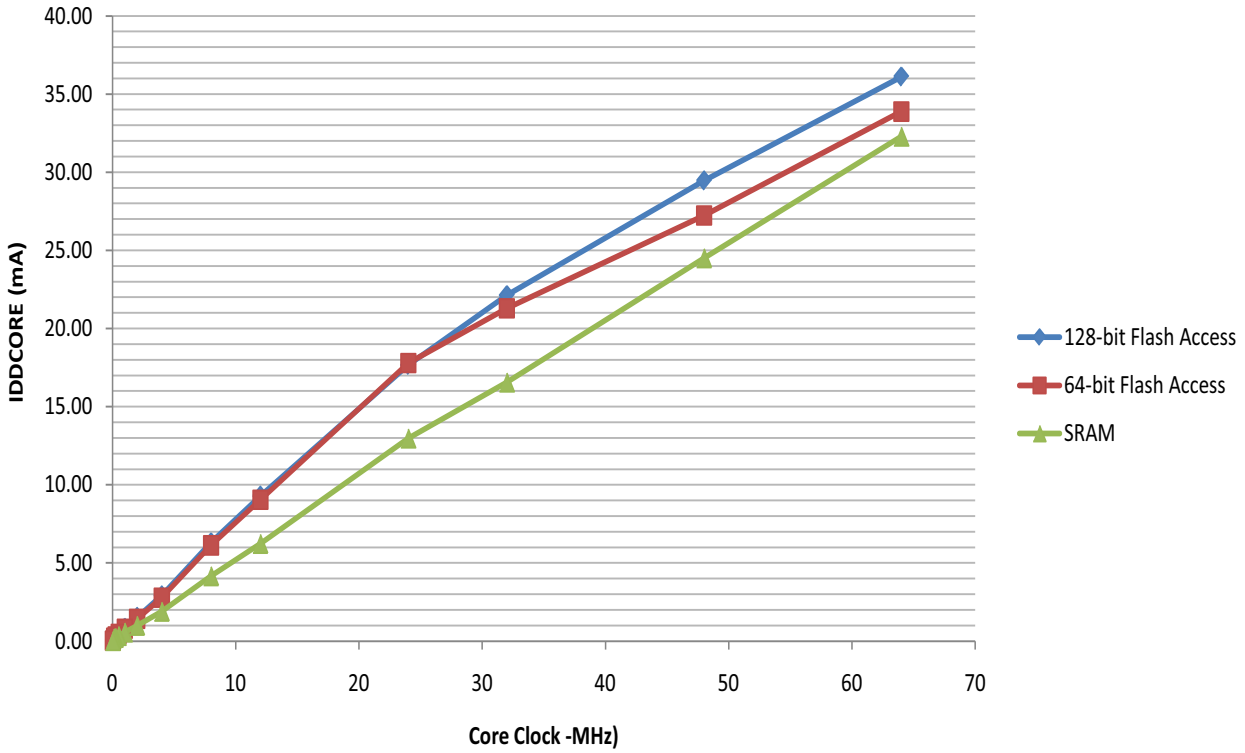
DRAFT

Table 45-12. Active Power Consumption with VDDCORE @ 1.8V Running from Flash Memory or SRAM

	CoreMark						
	128-bit Flash Access(1)		64-bit Flash Access(1)		SRAM		
Core Clock (MHz)	AMP1	AMP2	AMP1	AMP2	AMP1	AMP2	Unit
84	63.25	76.47	56.08	70.89	64.20	77.50	mA
72	57.63	70.83	51.71	66.86	49.40	62.80	
60	53.66	65.48	47.79	62.72	41.80	55.20	
48	46.55	57.57	44.70	55.65	34.20	47.60	
32	35.44	47.75	31.71	45.21	24.30	38.00	
24	29.59	42.44	26.11	39.19	18.90	32.60	
12	16.55	32.02	16.26	29.51	11.10	24.90	
8	13.40	28.80	12.66	27.51	8.70	22.70	
4	10.68	25.46	10.66	24.11	6.10	20.20	
2	12.88	23.58	11.53	22.56	6.40	19.50	
1	7.87	21.37	7.21	20.13	4.60	17.70	
0.5	5.939	18.239	5.321	17.63	3.4	16.7	
0.25	3.72	16.27	3.54	15.92	2.90	16.10	
0.125	3.12	15.41	2.89	15.24	2.70	15.90	
0.032	2.17	14.43	2.17	14.60	2.20	15.50	

Note: 1. Flash Wait State (FWS) in EEFC_FMR adjusted versus Core Frequency

Figure 45-10. Active Power Consumption with VDDCORE @ 1.8V Running from Flash Memory or SRAM



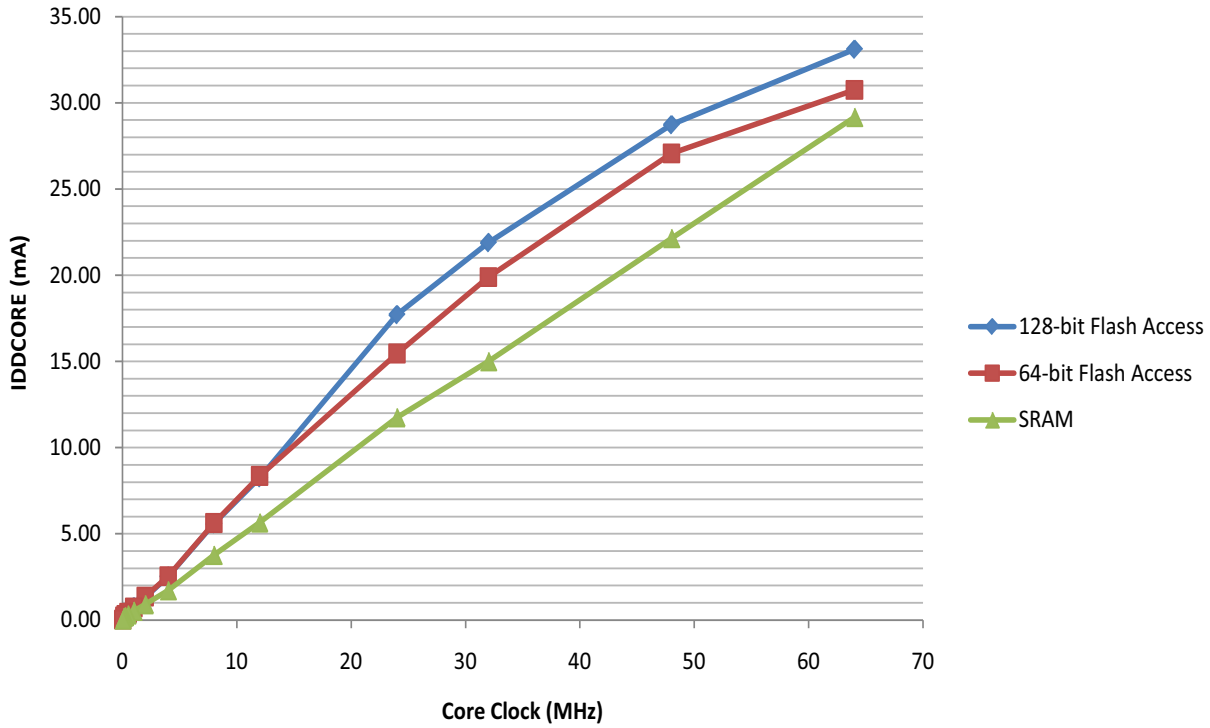
DRAFT

Table 45-13. Active Power Consumption with VDDCORE @ 1.62V Running from Flash Memory or SRAM

	CoreMark						
	128-bit Flash Access[2]		64-bit Flash Access[2]		SRAM		
Core Clock (MHz)	AMP1	AMP2[1]	AMP1	AMP2[1]	AMP1	AMP2[1]	Unit
84	57.66	12.32	51.41	12.32	50.10	13.30	mA
72	51.46	12.32	46.18	12.25	43.60	13.30	
60	45.98	12.27	41.97	12.30	36.80	13.30	
48	40.90	12.28	36.08	12.34	30.10	13.30	
32	31.45	12.32	27.94	12.26	21.30	13.30	
24	24.73	12.39	22.32	12.30	16.50	13.30	
12	15.32	12.37	14.72	12.30	9.60	13.30	
8	12.16	12.28	11.65	12.32	7.50	13.30	
4	8.79	12.30	8.01	12.29	5.20	13.30	
2	10.57	12.32	9.57	12.28	5.50	13.30	
1	6.83	12.30	6.05	12.27	3.70	13.30	
0.5	4.59	12.26	4.30	12.31	2.80	13.30	
0.25	3.42	12.25	3.12	12.29	2.40	13.30	
0.125	2.57	12.30	2.47	12.27	2.20	13.30	
0.032	1.74	12.35	1.73	12.34	1.80	13.30	

Notes: 1. Internal voltage regulator not used. VDDIO power consumption only.
2. Flash Wait State (FWS) in EEFC_FMR adjusted versus Core Frequency

Figure 45-11. Active Power Consumption with VDDCORE @ 1.62V Running from Flash Memory or SRAM



DRAFT

45.3.4 Peripheral Power Consumption in Active Mode

Table 45-14. Power Consumption on VDDCORE(1)

Peripheral	Consumption (Typ)	Unit
UART	11.96	µA/MHz
PIOA	9.23	µA/MHz
PIOB	9.63	µA/MHz
PIOC	10.16	µA/MHz
PIOD	9.72	µA/MHz
PIOE	11.20	µA/MHz
PIOF	2.58	µA/MHz
USART0	31.15	µA/MHz
USART1	24.64	µA/MHz
USART2	25.31	µA/MHz
USART3	23.98	µA/MHz
HSMCI	30.54	µA/MHz
PWM	65.76	µA/MHz
SSC	15.00	µA/MHz
TWI0	17.03	µA/MHz
TWI1	17.29	µA/MHz
SPI0	3.55	µA/MHz
SPI1	2.56	µA/MHz
TC0	13.65	µA/MHz
TC1	8.41	µA/MHz
TC2	7.66	µA/MHz
TC3	12.08	µA/MHz
TC4	8.01	µA/MHz
TC5	7.58	µA/MHz
TC6	12.87	µA/MHz
TC7	7.91	µA/MHz
TC8	8.00	µA/MHz
ADC	22.62	µA/MHz
DAC	12.34	µA/MHz
DMAC	106.41	µA/MHz
SMC	81.54	µA/MHz
CAN0	53.87	µA/MHz
CAN1	54.75	µA/MHz
EMAC	150.35	µA/MHz
TRNG	2.87	µA/MHz

DRAFT

Table 45-14. Power Consumption on VDDCORE(1)

Peripheral	Consumption (Typ)	Unit
USB	73.63	µA/MHz

Note: 1. VDDIO = 3.3V, VDDCORE = 1.80V, T_A = 25°C

DRAFT

45.4 Crystal Oscillators Characteristics

45.4.1 32 kHz RC Oscillator Characteristics

Table 45-15. 32 kHz RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{OSC}	RC Oscillator Frequency		20	32	44	kHz
	Frequency Supply Dependency		-3		3	%/V
	Frequency Temperature Dependency	Over temperature range -40 to 85 °C versus T_A 25°C	-11		11	%
Duty	Duty Cycle		45	50	55	%
t_{START}	Startup Time				100	µs
I_{DDON}	Current Consumption	After startup time Temp. range = -40 to 85 °C Typical consumption at 2.2V supply and T_A = 25°C		540	870	nA

45.4.2 4/8/12 MHz RC Oscillators Characteristics

Table 45-16. 4/8/12 MHz RC Oscillators Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{OSC}	RC Oscillator Frequency Range	(1)	4		12	MHz
ACC_4	4 MHz Total Accuracy	-40°C < Temp < +85°C 4 MHz output selected (1)(2)			±35	%
ACC_8	8 MHz Total Accuracy	-40°C < Temp < +85°C 8 MHz output selected (1)(3)			±3.5	%
		-20°C < Temp < +85°C 8 MHz output selected (1)(3)			±2.5	%
		0°C < Temp < +70°C 8 MHz output selected (1)(3)			±2	%
ACC_{12}	12 MHz Total Accuracy	-40°C < Temp < +85°C 12 MHz output selected (1)(3)			±3.5	%
		-20°C < Temp < +85°C 12 MHz output selected (1)(3)			±2.7	%
		0°C < Temp < +70°C 12 MHz output selected (1)(3)			±2	%
Duty	Duty Cycle		45	50	55	%
t_{START}	Startup Time				10	µs
I_{DDON}	Active Current Consumption	4 MHz 8 MHz 12 MHz		80 105 145	120 160 210	µA

Notes: 1. Frequency range can be configured in the Supply Controller registers.
2. Not trimmed from factory.
3. After trimming from factory.

DRAFT

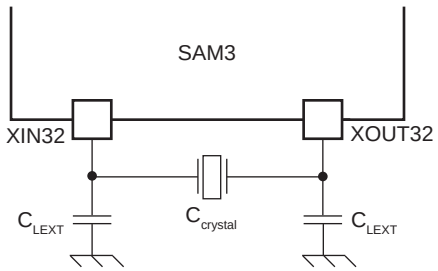
45.4.3 32.768 kHz Crystal Oscillator Characteristics

Table 45-17. 32.768 kHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
f_{OSC}	Operating Frequency	Normal mode with crystal				32.768	kHz
$V_{rip(VDDBU)}$	Supply Ripple Voltage (on VDDBU)	RMS value, 10 kHz to 10 MHz				30	mV
	Duty Cycle			40	50	60	%
t_{START}	Startup Time	R_S < 50 kΩ (1)	$C_{crystal}$ = 12.5 pF			900	ms
			$C_{crystal}$ = 6 pF			300	
		R_S < 100 kΩ (1)	$C_{crystal}$ = 12.5 pF			1200	
			$C_{crystal}$ = 6 pF			500	
I_{DDON}	Current Consumption	R_S < 50 kΩ (1)	$C_{crystal}$ = 12.5 pF		650	1400	nA
			$C_{crystal}$ = 6 pF		450	1200	
		R_S < 100 kΩ (1)	$C_{crystal}$ = 12.5 pF		900	1600	
			$C_{crystal}$ = 6 pF		650	1400	
I_{DDST}	Standby Current Consumption	Standby mode @ 3.6V				5	nA
P_{ON}	Drive level					0.1	µW
R_f	Internal resistor	Between XIN32 and XOUT32			10		MΩ
C_{LEXT}	Maximum external capacitor on XIN32 and XOUT32					20	pF
$C_{crystal}$	Allowed Crystal Capacitance Load	From crystal specification		6		12.5	pF
C_{para}	Internal Parasitic Capacitance			1.2	1.4	1.6	pF

Notes: 1. R_S is the series resistor.

Figure 45-12. 32.768 kHz Crystal Oscillator Schematics



$C_{LEXT} = 2 \times (C_{crystal} - C_{para} - C_{PCB})$

where:

C_{PCB} is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM3 pin.

DRAFT

45.4.4 32.768 kHz Crystal Characteristics

Table 45-18. 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		0.6		3	fF
C_{SHUNT}	Shunt capacitance		0.6		2	pF

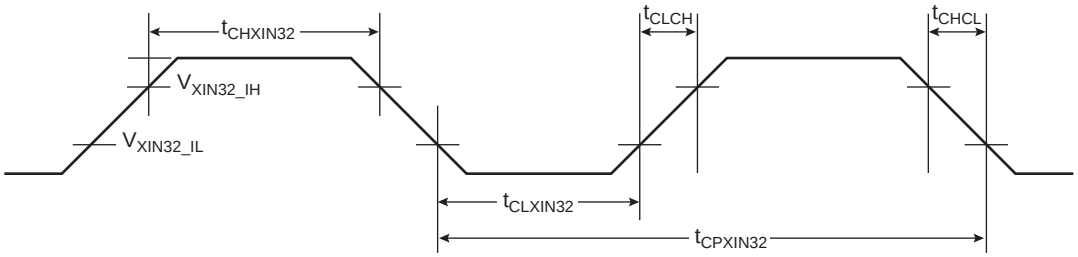
45.4.5 32.768 kHz XIN32 Clock Input Characteristics in Bypass Mode

Table 45-19. XIN32 Clock Electrical Characteristics (In Bypass Mode)

Symbol	Parameter	Conditions	Min	Max	Unit
$1/(t_{CPXIN32})$	XIN32 Clock Frequency	(1)		44	kHz
$t_{CPXIN32}$	XIN32 Clock Period	(1)	22		µs
$t_{CHXIN32}$	XIN32 Clock High Half-period	(1)	11		µs
$t_{CLXIN32}$	XIN32 Clock Low Half-period	(1)	11		µs
t_{CLCH}	Rise Time		400		ns
t_{CHCL}	Fall Time		400		ns
C_{IN}	XIN32 Input Capacitance			6	pF
R_{IN}	XIN32 Pull-down Resistor		3	5	MΩ
V_{XIN32_IL}	V_{XIN32} Input Low-level Voltage		-0.3	$0.3 \times V_{DDBU}$	V
V_{XIN32_IH}	V_{XIN32} Input High-level Voltage		$0.7 \times V_{DDBU}$	$V_{DDBU} + 0.3$	V

Note: 1. These characteristics apply only when the 32.768 kHz crystal oscillator is in Bypass mode (i.e., when SUPC_MR.OSCBYPASS = 1 and SUPC_CR.XTALSEL = 1.)

Figure 45-13. XIN32 Clock Timing



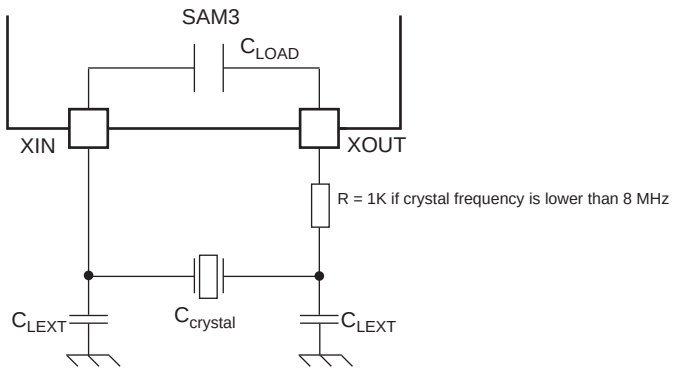
DRAFT

45.4.6 3 to 20 MHz Crystal Oscillator Characteristics

Table 45-20. 3 to 20 MHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{OSC}	Operating Frequency	Normal mode with crystal	3	16	20	MHz
$f_{OSC(bypass)}$	Operating Frequency In Bypass Mode	External Clock on XIN			50	MHz
$V_{rip(VDDPLL)}$	Supply Ripple Voltage (on VDDPLL)	RMS value, 10 kHz to 10 MHz			30	mV
	Duty Cycle		40	50	60	%
t_{START}	Startup Time	3 MHz, C_{SHUNT} = 3 pF 8 MHz, C_{SHUNT} = 7 pF 12 to 16 MHz, C_{SHUNT} = 7 pF 20 MHz, C_{SHUNT} = 7 pF			14.5 4 1.4 1	ms
I_{DDON}	Current Consumption	3 MHz 8 MHz 12 to 16 MHz 20 MHz		150 150 300 400	250 250 450 550	µA
$I_{DD(standby)}$	Standby Current Consumption	Standby mode @ 3.6V			5	nA
P_{ON}	Drive Level	3 MHz 8 MHz 12 MHz, 16 MHz, 20 MHz			15 30 50	µW
R_f	Internal Resistor	Between XIN and XOUT		1		MΩ
C_{LEXT}	Maximum external capacitor on XIN and XOUT				10	pF
$C_{crystal}$	Allowed Crystal Capacitance Load	From crystal specification	12.5		17.5	pF
C_{LOAD}	Internal Equivalent Load Capacitance	Integrated Load Capacitance (XIN and XOUT in series)	7.5	9.5	12	pF

Figure 45-14. 3 to 20 MHz Crystal Oscillator Schematic



$C_{LEXT} = 2 \times (C_{crystal} - C_{LOAD} - C_{PCB})$

where:

C_{PCB} is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM3 pin.

DRAFT

45.4.7 3 to 20 MHz Crystal Characteristics

Table 45-21. Crystal Characteristics

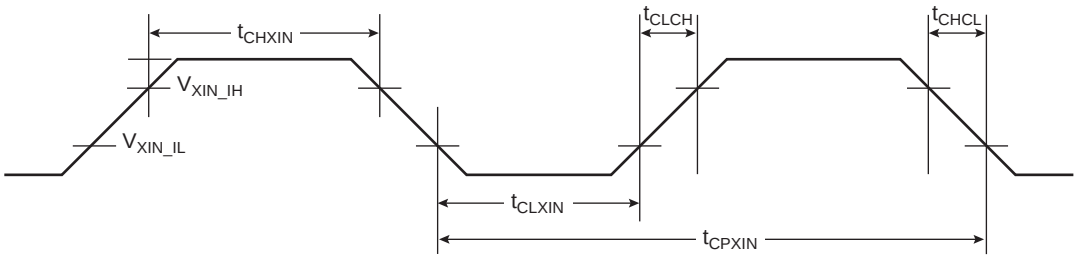
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor (R_S)	Fundamental @ 3 MHz			200	Ω
		Fundamental @ 8 MHz			100	
		Fundamental @ 12 MHz			80	
		Fundamental @ 16 MHz			80	
		Fundamental @ 20 MHz			50	
C_m	Motional capacitance				8	fF
C_{SHUNT}	Shunt capacitance				7	pF

45.4.8 3 to 20 MHz XIN Clock Input Characteristics in Bypass Mode

Table 45-22. XIN Clock Electrical Characteristics (In Bypass Mode)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CPXIN})$	XIN Clock Frequency	3–20 MHz crystal oscillator in Bypass mode			50	MHz
t_{CPXIN}	XIN Clock Period		20			ns
t_{CHXIN}	XIN Clock High Half-period		8			ns
t_{CLXIN}	XIN Clock Low Half-period		8			ns
t_{CLCH}	Rise Time		400			ns
t_{CHCL}	Fall Time		400			ns
C_i	XIN Input Capacitance				6	pF
R_{IN}	XIN Pull-down Resistor			1		MΩ
V_{XIN_IL}	V_{XIN} Input Low-level Voltage		-0.3		$0.3 \times V_{DDPLL}$	V
V_{XIN_IH}	V_{XIN} Input High-level Voltage		$0.7 \times V_{DDPLL}$		$V_{DDPLL} + 0.3$	V

Figure 45-15. XIN Clock Timing



DRAFT

45.4.9 Crystal Oscillator Design Consideration Information

45.4.9.1 Choosing a Crystal

When choosing a crystal for the 32.768 kHz Slow Clock Oscillator or for the 3–20 MHz Oscillator, several parameters must be taken into account. Important parameters between crystal and SAM3 specifications are as follows:

- Load Capacitance

  $C_{crystal}$ is the equivalent capacitor value the oscillator must “show” to the crystal in order to oscillate at the target frequency. Crystal must be chosen according to the internal load capacitance (C_{LOAD}) of the on-chip oscillator. Having a mismatch for the load capacitance will result in a frequency drift.

- Drive Level

  Crystal drive level ≥ Oscillator Drive Level. Having a crystal drive level number lower than the oscillator specification may damage the crystal.

- Equivalent Series Resistor (ESR)

  Crystal ESR ≤ 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 ≤ Oscillator Shunt Capacitance (C_{SHUNT}). Having a crystal with ESR value higher than the oscillator may cause the oscillator to not start.

45.4.9.2 Printed Circuit Board (PCB)

SAM3 oscillators are low-power oscillators requiring particular attention when designing PCB systems.

DRAFT

45.5 UPLL, PLLA Characteristics

Table 45-23. Supply Voltage Phase Lock Loop Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDPLL}	Supply Voltage		1.6	1.8	1.95	V
$V_{rip(VDDPLL)}$	Allowable Voltage Ripple	RMS value 10 kHz to 10 MHz RMS value > 10 MHz			30 10	mV

Table 45-24. PLLA Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{IN}	Input Frequency		1		40	MHz
f_{OUT}	Output Frequency		80		240	MHz
I_{PLL}	Current Consumption	Active mode @ 96 MHz @ 1.8V Active mode @ 160 MHz @ 1.8V Active mode @ 192 MHz @ 1.8V		1 1.6 2.4	1.3 2 3	mA
		Standby mode			1	µA
t_s	Settling Time				200	µs

Table 45-25. UPLL Characteristics for USB High Speed Device Port

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{IN}	Input Frequency			12		MHz
$f_{IN(accuracy)}$	Input Frequency Accuracy	(1)	-0.05		+0.05	%
f_{OUT}	Output Frequency			480		MHz
I_{PLL}	Current Consumption	Active mode @ 480 MHz @ 1.8V		2.5	5	mA
		Standby mode			2	µA

Note: 1. Required for 12 MHz Clock Signal injection on XIN pin (oscillator in Bypass mode).

DRAFT

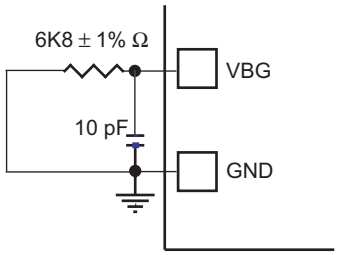
45.6 USB On-the-Go High Speed Port

45.6.1 Typical Connection

For a typical connection, refer to Section 39. “USB On-The-Go Interface (UOTGHS)”.

For an external connection of the VBG pin, see the figure below.

Figure 45-16. External Connection of VBG Pin



45.6.2 Electrical Characteristics

Table 45-26. Electrical Parameters

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
R_{PUI}	Bus Pull-up Resistor on Upstream Port (idle bus)	In FS or HS Mode		1.5		kΩ
R_{PUA}	Bus Pull-up Resistor on Upstream Port (upstream port receiving)	In FS or HS Mode		15		kΩ

45.6.2.1 USB Transceiver

USB 2.0 Compliant in full-speed and high-speed modes. Refer to Chapter 7 of the USB 2.0, Revision 2.0 April 27, 2000.

45.6.2.2 VBUS Pin Characteristics

Compliant with On-The-Go and Embedded Host Supplement to the USB Revision 2.0 Specification Revision 2.0 version 1.1a.

45.6.3 Static Power Consumption

Table 45-27. Static Power Consumption

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{BIAS}	Bias current consumption on VBG				1	µA
$I_{VDDUTMII}$	HS Transceiver & I/O current consumption				8	µA
	FS / HS Transceiver & I/O current consumption	No connection(1)			3	µA

Note: 1. If cable is connected add 200 µA (Typical) due to Pull-up/Pull-down current consumption.

DRAFT

45.6.4 Dynamic Power Consumption

Table 45-28. Dynamic Power Consumption

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{BIAS}	Bias current consumption on VBG			0.7	0.8	mA
$I_{VDDUTMII}$	HS Transceiver current consumption	HS transmission		47	60	mA
	HS Transceiver current consumption	HS reception		18	27	mA
	FS/HS Transceiver current consumption	FS transmission 0m cable(1)		4	6	mA
	FS/HS Transceiver current consumption	FS transmission 5m cable(1)		26	30	mA
	FS/HS Transceiver current consumption	FS reception(1)		3	4.5	mA

Note: 1. Including 1 mA due to Pull-up/Pull-down current consumption.

DRAFT

45.7 12-bit ADC Characteristics

Table 45-29. Analog Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDIN}	ADC Analog Supply	12-bit or 10-bit resolution	2.4	3.0	3.6	V
		10-bit resolution	2.0		3.6	V
$V_{rip(max)}$	Max. Voltage Ripple	RMS value, 10 kHz to 20 MHz			20	mV
I_{VDDIN}	Current Consumption	Sleep Mode		0.1	1	µA
		Fast Wake Up Mode		1.8	2.6	mA
		Normal Mode (ADC_ACR.IBCTL = 00)(1)		4.7	7.1	mA
		Normal Mode (ADC_ACR.IBCTL = 01)(2)		6	9	mA

Notes: 1. Use ADC_ACR.IBCTL = 00 for sampling frequency below 500 kHz.
2. Use ADC_ACR.IBCTL = 01 for sampling frequency between 500 kHz and 1 MHz.

Table 45-30. Channel Conversion Time and ADC Clock

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{ADC}	ADC Clock Frequency		1		22	MHz
t_{CP_ADC}	ADC Clock Period		50		1000	ns
f_S	Sampling Frequency		0.05		1	MHz
t_{START}	ADC Startup time	From OFF Mode to Normal Mode: - Voltage Reference OFF - Analog Circuitry OFF	20	30	40	µs
		From Standby Mode to Normal Mode: - Voltage Reference ON - Analog Circuitry OFF	4	8	12	
t_{TRACK}	Track and Hold Time	See Section 45.7.2.1 “Track and Hold Time versus Source Output Impedance” for more details	160			ns
t_{CONV}	Conversion Time			20		t_{CP_ADC}
t_s	Settling time to change offset and gain		200			ns

Table 45-31. External Voltage Reference Input

Parameter	Conditions	Min	Typ	Max	Unit
ADVREF Input Voltage Range, 12-bit	2.4V < V_{DDIN} < 3.6V	2.4		V_{DDIN}	V
ADVREF Input Voltage Range, 10-bit	2.0V < V_{DDIN} < 3.6V	2.0		V_{DDIN}	V
ADVREF Current				250	µA
ADVREF Input DC impedance			14		kΩ

DRAFT

45.7.1 Static Performance Characteristics

Minimal code = 0

Maximal code = 4095

ADC resolution = 12 bits (4096)

Temperature range -40 to 100 °C

f_{ADC} = 2 MHz, ADC_ACR.IBCTL = 01

In Table 45-32 and Table 45-33, the LSB is relative to analog scale for 12-bit ADC:

- Single-ended (ex: ADVREF = 3.0V),
  - Gain = 1, LSB = (3.0V / 4096) = 732 µV
  - Gain = 2, LSB = (1.5V / 4096) = 366 µV
  - Gain = 4, LSB = (750mV / 4096) = 183 µV
- Differential (ex: ADVREF = 3.0V),
  - Gain = 0.5, LSB = (6.0V / 4096) = 1465 µV
  - Gain = 1, LSB = (3.0V / 4096) = 732 µV
  - Gain = 2, LSB = (750mV / 4096) = 366 µV

In Table 45-34, the LSB is relative to analog scale for 10-bit ADC: LSB = 3.0V/1024.

Table 45-32. INL, DNL, 12-bit mode

Symbol	Parameter	Conditions[1]	Min	Typ	Max	Unit
INL	Integral Non-linearity	VDDIN 2.4V to < 3.0V Differential, DIFF = 1, OFF = x, GAIN = xx	-2.2	±1	+2.2	LSB
		VDDIN 2.4V to < 3.0V Single-ended, DIFF = 0, OFF = x, GAIN = xx	-1.8	±1	+1.8	LSB
		VDDIN 3.0V to < 3.6V Differential, DIFF = 1, OFF = x, GAIN = xx	-1.4	±1	+1.4	LSB
		VDDIN 3.3V to < 3.6V Single-ended, DIFF = 0, OFF = x, GAIN = xx	-1.2	±1	+1.2	LSB
DNL	Differential Non-linearity	VDDIN 2.4V to < 3.0V Differential, DIFF = 1, OFF = x, GAIN = xx	-0.8	±0.5	+0.8	LSB
		VDDIN 2.4V to < 3.0V Single-ended, DIFF = 0, OFF = x, GAIN = xx	-0.8	±0.5	+0.8	LSB
		VDDIN 3.0V to < 3.6V Differential, DIFF = 1, OFF = x, GAIN = xx	-0.7	±0.5	+0.7	LSB
		VDDIN 3.0V to < 3.6V Single-ended, DIFF = 0, OFF = x, GAIN = xx	-0.7	±0.5	+0.7	LSB

Note: 1. 'x' in conditions stands for digital 0 or 1

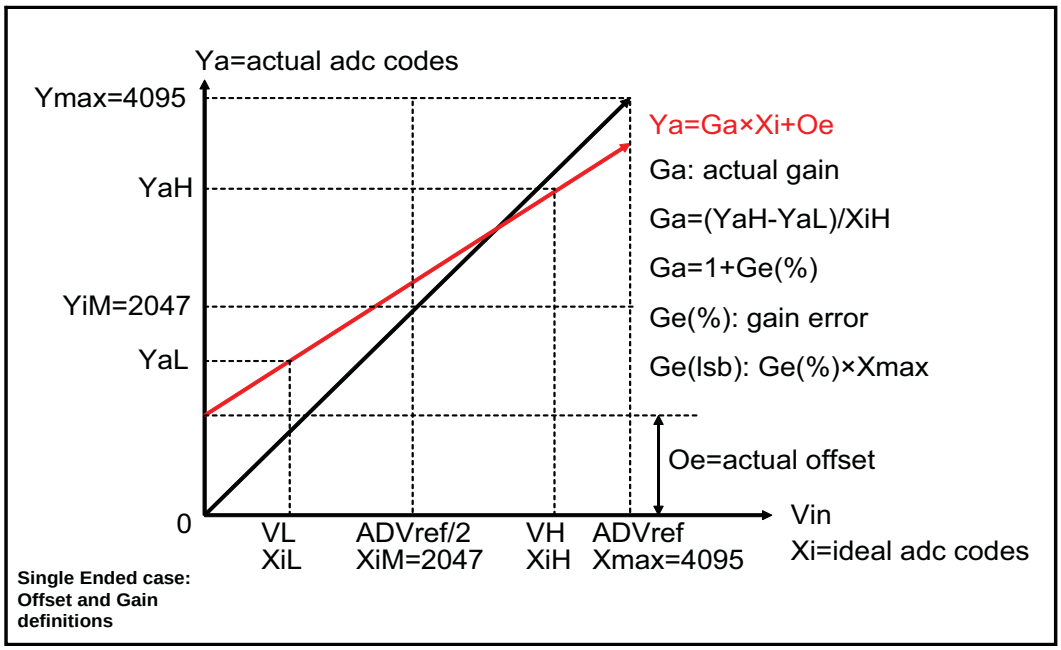
DRAFT

Table 45-33. Gain Error, Offset Error, 12-bit mode

Symbol	Parameter	Conditions[1]	Min	Typ	Max	Unit
E_G	Gain Error	VDDIN 2.4V to < 3.0V Differential, DIFF = 1, OFF = x, GAIN = xx	-1.56 -64	-0.56 -23	+0.29 +12	% LSB
		VDDIN 2.4V to < 3.0V Single-ended, DIFF = 0, OFF = x, GAIN = xx	-1.56 -64	-0.56 -23	+0.29 +12	% LSB
		VDDIN 3.0V to < 3.6V Differential, DIFF = 1, OFF = x, GAIN = xx	-1.56 -64	-0.56 -23	+0.29 +12	% LSB
		VDDIN 3.0V to < 3.6V Single-ended, DIFF = 0, OFF = x, GAIN = xx	-1.56 -64	-0.56 -23	+0.29 +12	% LSB
E_O	Offset Error	VDDIN 2.4V to < 3.0V Differential, DIFF = 1, OFF = x, GAIN = xx	0	+11.5	+64	LSB
		VDDIN 2.4V to < 3.0V Single-ended, DIFF = 0, OFF = x, GAIN = xx	-54	+11.5	+80	LSB
		VDDIN 3.0V to < 3.6V Differential, DIFF = 1, OFF = x, GAIN = xx	-22	+11.5	+48	LSB
		VDDIN 3.0V to < 3.6V Single-ended, DIFF = 0, OFF = x, GAIN = xx	-30	+11.5	+56	LSB

Note: 1. ‘x’ in conditions stands for digital 0 or 1

Figure 45-17. Offset and Gain Definitions



DRAFT

Table 45-34. Static Performance Characteristics - 10-bit Mode (1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
	Resolution			10		Bit
INL	Integral Non-linearity		-1	±0.5	+1	LSB
DNL	Differential Non-linearity	No missing code	-1	±0.5	+1	LSB
$E_{O(uncalib)}$	Offset Error	All gain, Differential or Single-ended, no calibration	-8	+3	+20	LSB
$E_{G(uncalib)}$	Gain Error without calibration	All gain, Differential or Single-ended, no calibration	-16	-6	+3	LSB

Note: 1. Single-ended or differential mode, any gain values.

DRAFT

45.7.2 Dynamic Performance Characteristics

The following conditions apply for the characteristics provided in Table 45-35 and Table 45-36:

- ADC resolution = 12 bits
- Temperature range -40 to 100 °C
- f_{ADC} = 20 MHz
- ADC_ACR.IBCTL = 01
- f_s = 1 MHz
- f_{IN} = 127 kHz
- FFT using 1024 points or more
- Frequency band = [1–500 kHz]
- Nyquist conditions fulfilled

Table 45-35. Dynamic Performance Characteristics in Single-ended and 12-bit Mode

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
SNR	Signal to Noise Ratio	Single-ended, DIFF = 0, OFF = x, GAIN = 10	66	71	74	dB
		Single-ended, Other Cases	57	62	68	dB
THD	Total Harmonic Distortion	Single-ended, DIFF = 0, OFF = x, GAIN = 10	-71	-84	–	dB
		Single-ended, Other Cases	-67	-80	–	dB
SINAD	Signal to Noise and Distortion	Single-ended, DIFF = 0, OFF = x, GAIN = 10	64	69	74	dB
		Single-ended, Other Cases	57	62	68	dB
ENOB	Effective Number of Bits	Single-ended, DIFF = 0, OFF = x, GAIN = 10	10.4	11.2	12	bits
		Single-ended, Other Cases	9.2	10.0	11	bits

Table 45-36. Dynamic Performance Characteristics in Differential and 12-bit Mode

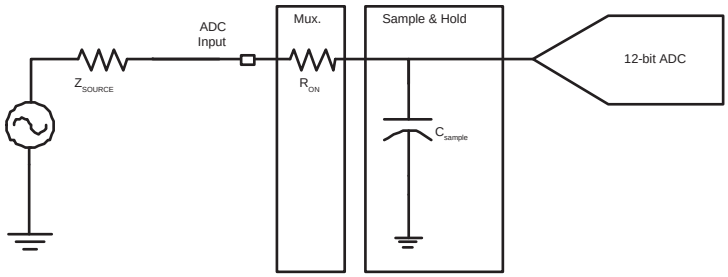
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
SNR	Signal to Noise Ratio	Differential, DIFF = 1, OFF = x, GAIN = 10	61	66	73	dB
		Differential, Other Cases	59	64	67	dB
THD	Total Harmonic Distortion	Differential, DIFF = 1, OFF = x, GAIN = 10	-67	-83	–	dB
		Differential, Other Cases	-69	-85	–	dB
SINAD	Signal to Noise and Distortion	Differential, DIFF = 1, OFF = x, GAIN = 10	60	65	73	dB
		Differential, Other Cases	59	64	67	dB
ENOB	Effective Number of Bits	Differential, DIFF = 0, OFF = x, GAIN = 10	9.8	10.5	11.5	bits
		Differential, Other Cases	9.5	10.3	11.3	bits

DRAFT

45.7.2.1 Track and Hold Time versus Source Output Impedance

The following figure gives a simplified acquisition path.

Figure 45-18. Simplified Acquisition Path



During the tracking phase the ADC needs to track the input signal during the tracking time shown below:

- 10-bit mode: $t_{TRACK} = 0.042 \times Z_{SOURCE} + 160$
- 12-bit mode: $t_{TRACK} = 0.054 \times Z_{SOURCE} + 205$

With t_{TRACK} expressed in ns and Z_{SOURCE} expressed in ohms.

Two cases must be considered:

1. The calculated tracking time (t_{TRACK}) is lower than 15 t_{CP_ADC}.
   Set TRANSFER = 1 and TRACKTIM = 0 in ADC_MR.
   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 $t_{CP_ADC} = 1/f_{ADC}$. See Table 45-37.
2. The calculated tracking time (t_{TRACK}) is higher than 15 t_{CP_ADC}.
   Set TRANSFER = 1 and TRACKTIM = 0 in ADC_MR.
   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 nanoseconds, computed by the previous formula but with minus $15 \times t_{CP_ADC}$ and plus TRANSFER time.

- 10 bit mode: $1/f_S = t_{TRACK} - 15 \times t_{CP_ADC} + 5\ t_{CP_ADC}$
- 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 45-37. Source Impedance Values

f_{ADC} = ADC clock (MHz)	Z_{SOURCE} (kΩ) for 12 bits	Z_{SOURCE} (kΩ) for 10 bits
20.00	10	14
16.00	14	19
10.67	22	30
8.00	31	41
6.40	40	52
5.33	48	63
4.57	57	74
4.00	66	85

DRAFT

Table 45-37. Source Impedance Values (Continued)

f_{ADC} = ADC clock (MHz)	Z_{SOURCE} (kΩ) for 12 bits	Z_{SOURCE} (kΩ) for 10 bits
3.56	74	97
3.20	83	108
2.91	92	119
2.67	100	130
2.46	109	141
2.29	118	152
2.13	126	164
2.00	135	175
1.00	274	353

Table 45-38. Analog Inputs

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IR}	Input Voltage Range[1]		0		V_{ADVREF}	V
I_{lkg}	Input Leakage Current				±0.5	µA
C_i	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 V_{DDIO}.

45.7.3 ADC Application Information

For more information on data converter terminology, please refer to the application note *Data Converter Terminology* (Atmel literature No. 6022) available on www.atmel.com.

DRAFT

45.8 Temperature Sensor

The temperature sensor is connected to channel 15 of the ADC.

The temperature sensor provides an output voltage (V_{O_TS}) that is proportional to absolute temperature (PTAT). V_{O_TS} linearly varies with a temperature slope dV_{O_TS}/dT = 2.65 mV/°C.

V_{O_TS} equals 0.8V at T_A 27°C, with a ±15% accuracy. The V_{O_TS} slope versus temperature dV_{O_TS}/dT = 2.65 mV/°C only shows a ±5% slight variation over process, mismatch and supply voltage.

The user needs to calibrate it (offset calibration) at ambient temperature to eliminate the V_{O_TS} spread at ambient temperature (±15%).

Table 45-39. Temperature Sensor Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{O_TS}	Output Voltage	T_A = 27° C		0.800		V
$V_{O_TS(accuracy)}$	Output Voltage Accuracy	T_A = 27° C	-15		+15	%
dV_{O_TS}/dT	Temperature Sensitivity (Slope Voltage vs Temperature)			2.65		mV/°C
	Slope accuracy		-5		+5	%
	Temperature accuracy	After offset calibration Over temperature range -40 to 85 °C	-5		+5	°C
		After offset calibration Over temperature range 0 to 80 °C	-3		+3	°C
t_{START}	Startup Time	After ADC_ACR.TSON =1		20	40	µs
z_o	Output Impedance				30	Ω
$I_{VDDCORE}$	Current Consumption			50	100	µA

DRAFT

45.9 12-bit DAC Characteristics

Table 45-40. Analog Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDIN}	Analog Supply		2.4		3.6	V
$V_{rip(VDDIN)}$	Max. Voltage Ripple	RMS value, 10 kHz to 20 MHz			20	mV
I_{VDDIN}	Current Consumption	Sleep Mode		0.05	1	µA
		Fast Wake Up		1.8		mA
		Normal Mode with 1 Output ON (DACC_ACR.IBCTLDACCORE = 01, DACC_ACR.IBCTLCHx = 10)	3.4	4.3	7.2	mA
		Normal Mode with 2 Outputs ON (DACC_ACR.IBCTLDACCORE = 01, DACC_ACR.IBCTLCHx = 10)	3.9	5	8.1	mA

Table 45-41. Channel Conversion Time and DAC Clock

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{DAC}	Clock Frequency		1		50	MHz
t_{CP_DAC}	Clock Period		20		1000	ns
f_S	Sampling Frequency		0.04		2	MHz
t_{START}	Startup time	From Sleep Mode to Normal Mode: - Voltage Reference OFF(1) - DAC Core OFF	23	34	45	µs
		From Fast Wake-up to Normal Mode: - Voltage Reference ON - DAC Core OFF	2.5	4	5	
t_{CONV}	Conversion Time			25		t_{CP_DAC}

Notes: 1. External voltage reference for DAC is ADVREF. See the ADC voltage reference characteristics in Table 45-31, “External Voltage Reference Input,” on page 1403.

Table 45-42. Static Performance Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
	Resolution	DAC Clock (f_{DAC}) = 50 MHz, f_s = 2 MHz, f_{IN} = 127 kHz, ADC_ACR.IBCTL = 01, FFT using 1024 points or more, Frequency band = [10–500 kHz], Nyquist conditions fulfilled		12		bit
INL	Integral Non-linearity		-4	±1	+4	LSB
DNL	Differential Non-linearity		-3	±0.5	+3	LSB
E_O	Offset Error		-4	±2	+4	LSB
E_G	Gain Error		-16	±8	+16	LSB

Table 45-43. Dynamic Performance Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
SNR	Signal to Noise Ratio	DAC Clock (f_{DAC}) = 50 MHz, f_s = 2 MHz, f_{IN} = 127 kHz, ADC_ACR.IBCTL = 01, FFT using 1024 points or more, Frequency band = [10–500 kHz], Nyquist conditions fulfilled	64	71	74	dB
THD	Total Harmonic Distortion		-64	-71	-80	dB
SINAD	Signal to Noise and Distortion		62	68	73	dB
ENOB	Effective Number of Bits		62	68	73	dB

DRAFT

Table 45-44. Analog Outputs

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{OR}	Voltage Range		$(1/6) \times V_{ADVREF}$[1]	–	$(5/6) \times V_{ADVREF}$[1]	V

Note: 1. External voltage reference for DAC is ADVREF. See the ADC voltage reference characteristics in Table 45-31, “External Voltage Reference Input,” on page 1403.

DRAFT

45.10 AC Characteristics

45.10.1 Master Clock Characteristics

Table 45-45. Master Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Unit
$1/(t_{CPMCK})$	Master Clock Frequency	VDDCORE @ 1.62V		78	MHz
		VDDCORE @ 1.8V		90	

45.10.2 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 V_{DDIO} - 100 mV
- Addition of rising and falling time inferior to 75% of the period

Table 45-46. I/O Characteristics

Symbol	Parameter	Conditions		Min	Max	Unit
FreqMax1	Pin Group 1 (1) Maximum output frequency	30 pF	V_{DDIO} = 1.62V		45	MHz
			V_{DDIO} = 3.0V		65	
		45 pF	V_{DDIO} = 1.62V		34	
			V_{DDIO} = 3.0V		45	
$PulseminH_1$	Pin Group 1 (1) High Level Pulse Width	30 pF	V_{DDIO} = 1.62V	11		ns
			V_{DDIO} = 3.0V	7.7		
		45 pF	V_{DDIO} = 1.62V	14.7		
			V_{DDIO} = 3.0V	11		
$PulseminL_1$	Pin Group 1 (1) Low Level Pulse Width	30 pF	V_{DDIO} = 1.62V	11		ns
			V_{DDIO} = 3.0V	7.7		
		45 pF	V_{DDIO} = 1.62V	14.7		
			V_{DDIO} = 3.0V	11		
FreqMax2	Pin Group 2 (2) Maximum output frequency	25 pF	$1.62V < V_{DDIO} < 3.6V$		35	MHz
$PulseminH_2$	Pin Group 2 (2) High Level Pulse Width	25 pF	$1.62V < V_{DDIO} < 3.6V$	14.5		ns
$PulseminL_2$	Pin Group 2 (2) Low Level Pulse Width	25 pF	$1.62V < V_{DDIO} < 3.6V$	14.5		ns

Notes: 1. Pin Group 1 = PA3, PA15
2. Pin Group 2 = PA[0–2], PA[4–14], PA[16–31], PB[0–31], PC[0–31]

Table 45-47. NRSTB Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$t_{NRSTB(filtered)}$	Filtered Pulse Width				1	µs
$t_{NRSTB(unfiltered)}$	Unfiltered Pulse Width		100			

DRAFT

45.10.3 SPI Characteristics

Figure 45-19. SPI Master Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA= 1)

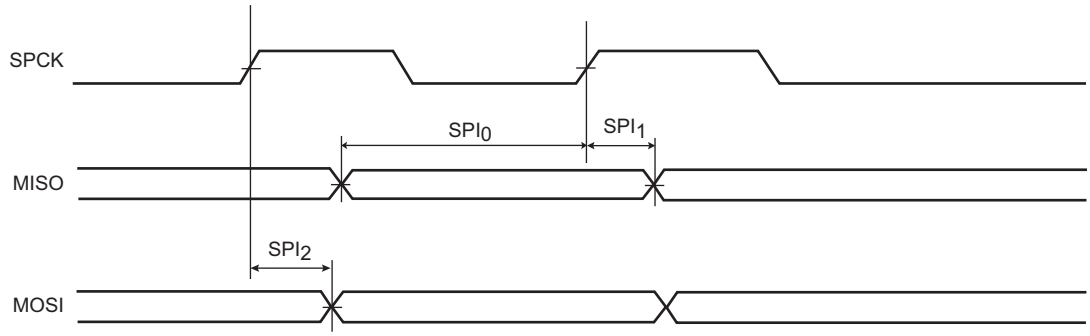


Figure 45-20. SPI Master Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)

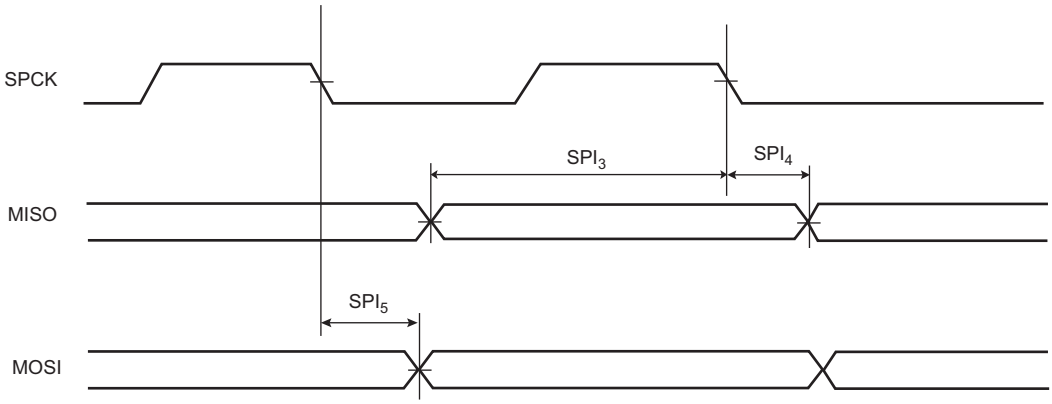
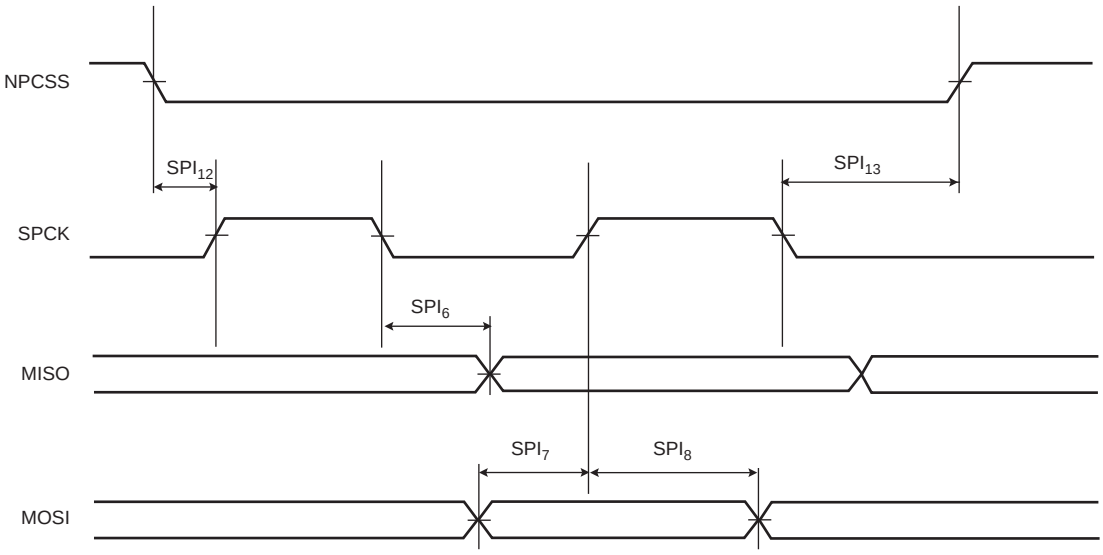
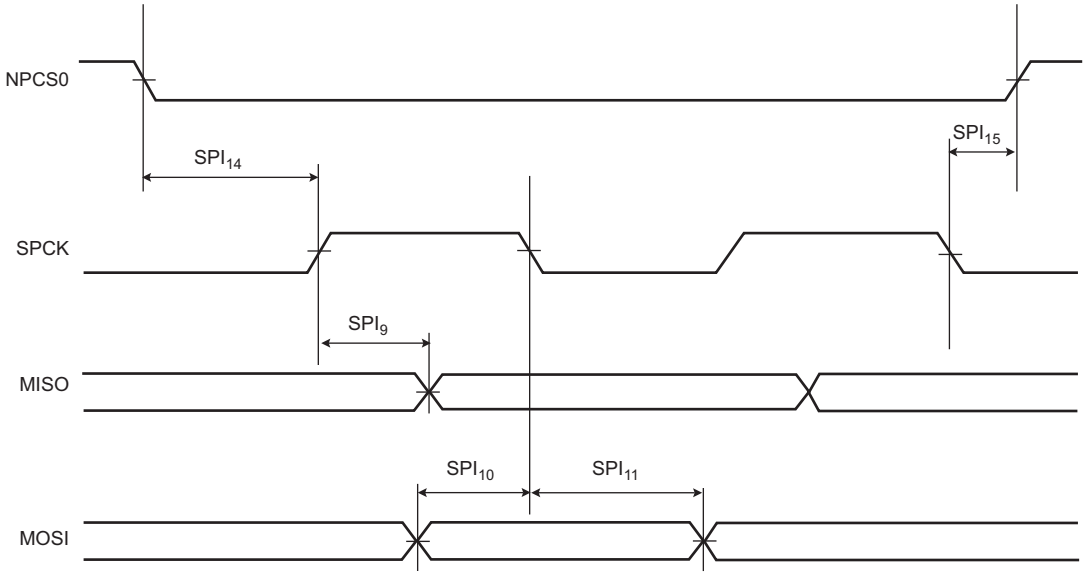


Figure 45-21. SPI Slave Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)



DRAFT

Figure 45-22. SPI Slave Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)



45.10.3.1 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 only sends data to a slave device such as an LCD, for example. The limit is given by SPI_2 (or SPI_5) timing. Since it gives a maximum frequency above the maximum pad speed (see Section 45.10.2 “I/O Characteristics”), the max SPI frequency is defined by the pin FreqMax value.

Master Read Mode

$$f_{SPCK}Max = \frac{1}{SPI_0(orSPI_3) + t_{valid}}$$

t_{valid} is the slave time response to output data after detecting an SPCK edge. For a non-volatile memory with t_{valid} (or t_v) = 12 ns, f_{SPCK}Max = 38.5 MHz at V_{DDIO} = 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_7/SPI_8(or SPI_{10}/SPI_{11}). Since this gives a frequency well above the pad limit, the limit in slave read mode is given by SPCK pad.

Slave Write Mode

$$f_{SPCK}Max = \frac{1}{SPI_6(orSPI_9) + t_{setup}}$$

For 3.3V I/O domain and SPI6, f_{SPCK}Max = 33 MHz. t_{setup} is the setup time from the master before sampling data.

DRAFT

45.10.3.2 SPI Timings

SPI timings are given for the following domains:

- 3.3V domain: V_{DDIO} from 3.0V to 3.6V, maximum external capacitor = 30 pF
- 1.8V domain: V_{DDIO} from 1.65V to 1.95V, maximum external capacitor = 30 pF

Table 45-48. SPI Timings

Symbol	Parameter	Conditions	Min	Max	Unit
SPI_0	MISO Setup time before SPCK rises (master)	3.3V domain	15		ns
		1.8V domain	19.5		ns
SPI_1	MISO Hold time after SPCK rises (master)	3.3V domain	0		ns
		1.8V domain	0		ns
SPI_2	SPCK rising to MOSI Delay (master)	3.3V domain		3	ns
		1.8V domain		3	ns
SPI_3	MISO Setup time before SPCK falls (master)	3.3V domain	16		ns
		1.8V domain	20		ns
SPI_4	MISO Hold time after SPCK falls (master)	3.3V domain	0		ns
		1.8V domain	0		ns
SPI_5	SPCK falling to MOSI Delay (master)	3.3V domain		3.5	ns
		1.8V domain		3.5	ns
SPI_6	SPCK falling to MISO Delay (slave)	3.3V domain		16	ns
		1.8V domain		20	ns
SPI_7	MOSI Setup time before SPCK rises (slave)	3.3V domain	0.5		ns
		1.8V domain	1.5		ns
SPI_8	MOSI Hold time after SPCK rises (slave)	3.3V domain	0		ns
		1.8V domain	0		ns
SPI_9	SPCK rising to MISO Delay (slave)	3.3V domain		16	ns
		1.8V domain		30	ns
SPI_{10}	MOSI Setup time before SPCK falls (slave)	3.3V domain	0		ns
		1.8V domain	0.5		ns
SPI_{11}	MOSI Hold time after SPCK falls (slave)	3.3V domain	1.5		ns
		1.8V domain	1.5		ns
SPI_{12}	NPCS Setup time to SPCK (slave)	3.3V domain	5		ns
		1.8V domain	6.2		ns
SPI_{13}	NPCS Hold time after SPCK (slave)	3.3V domain	0		ns
		1.8V domain	0		ns
SPI_{14}	NPCS Setup time to SPCK (slave)	3.3V domain	5.3		ns
		1.8V domain	5.7		ns
SPI_{15}	NPCS Hold time after SPCK (slave)	3.3V domain	0		ns
		1.8V domain	0		ns

DRAFT

Note that in SPI master mode the SAM3X/A does not sample the data (MISO) on the opposite edge where data clocks out (MOSI) but the same edge is used as shown in Figure 45-19 and Figure 45-20.

45.10.4 MCI 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.

DRAFT

45.10.5 SSC Timings

SSC timings are given for the following domains:

- 1.8V domain: V_{DDIO} from 1.62V to 1.95V, maximum external capacitor = 25 pF
- 3.3V domain: V_{DDIO} from 3.0V to 3.6V, maximum external capacitor = 25 pF

Figure 45-23. SSC Transmitter, TK and TF as output

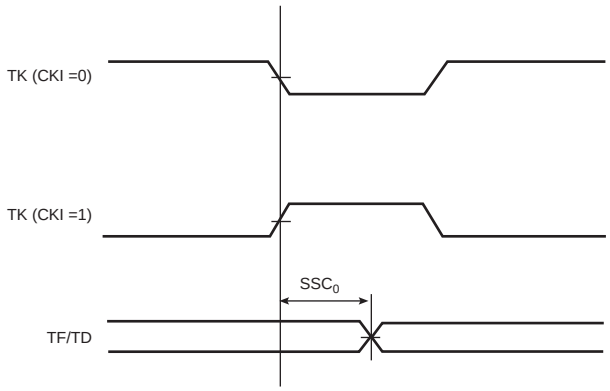


Figure 45-24. SSC Transmitter, TK as input and TF as output

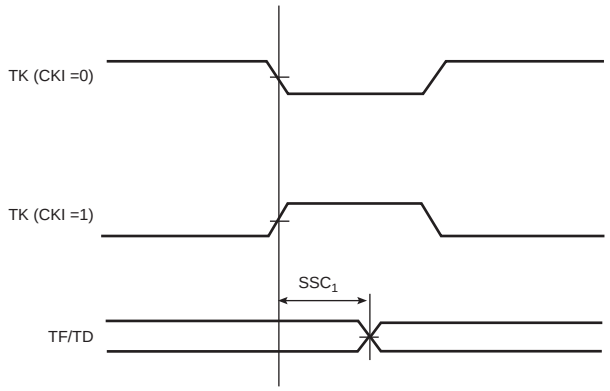
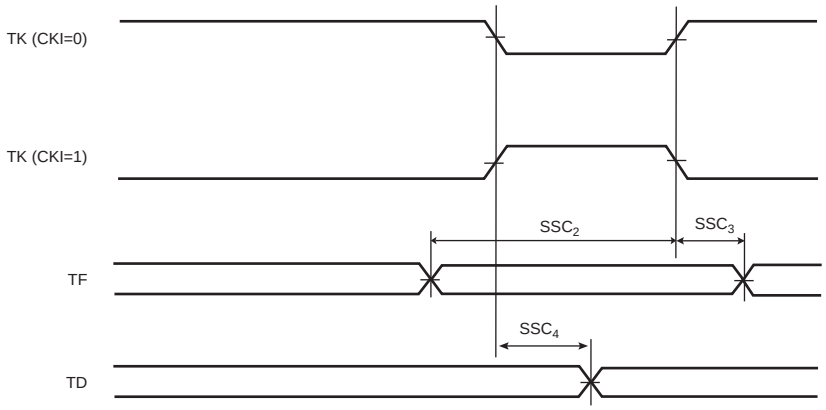


Figure 45-25. SSC Transmitter, TK as output and TF as input



DRAFT

Figure 45-26. SSC Transmitter, TK and TF as input

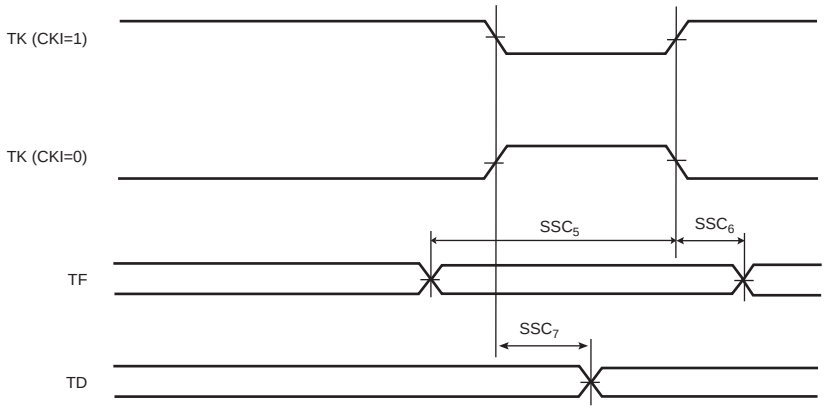


Figure 45-27. SSC Receiver RK and RF as input

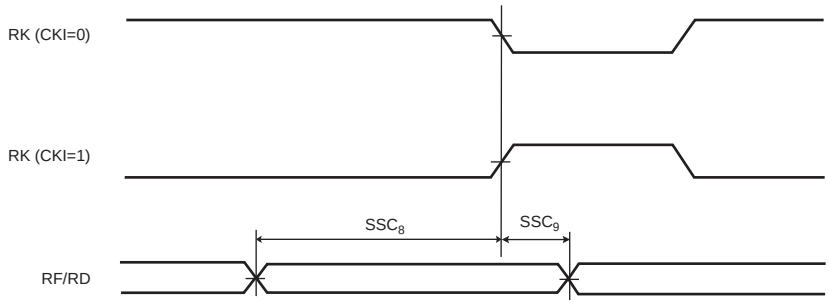
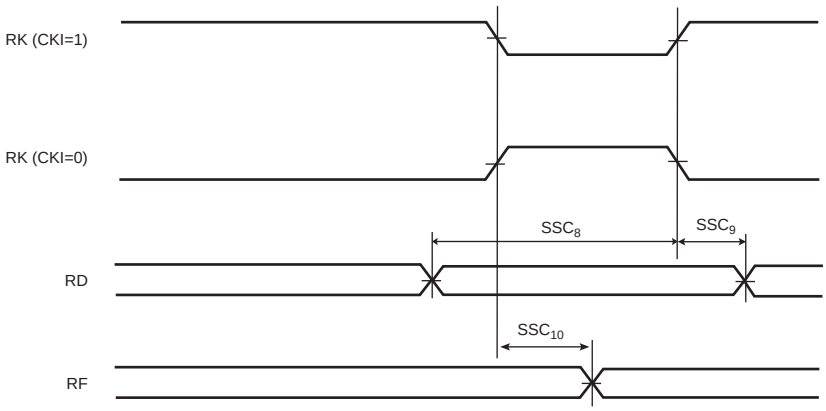


Figure 45-28. SSC Receiver, RK as input and RF as output



DRAFT

Figure 45-29. SSC Receiver, RK and RF as output

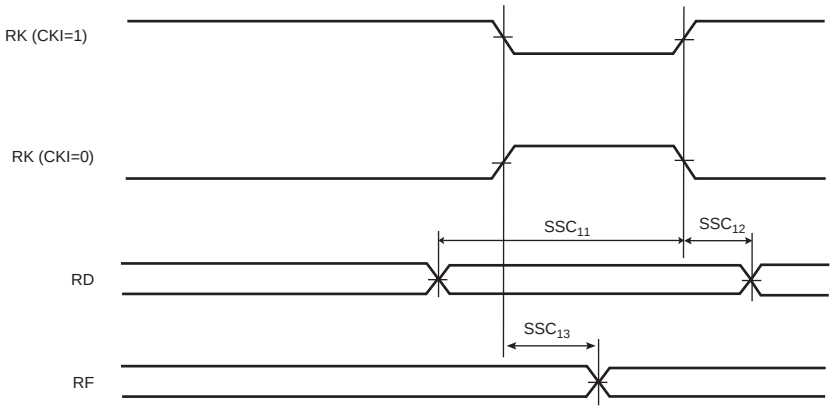


Figure 45-30. SSC Receiver, RK as output and RF as input

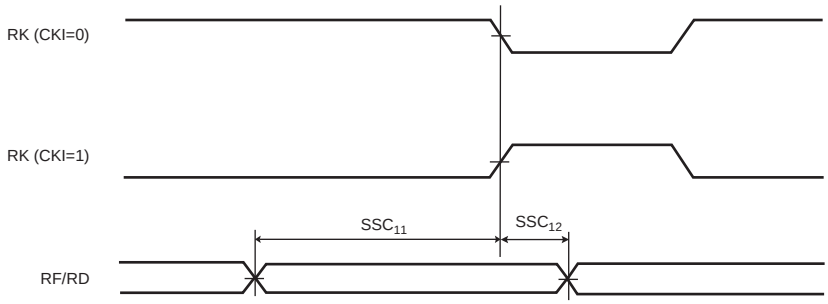


Table 45-49. SSC Timings

Symbol	Parameter	Conditions	Min	Max	Unit
		Transmitter			
SSC_0	TK edge to TF/TD (TK output, TF output)	1.8V domain 3.3V domain	0[1] 0[1]	2.5[1] 7.3[1]	ns
SSC_1	TK edge to TF/TD (TK input, TF output)	1.8V domain 3.3V domain	6.4[1] 6.4[1]	18.1[1] 19.5[1]	ns
SSC_2	TF setup time before TK edge (TK output)	1.8V domain 3.3V domain	29 - t_{CPMCK} 50 - t_{CPMCK}		ns
SSC_3	TF hold time after TK edge (TK output)	1.8V domain 3.3V domain	t_{CPMCK} - 15 t_{CPMCK} - 36		ns
SSC_4	TK edge to TF/TD (TK output, TF input)	1.8V domain 3.3V domain	0 (+ 2 × t_{CPMCK})[1] 0 (+ 2 × t_{CPMCK})[1]	2.5 (+ 2 × t_{CPMCK})[1] 7.3 (+ 2 × t_{CPMCK})[1]	ns
SSC_5	TF setup time before TK edge (TK input)	1.8V domain 3.3V domain	0 0		ns
SSC_6	TF hold time after TK edge (TK input)	1.8V domain 3.3V domain	t_{CPMCK} t_{CPMCK}		ns
SSC_7	TK edge to TF/TD (TK input, TF input)	1.8V domain 3.3V domain	6.4 (+ 3 × t_{CPMCK})[1] 6.4 (+ 3 × t_{CPMCK})[1]	18 (+ 3 × t_{CPMCK})[1] 20 (+ 3 × t_{CPMCK})[1]	ns

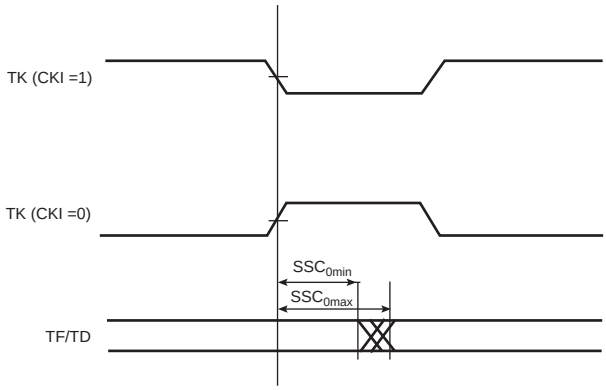
DRAFT

Table 45-49. SSC Timings (Continued)

Symbol	Parameter	Conditions	Min	Max	Unit
		Receiver			
SSC_8	RF/RD setup time before RK edge (RK input)	1.8V domain 3.3V domain	0 0		ns
SSC_9	RF/RD hold time after RK edge (RK input)	1.8V domain 3.3V domain	t_{CPMCK} t_{CPMCK}		ns
SSC_{10}	RK edge to RF (RK input)	1.8V domain 3.3V domain	5(1) 5(1)	16(1) 16(1)	ns
SSC_{11}	RF/RD setup time before RK edge (RK output)	1.8V domain 3.3V domain	16 - t_{CPMCK} 16 - t_{CPMCK}		ns
SSC_{12}	RF/RD hold time after RK edge (RK output)	1.8V domain 3.3V domain	t_{CPMCK} - 5 t_{CPMCK} - 5		ns
SSC_{13}	RK edge to RF (RK output)	1.8V domain 3.3V domain	0(1) 0(1)	1(1) 1(1)	ns

Note: 1. For output signals (TF, TD, RF), Min and Max access times are defined. The Min access time is the time between the TK (or RK) edge and the signal change. The Max access timing is the time between the TK edge and the signal stabilization. Figure 45-31 illustrates Min and Max accesses for SSC_0. The same applies for SSC_1, SSC_4, and SSC_7, SSC_{10} and SSC_{13}.

Figure 45-31. Min and Max Access Time of Output Signals



DRAFT

45.10.6 SMC Timings

SMC timings are given for the following domains:

- 1.8V domain: V_{DDIO} from 1.65V to 1.95V, maximum external capacitor = 30 pF
- 3.3V domain: V_{DDIO} from 3.0V to 3.6V, maximum external capacitor = 50 pF

Timings are given assuming a capacitance load on data, control and address pads.

In the following tables t_{CPMCK} is MCK period.

45.10.6.1 Read Timings

Table 45-50. SMC Read Signals - NRD Controlled (READ_MODE = 1)

Symbol	Parameter	Min		Max		Unit
	VDDIO supply	1.8V Domain	3.3V Domain	1.8V Domain	3.3V Domain	
NO HOLD SETTINGS (NRD_HOLD = 0)						
SMC_1	Data Setup before NRD High	22.5	18			ns
SMC_2	Data Hold after NRD High	0	0			ns
HOLD SETTINGS (NRD_HOLD ≠ 0)						
SMC_3	Data Setup before NRD High	20.3	16			ns
SMC_4	Data Hold after NRD High	0	0			ns
HOLD or NO HOLD SETTINGS (NRD_HOLD ≠ 0, NRD_HOLD = 0)						
SMC_5	NBS0/A0, NBS1, NBS2/A1, NBS3, A2–A25 Valid before NRD High	(NRD_SETUP + NRD_PULSE) × t_{CPMCK} - 20	(NRD_SETUP + NRD_PULSE) × t_{CPMCK} - 20			ns
SMC_6	NCS low before NRD High	(NRD_SETUP + NRD_PULSE - NCS_RD_SETUP) × t_{CPMCK} - 20	(NRD_SETUP + NRD_PULSE - NCS_RD_SETUP) × t_{CPMCK} - 20			ns
SMC_7	NRD Pulse Width	nrd pulse × t_{CPMCK} - 3	nrd pulse × t_{CPMCK} - 3			ns

Table 45-51. SMC Read Signals - NCS Controlled (READ_MODE = 0)

Symbol	Parameter	Min		Max		Unit
	VDDIO supply	1.8V Domain	3.3V Domain	1.8V Domain	3.3V Domain	
NO HOLD SETTINGS (NCS_RD_HOLD = 0)						
SMC_8	Data Setup before NCS High	26.3	24.6			ns
SMC_9	Data Hold after NCS High	0	0			ns
HOLD SETTINGS (NCS_RD_HOLD ≠ 0)						
SMC_{10}	Data Setup before NCS High	24.1	22.4			ns
SMC_{11}	Data Hold after NCS High	0	0			ns
HOLD or NO HOLD SETTINGS (NCS_RD_HOLD ≠ 0, NCS_RD_HOLD = 0)						
SMC_{12}	NBS0/A0, NBS1, NBS2/A1, NBS3, A2–A25 valid before NCS High	(NCS_RD_SETUP + NCS_RD_PULSE) × t_{CPMCK} - 3	(NCS_RD_SETUP + NCS_RD_PULSE) × t_{CPMCK} - 3			ns
SMC_{13}	NRD low before NCS High	(NCS_RD_SETUP + NCS_RD_PULSE - NRD_SETUP) × t_{CPMCK} + 0.7	(NCS_RD_SETUP + NCS_RD_PULSE - NRD_SETUP) × t_{CPMCK} + 0.6			ns
SMC_{14}	NCS Pulse Width	NCS_RD_PULSE length × t_{CPMCK} - 6	NCS_RD_PULSE length × t_{CPMCK} - 6			ns

DRAFT

45.10.6.2 Write Timings

Table 45-52. SMC Write Signals - NWE Controlled (WRITE_MODE = 1)

Symbol	Parameter	Min		Max		Unit
	VDDIO supply	1.8V Domain	3.3V Domain	1.8V Domain	3.3V Domain	
HOLD or NO HOLD SETTINGS (NWE_HOLD ≠ 0, NWE_HOLD = 0)						
SMC_{15}	Data Out Valid before NWE High	NWE_PULSE × t_{CPMCK} - 7.5	NWE_PULSE × t_{CPMCK} - 7			ns
SMC_{16}	NWE Pulse Width	NWE_PULSE × t_{CPMCK} - 3	NWE_PULSE × t_{CPMCK} - 3			ns
SMC_{17}	NBS0/A0 NBS1, NBS2/A1, NBS3, A2–A25 valid before NWE low	NWE_SETUP × t_{CPMCK} + 6	NWE_SETUP × t_{CPMCK} + 6			ns
SMC_{18}	NCS low before NWE high	(NWE_SETUP - NCS_RD_SETUP + NWE_PULSE) × t_{CPMCK} + 10	(NWE_SETUP - NCS_RD_SETUP + NWE_PULSE) × t_{CPMCK} + 12			ns
HOLD SETTINGS (NWE_HOLD ≠ 0)						
SMC_{19}	NWE High to Data OUT, NBS0/A0 NBS1, NBS2/A1, NBS3, A2–A25 change	NWE_HOLD × t_{CPMCK} - 2.4	NWE_HOLD × t_{CPMCK} - 1.8			ns
SMC_{20}	NWE High to NCS Inactive(1)	(NWE_HOLD - NCS_WR_HOLD) × t_{CPMCK} - 0.3	(NWE_HOLD - NCS_WR_HOLD) × t_{CPMCK} - 0.3			ns
NO HOLD SETTINGS (NWE_HOLD = 0)						
SMC_{21}	NWE High to Data OUT, NBS0/A0 NBS1, NBS2/A1, NBS3, A2–A25, NCS change(1)	4	4			ns

Notes: 1. Hold length = total cycle duration - setup duration - pulse duration. “Hold length” is for “NCS_WR_HOLD length” or “NWE_HOLD length”.

Table 45-53. SMC Write NCS Controlled (WRITE_MODE = 0)

Symbol	Parameter	Min		Max		Unit
	VDDIO supply	1.8V Domain	3.3V Domain	1.8V Domain	3.3V Domain	
SMC_{22}	Data Out Valid before NCS High	NCS_WR_PULSE × t_{CPMCK} - 1	NCS_WR_PULSE × t_{CPMCK} - 1			ns
SMC_{23}	NCS Pulse Width	NCS_WR_PULSE × t_{CPMCK} - 6	NCS_WR_PULSE × t_{CPMCK} - 6			ns
SMC_{24}	NBS0/A0 NBS1, NBS2/A1, NBS3, A2–A25 valid before NCS low	NCS_WR_SETUP × t_{CPMCK} - 5	NCS_WR_SETUP × t_{CPMCK} - 5			ns
SMC_{25}	NWE low before NCS high	(NCS_WR_SETUP - NWE_SETUP + NCS pulse) × t_{CPMCK} + 1.3	(NCS_WR_SETUP - NWE_SETUP + NCS pulse) × t_{CPMCK} + 1.3			ns
SMC_{26}	NCS High to Data Out, NBS0/A0, NBS1, NBS2/A1, NBS3, A2–A25, change	NCS_WR_HOLD × t_{CPMCK} - 8.2	NCS_WR_HOLD × t_{CPMCK} - 9.6			ns
SMC_{27}	NCS High to NWE Inactive	(NCS_WR_HOLD - NWE_HOLD) × t_{CPMCK} - 6.2	(NCS_WR_HOLD - NWE_HOLD) × t_{CPMCK} - 9			ns

DRAFT

Figure 45-32. SMC Timings - NCS Controlled Read and Write

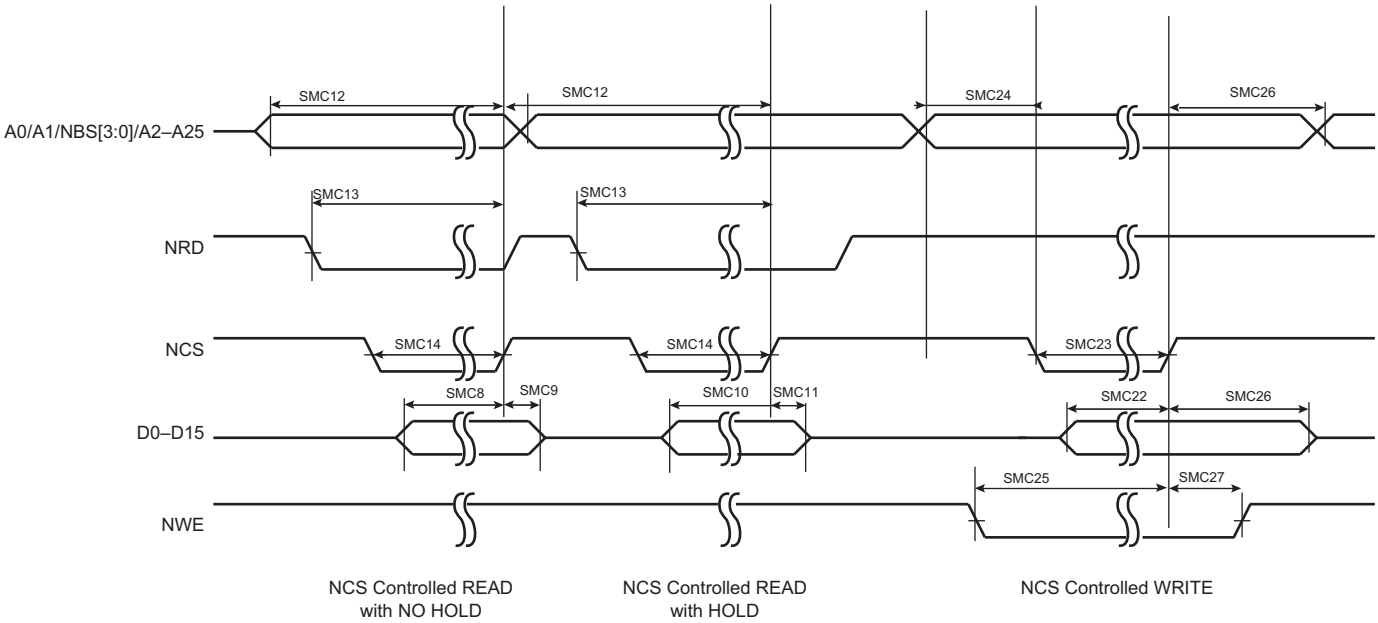
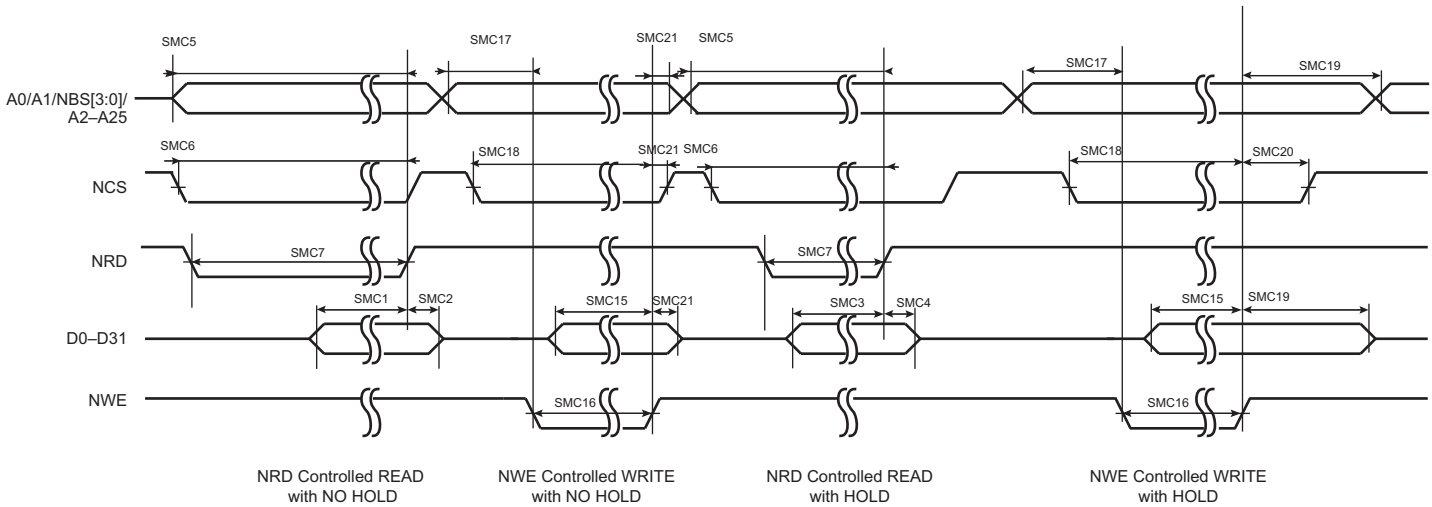


Figure 45-33. SMC Timings - NRD Controlled Read and NWE Controlled Write



DRAFT

Table 45-54. SDRAM PC100 Characteristics

Parameter	Min	Max		Unit
	3V Supply[1]	3V Supply[1]	3.3V Supply[2]	
SDRAM Controller Clock Frequency		76.9	92	MHz
Control/Address/Data In Setup[3][4]	2			ns
Control/Address/Data In Hold[3][4]	1			ns
Data Out Access time after SDCK rising		6	6	ns
Data Out change time after SDCK rising	3			ns

Notes:
1. 3V domain: V_{DDIO} from 3.0V to 3.6V, maximum external capacitor on data, control and address = 50 pF, maximum external capacitor on clock = 10 pF
2. 3.3V domain: V_{DDIO} from 3.3V to 3.6V, maximum external capacitor on data, control and address = 50 pF, maximum external capacitor on clock = 10 pF
3. Control is the tNSSet of following signals : SDCKE, SDCS, RAS, CAS, SDA10, BAx, DQMx, and SDWE
4. Address is the set of A0–A9, A11–A13

Table 45-55. SDRAM PC133 Characteristics

Parameter	Min	Max		Unit
	3V Supply[1]	3V Supply[1]	3.3V Supply[2]	
SDRAM Controller Clock Frequency		80.6	100	MHz
Control/Address/Data In Setup[3][4]	1.5			ns
Control/Address/Data In Hold[3][4]	0.8			ns
Data Out Access time after SDCK rising		5.4	5.4	ns
Data Out change time after SDCK rising	3.0			ns

Notes:
1. 3V domain: V_{DDIO} from 3.0V to 3.6V, maximum external capacitor on data, control and address = 50 pF, maximum external capacitor on clock = 10 pF
2. 3.3V domain: V_{DDIO} from 3.3V to 3.6V, maximum external capacitor on data, control and address = 50 pF, maximum external capacitor on clock = 10 pF
3. Control is the tNSSet of following signals: SDCKE, SDCS, RAS, CAS, SDA10, BAx, DQMx, and SDWE
4. Address is the set of A0–A9, A11–A13

Table 45-56. Mobile Characteristics

Parameter	Min	Max		Unit
	1.62V Supply[1]	1.62V Supply[1]	1.8V Supply[1]	
SDRAM Controller Clock Frequency		71.9	84	MHz
Control/Address/Data In Setup[2][3]	1.5			ns
Control/Address/Data In Hold[2][3]	1			ns
Data Out Access time after SDCK rising		6	6	ns
Data Out change time after SDCK rising	2.5			ns

Notes:
1. 1.8V domain: V_{DDIO} from 1.62V to 1.95V, maximum external capacitor on data, control and address = 30 pF, maximum external capacitor on clock = 10 pF
2. Control is the tNSSet of following signals : SDCKE, SDCS, RAS, CAS, SDA10, BAx, DQMx, and SDWE
3. Address is the set of A0–A9, A11–A13

DRAFT

45.10.7 USART in SPI Mode Timings

USART in SPI mode timings are given for the following domains:

- 1.8V domain: V_{DDIO} from 1.62V to 1.95V, maximum external capacitor = 30 pF
- 3.3V domain: V_{DDIO} from 3.0V to 3.6V, maximum external capacitor = 30 pF

Figure 45-34. USART SPI Master Mode

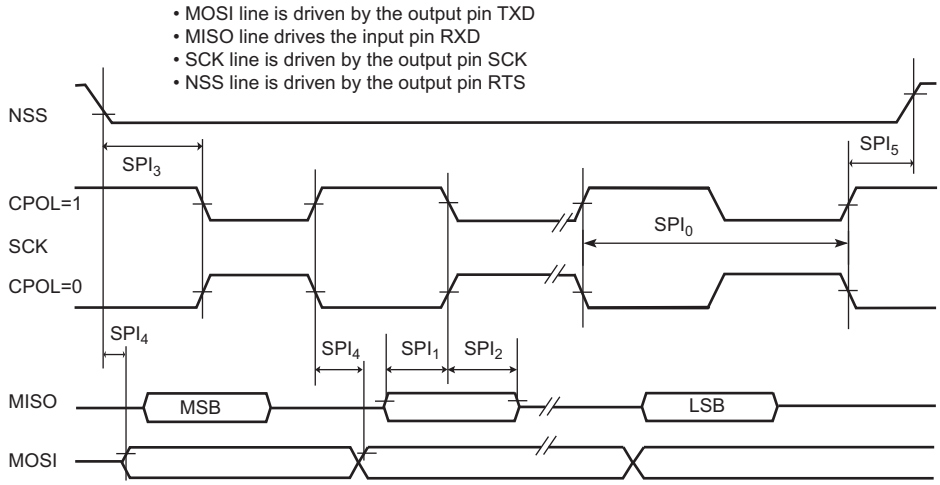
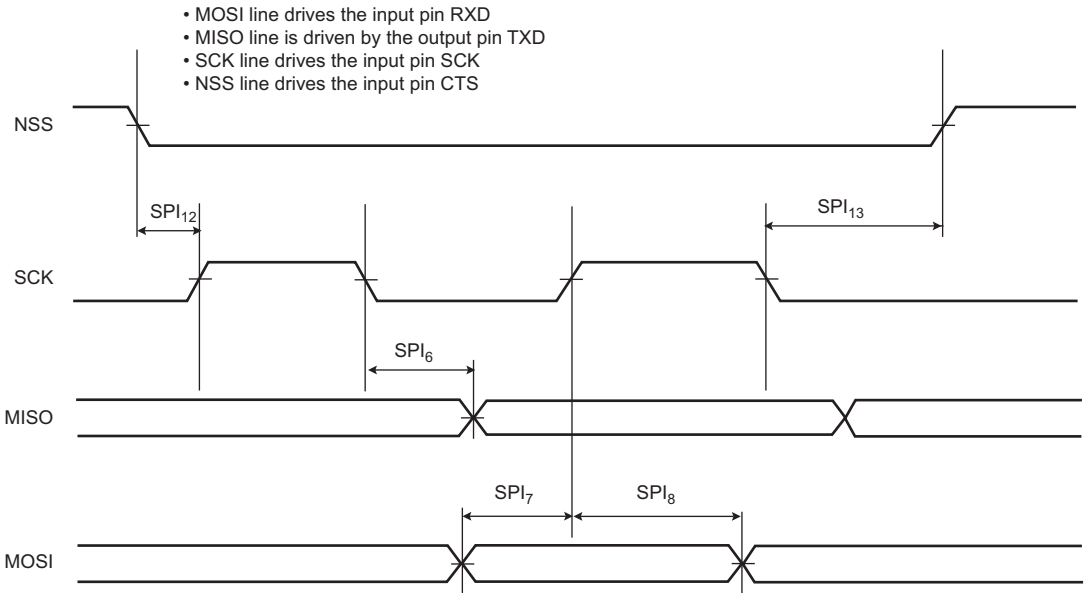
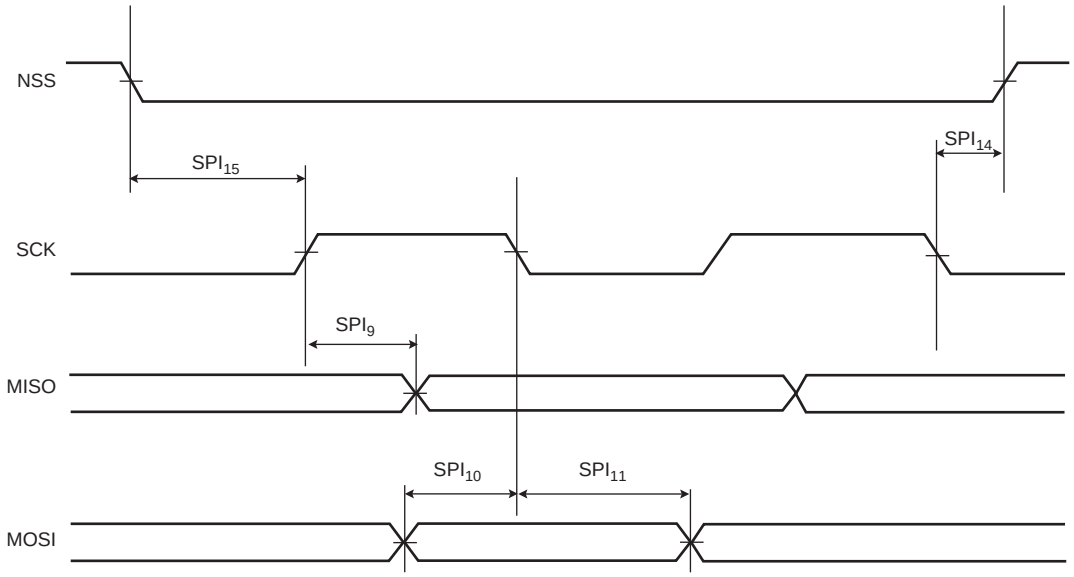


Figure 45-35. USART SPI Slave mode (Mode 1 or 2)



DRAFT

Figure 45-36. USART SPI Slave mode (Mode 0 or 3)



DRAFT

Table 45-57. USART SPI Timings

Symbol	Parameter	Conditions	Min	Max	Unit
Master Mode					
SPI_0	t_{CPSCK} Period	1.8V domain 3.3V domain	t_{CPMCK} / 6		ns
SPI_1	Input Data Setup Time	1.8V domain 3.3V domain	$0.5 \times t_{CPMCK}$ + 7.7		ns
SPI_2	Input Data Hold Time	1.8V domain 3.3V domain	$1.5 \times t_{CPMCK}$ + 1.8 $1.5 \times t_{CPMCK}$ + 0.7		ns
SPI_3	Chip Select Active to Serial Clock	1.8V domain 3.3V domain	$1.5 \times t_{CPSCK}$ -1		ns
SPI_4	Output Data Setup Time	1.8V domain 3.3V domain	0	7.4	ns
SPI_5	Serial Clock to Chip Select Inactive	1.8V domain 3.3V domain		$1 \times t_{CPSCK}$ - 1	ns
Slave Mode					
SPI_6	t_{CPSCK} falling to MISO	1.8V domain 3.3V domain	7	30	ns
SPI_7	MOSI Setup time before t_{CPSCK} rises	1.8V domain 3.3V domain	$2 \times t_{CPMCK}$ + 6.8		ns
SPI_8	MOSI Hold time after t_{CPSCK} rises	1.8V domain 3.3V domain	1		ns
SPI_9	t_{CPSCK} rising to MISO	1.8V domain 3.3V domain	7		ns
SPI_{10}	MOSI Setup time before t_{CPSCK} falls	1.8V domain 3.3V domain	$2 \times t_{CPMCK}$ + 6		ns
SPI_{11}	MOSI Hold time after t_{CPSCK} falls	1.8V domain 3.3V domain	2.7 1		ns
SPI_{12}	NPCS0 setup to t_{CPSCK} rising	1.8V domain 3.3V domain	$2.5 \times t_{CPMCK}$ +1		ns
SPI_{13}	NPCS0 hold after t_{CPSCK} falling	1.8V domain 3.3V domain	$1.5 \times t_{CPMCK}$ + 4		ns
SPI_{14}	NPCS0 setup to t_{CPSCK} falling	1.8V domain 3.3V domain	$2.5 \times t_{CPMCK}$ + 0.4		ns
SPI_{15}	NPCS0 hold after t_{CPSCK} rising	1.8V domain 3.3V domain	$1.5 \times t_{CPMCK}$ + 2.7		ns

DRAFT

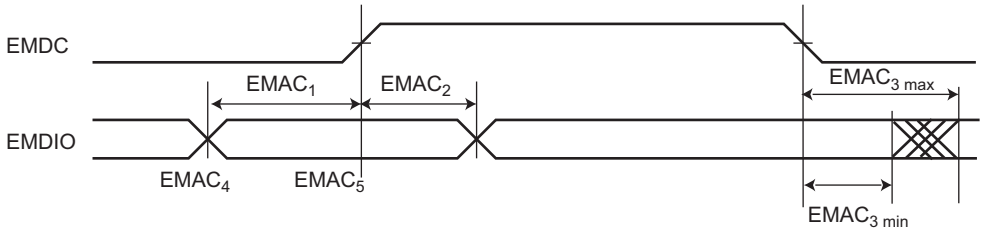
45.10.8 EMAC

Table 45-58. EMAC Signals Relative to EMDC

Symbol	Parameter	Conditions	Min	Max	Unit
$EMAC_1$	Setup for EMDIO from EMDC rising	VDDIO from 3.0V to 3.6V, maximum external capacitor = 20 pF	10		ns
$EMAC_2$	Hold for EMDIO from EMDC rising		10		
$EMAC_3$	EMDIO toggling from EMDC rising		0(1)	300(1)	

Notes: 1. For EMAC output signals, Min and Max access time are defined. The Min access time is the time between the EDMC rising edge and the signal change. The Max access timing is the time between the EDMC rising edge and the signal stabilizes. Figure 45-37 illustrates Min and Max accesses for $EMAC_3$.

Figure 45-37. Min and Max Access Time of EMAC Output Signals



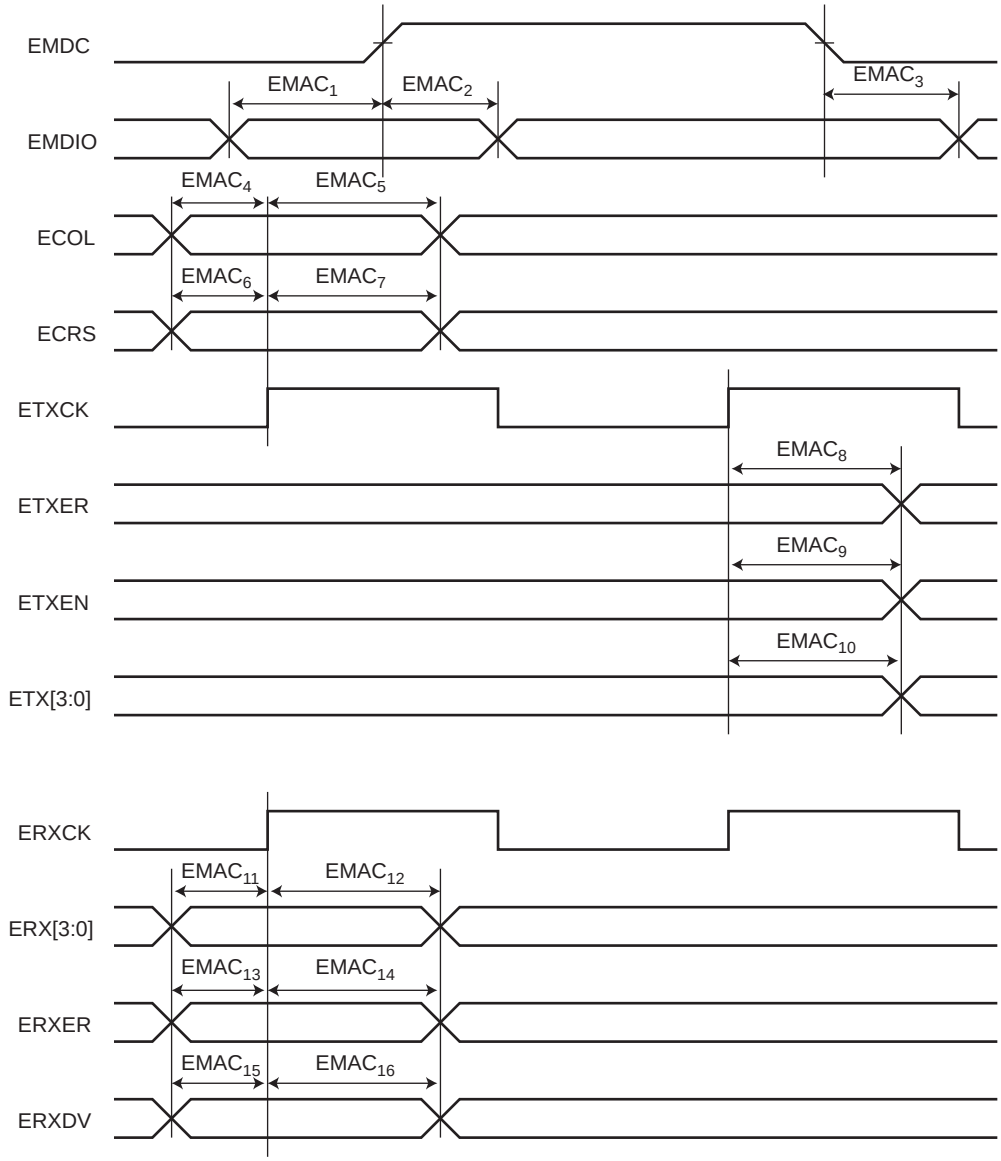
45.10.8.1 MII Mode

Table 45-59. EMAC MII Specific Signals

Symbol	Parameter	Conditions	Min	Max	Unit
$EMAC_4$	Setup for ECOL from ETXCK rising	VDDIO from 3.0V to 3.6V, maximum external capacitor = 20 pF	10		ns
$EMAC_5$	Hold for ECOL from ETXCK rising		10		
$EMAC_6$	Setup for ECRS from ETXCK rising		10		
$EMAC_7$	Hold for ECRS from ETXCK rising		10		
$EMAC_8$	ETXER toggling from ETXCK rising		10	25	
$EMAC_9$	ETXEN toggling from ETXCK rising		10	25	
$EMAC_{10}$	ETX toggling from ETXCK rising		10	25	
$EMAC_{11}$	Setup for ERX from ERXCK		10		
$EMAC_{12}$	Hold for ERX from ERXCK		10		
$EMAC_{13}$	Setup for ERXER from ERXCK		10		
$EMAC_{14}$	Hold for ERXER from ERXCK		10		
$EMAC_{15}$	Setup for ERXDV from ERXCK		10		
$EMAC_{16}$	Hold for ERXDV from ERXCK		10		

DRAFT

Figure 45-38. EMAC MII Mode



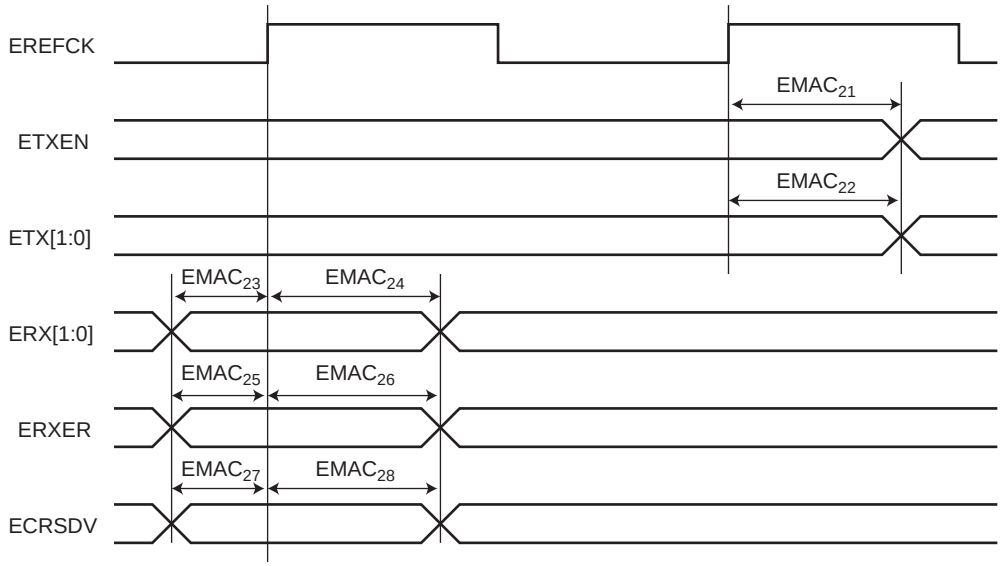
DRAFT

45.10.8.2 RMII Mode

Table 45-60. RMII Mode

Symbol	Parameter	Conditions	Min	Max	Unit
$EMAC_{21}$	ETXEN toggling from EREFCK rising	VDDIO from 3.0V to 3.6V, maximum external capacitor = 20 pF	2	16	ns
$EMAC_{22}$	ETX toggling from EREFCK rising		2	16	
$EMAC_{23}$	Setup for ERX from EREFCK rising		4		
$EMAC_{24}$	Hold for ERX from EREFCK rising		2		
$EMAC_{25}$	Setup for ERXER from EREFCK rising		4		
$EMAC_{26}$	Hold for ERXER from EREFCK rising		2		
$EMAC_{27}$	Setup for ECRSDV from EREFCK rising		4		
$EMAC_{28}$	Hold for ECRSDV from EREFCK rising		2		

Figure 45-39. EMAC RMII Timings



DRAFT

45.10.9 Two-wire Serial Interface Characteristics

Table 45-61 describes the requirements for devices connected to the Two-wire Serial Bus. For timing symbols refer to Figure 45-40.

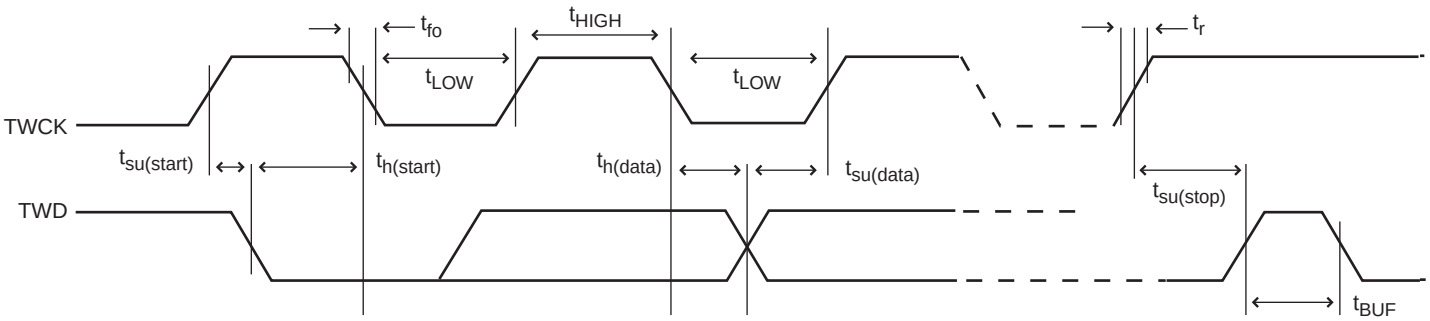
Table 45-61. Two-wire Serial Bus Requirements

Symbol	Parameter	Conditions	Min	Max	Unit
V_{IL}	Input Low-voltage		-0.3	$0.3 \times V_{DDIO}$	V
V_{IH}	Input High-voltage		$0.7 \times V_{DDIO}$	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		$20 + 0.1C_b$ (1)(2)	300	ns
t_{fo}	Output Fall Time from V_{IHmin} to V_{ILmax}	10 pF < C_b < 400 pF Figure 45-40	$20 + 0.1C_b$ (1)(2)	250	ns
C_i (1)	Capacitance for each I/O pin		–	10	pF
f_{TWCK}	TWCK Clock Frequency		0	400	kHz
Rp	Value of Pull-up resistor	$f_{TWCK} \le 100$ kHz	$(V_{DDIO}$ - 0.4V) ÷ 3mA	1000ns ÷ C_b	Ω
		$f_{TWCK} > 100$ kHz		300ns ÷ C_b	
t_{LOW}	Low Period of the TWCK clock	$f_{TWCK} \le 100$ kHz	(3)	–	µs
		$f_{TWCK} > 100$ kHz	(3)	–	µs
t_{HIGH}	High period of the TWCK clock	$f_{TWCK} \le 100$ kHz	(4)	–	µs
		$f_{TWCK} > 100$ kHz	(4)	–	µs
$t_{h(start)}$	Hold Time (repeated) START Condition	$f_{TWCK} \le 100$ kHz	t_{HIGH}	–	µs
		$f_{TWCK} > 100$ kHz	t_{HIGH}	–	µs
$t_{su(start)}$	Setup time for a repeated START condition	$f_{TWCK} \le 100$ kHz	t_{HIGH}	–	µs
		$f_{TWCK} > 100$ kHz	t_{HIGH}	–	µs
$t_{h(data)}$	Data hold time	$f_{TWCK} \le 100$ kHz	0	$3 \times t_{CPMCK}$ (5)	µs
		$f_{TWCK} > 100$ kHz	0	$3 \times t_{CPMCK}$ (5)	µs
$t_{su(data)}$	Data setup time	$f_{TWCK} \le 100$ kHz	$t_{LOW} - 3 \times t_{CPMCK}$ (5)	–	ns
		$f_{TWCK} > 100$ kHz	$t_{LOW} - 3 \times t_{CPMCK}$ (5)	–	ns
$t_{su(stop)}$	Setup time for STOP condition	$f_{TWCK} \le 100$ kHz	t_{HIGH}	–	µs
		$f_{TWCK} > 100$ kHz	t_{HIGH}	–	µs
t_{BUF}	Bus free time between a STOP and START condition	$f_{TWCK} \le 100$ kHz	t_{LOW}	–	µs
		$f_{TWCK} > 100$ kHz	t_{LOW}	–	µs

Note:
1. Required only for $f_{TWCK} > 100$ kHz
2. C_b = capacitance of one bus line in pF. Per I2C Standard compatibility, C_b Max = 400 pF
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_{CPMCK} = MCK bus period

DRAFT

Figure 45-40. Two-wire Serial Bus Timing



DRAFT

45.10.10 Embedded Flash Characteristics

The maximum operating frequency given in Table 45-62 is limited by the Embedded Flash access time when the processor is fetching code out of it. The table provides the device maximum operating frequency defined by the value of field FWS in the EEFC_FMR. This field defines the number of wait states required to access the Embedded Flash Memory.

Note: The embedded flash is fully tested during production test. The flash contents is not set to a known state prior to shipment. Therefore, the flash contents should be erased prior to programming an application.

Table 45-62. Embedded Flash Wait State - VDDCORE 1.62V/1.80V

EEFC_FMR.FWS	Read Operations	Maximum Operating Frequency (MHz)	
		VDDCORE 1.62V	VDDCORE 1.80V
0	1 cycle	17	19
1	2 cycles	45	50
2	3 cycles	58	64
3	4 cycles	70	80
4	5 cycles	78	90

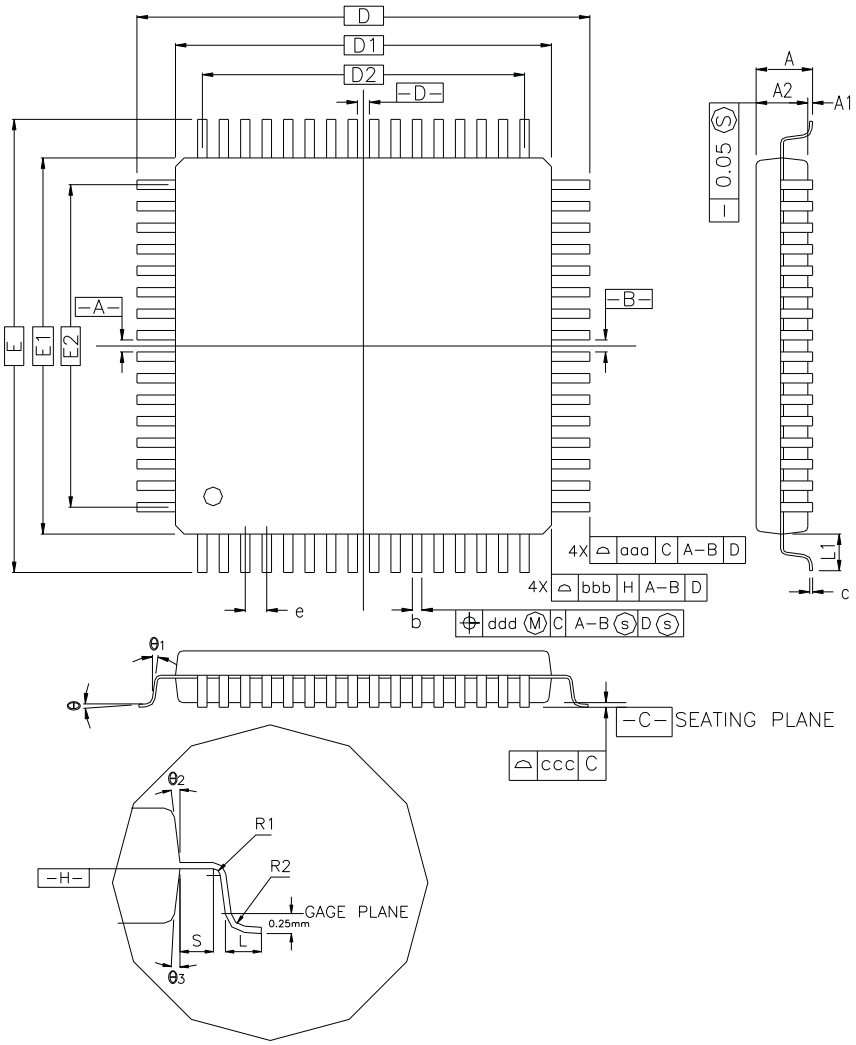
Table 45-63. AC Flash Characteristics

Parameter	Conditions	Min	Typ	Max	Unit
Program Cycle Time	Per page including auto-erase			4.6	ms
	Per page without auto-erase			2.3	ms
Full Chip Erase			10	11.5	ms
Data Retention	Not Powered or Powered	10			years
Endurance	Write/Erase cycles @ 25°C		30K		cycles
	Write/Erase cycles @ 85°C	10K			

DRAFT

46. Mechanical Characteristics

Figure 46-1. 100-lead LQFP Package Drawing



COTROL DIMENSIONS ARE IN MILLIMETERS.

SYMBOL	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	—	—	1.60	—	—	0.063
A1	0.05	—	0.15	0.002	—	0.006
A2	1.35	1.40	1.45	0.053	0.055	0.057
D	16.00 BSC.			0.630 BSC.		
D1	14.00 BSC.			0.551 BSC.		
E	16.00 BSC.			0.630 BSC.		
E1	14.00 BSC.			0.551 BSC.		
R2	0.08	—	0.20	0.003	—	0.008
R1	0.08	—	—	0.003	—	—
θ	0°	3.5°	7°	0°	3.5°	7°
θ1	0°	—	—	0°	—	—
θ2	11°	12°	13°	11°	12°	13°
θ3	11°	12°	13°	11°	12°	13°
c	0.09	—	0.20	0.004	—	0.008
L	0.45	0.60	0.75	0.018	0.024	0.030
L1	1.00 REF			0.039 REF		
S	0.20	—	—	0.008	—	—
b	0.17	0.20	0.27	0.007	0.008	0.011
e	0.50 BSC.			0.020 BSC.		
D2	12.00			0.472		
E2	12.00			0.472		
TOLERANCES OF FORM AND POSITION						
aaa	0.20			0.008		
bbb	0.20			0.008		
ccc	0.08			0.003		
ddd	0.08			0.003		

Table 46-1. 100-lead LQFP Device and Package Maximum Weight

800	mg

Table 46-2. 100-lead LQFP Package Reference

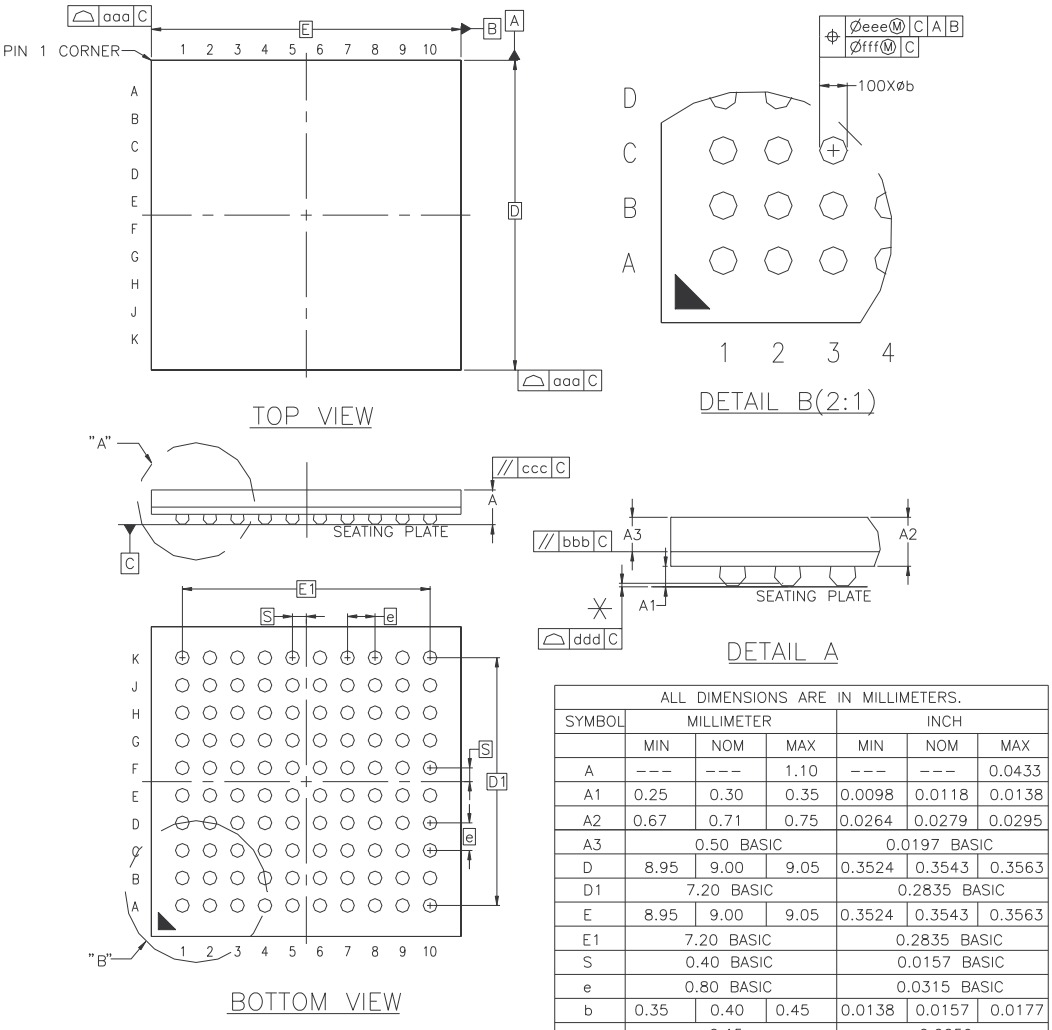
JEDEC Drawing Reference	MS-026
JESD97 Classification	e3

Table 46-3. 100-lead LQFP Package Characteristics

Moisture Sensitivity Level	3

DRAFT

Figure 46-2. 100-ball TFBGA Package Drawing



ALL DIMENSIONS ARE IN MILLIMETERS.						
SYMBOL	MILLIMETER			INCH		
	MIN	NOM	MAX	MIN	NOM	MAX
A	---	---	1.10	---	---	0.0433
A1	0.25	0.30	0.35	0.0098	0.0118	0.0138
A2	0.67	0.71	0.75	0.0264	0.0279	0.0295
A3	0.50 BASIC			0.0197 BASIC		
D	8.95	9.00	9.05	0.3524	0.3543	0.3563
D1	7.20 BASIC			0.2835 BASIC		
E	8.95	9.00	9.05	0.3524	0.3543	0.3563
E1	7.20 BASIC			0.2835 BASIC		
S	0.40 BASIC			0.0157 BASIC		
e	0.80 BASIC			0.0315 BASIC		
b	0.35	0.40	0.45	0.0138	0.0157	0.0177
aaa	0.15			0.0059		
bbb	0.20			0.0079		
ccc	0.20			0.0079		
ddd	0.12			0.0047		
eee	0.15			0.0059		
fff	0.08			0.0031		

Table 46-4. 100-ball TFBGA Package Reference - Soldering Information (Substrate Level)

Ball Land	Diameter 0.45 mm
Soldering Mask Opening	0.35 mm

Table 46-5. 100-ball TFBGA Device and Package Maximum Weight

141	mg

Table 46-6. 100-ball TFBGA Package Characteristics

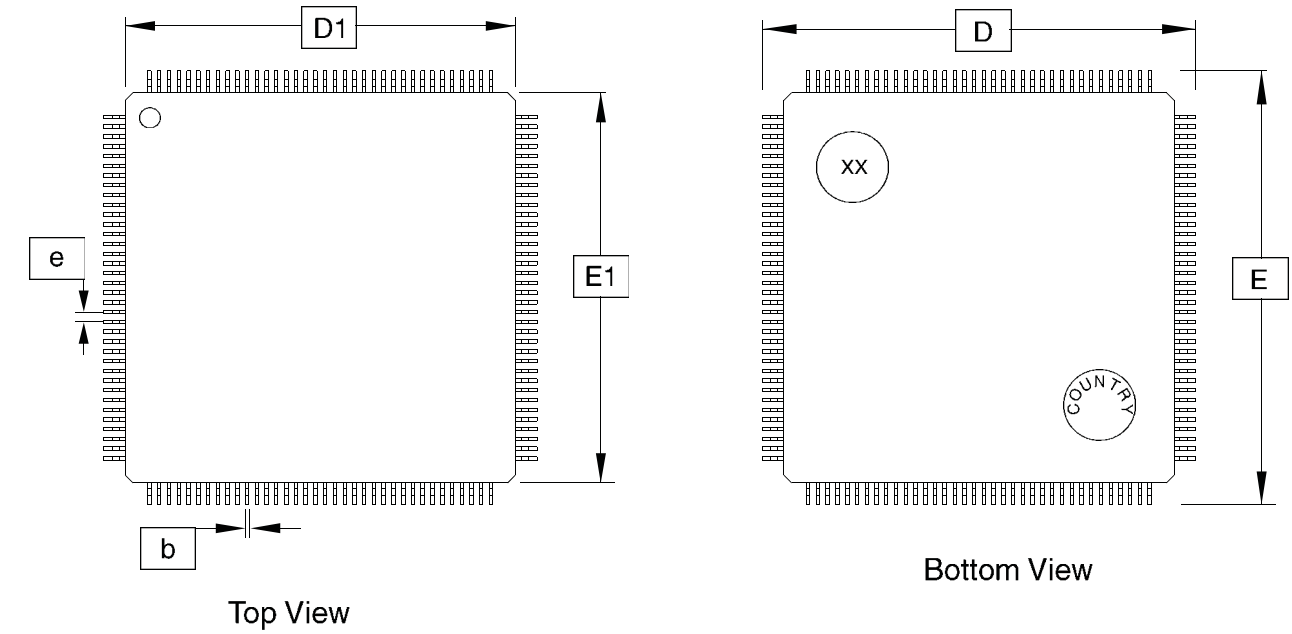
Moisture Sensitivity Level	3

Table 46-7. 100-ball TFBGA Package Reference

JEDEC Drawing Reference	MO-275-DDAC-1
JESD97 Classification	e8

DRAFT

Figure 46-3. 144-lead LQFP Package Drawing



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A1	0.05		0.15	6
A2	1.35	1.40	1.45	
D		22.00 BSC		
D1		20.00 BSC		2, 3
E		22.00 BSC		
E1		20.00 BSC		2, 3
e		0.50 BSC		
b	0.17	0.22	0.27	4, 5
L1		1.00 REF		

Notes:
1. This drawing is for general information only; refer to JEDEC Drawing MS-026 for additional information.
2. The top package body size may be smaller than the bottom package size by as much as 0.15 mm.
3. Dimensions D1 and E1 do not include mold protrusions. Allowable protrusion is 0.25 mm per side. D1 and E1 are maximum plastic body size dimensions including mold mismatch.
4. Dimension b does not include Dambar protrusion. Allowable Dambar protrusion shall not cause the lead width to exceed the maximum b dimension by more than 0.08 mm. Dambar cannot be located on the lower radius or the foot. Minimum space between protrusion and an adjacent lead is 0.07 mm for 0.4 and 0.5 mm pitch packages.
5. These dimensions apply to the flat section of the lead between 0.10 mm and 0.25 mm from the lead tip.
6. A1 is defined as the distance from the seating place to the lowest point on the package body.

Table 46-8. 144-lead LQFP Device and Package Maximum Weight

1400	mg

Table 46-9. 144-lead LQFP Package Characteristics

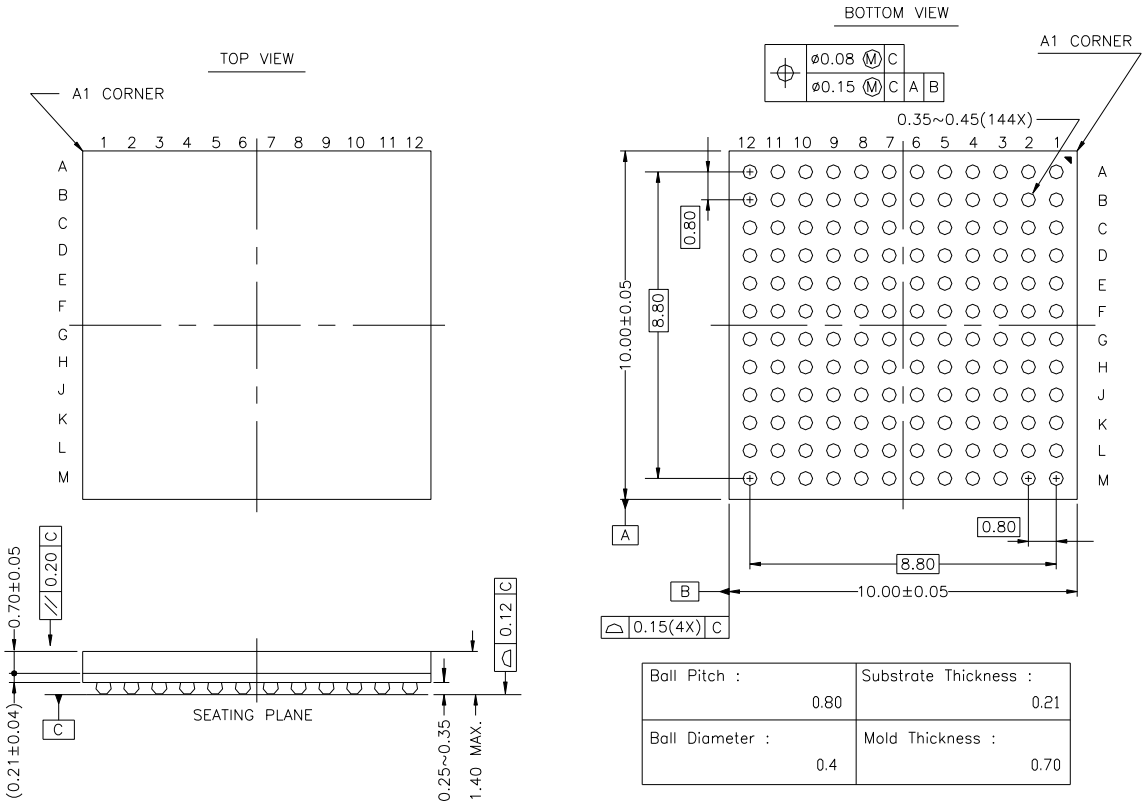
Moisture Sensitivity Level	3

Table 46-10. 144-lead LQFP Package Reference

JEDEC Drawing Reference	MS-026
JESD97 Classification	e3

DRAFT

Figure 46-4. 144-ball LFBGA Package Drawing



All dimensions are in mm

Table 46-11. Soldering Information (Substrate Level)

Ball Land	0.380 mm
Soldering Mask Opening	0.280 mm

Table 46-12. 144-ball BGA Device and Package Maximum Weight

300	mg

Table 46-13. 144-ball BGA Package Characteristics

Moisture Sensitivity Level	3

Table 46-14. Package Reference

JEDEC Drawing Reference	none
JESD97 Classification	e1

DRAFT

46.1 Soldering Profile

Table 46-15 gives the recommended soldering profile from J-STD-020C.

Table 46-15. 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.

46.2 Packaging Resources

Land Pattern Definition.

Refer to the following IPC Standards:

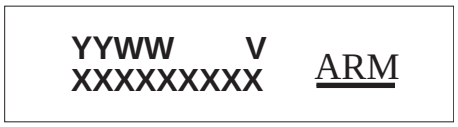
- IPC-7351A and IPC-782 (*Generic Requirements for Surface Mount Design and Land Pattern Standards*) http://landpatterns.ipc.org/default.asp
- Atmel Green and RoHS Policy and Package Material Declaration Data Sheet available on www.atmel.com

DRAFT

47. 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
- “XXXXXXXXX”: lot number

DRAFT

48. Ordering Information

Table 48-1. SAM3X/A Ordering Information

Ordering Code	MRL	Flash (Kbytes)	Package	Operating Temperature Range
ATSAM3A4CA-AU	A	256	LQFP100	Industrial -40°C to 85°C
ATSAM3A8CA-AU	A	512	LQFP100	Industrial -40°C to 85°C
ATSAM3A4CA-CU	A	256	TFBGA100	Industrial -40°C to 85°C
ATSAM3A8CA-CU	A	512	TFBGA100	Industrial -40°C to 85°C
ATSAM3X4CA-AU	A	256	LQFP100	Industrial -40°C to 85°C
ATSAM3X8CA-AU	A	512	LQFP100	Industrial -40°C to 85°C
ATSAM3X4CA-CU	A	256	TFBGA100	Industrial -40°C to 85°C
ATSAM3X8CA-CU	A	512	TFBGA100	Industrial -40°C to 85°C
ATSAM3X4EA-AU	A	256	LQFP144	Industrial -40°C to 85°C
ATSAM3X8EA-AU	A	512	LQFP144	Industrial -40°C to 85°C
ATSAM3X4EA-CU	A	256	LFBGA144	Industrial -40°C to 85°C
ATSAM3X8EA-CU	A	512	LFBGA144	Industrial -40°C to 85°C

DRAFT

49. SAM3X/A Series Errata

49.1 Errata Revision A Parts

The table below lists the chip IDs for revision A parts.

Table 49-1. Revision A Parts Chip IDs

Chip Name	Revision	Chip ID
SAM3X8H[1]	A	0x286E0A60
SAM3X8E	A	0x285E0A60
SAM3X4E	A	0x285B0960
SAM3X8C	A	0x284E0A60
SAM3X4C	A	0x284B0960
SAM3A8C	A	0x283E0A60
SAM3A4C	A	0x283B0960

Note: 1. This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

49.1.1 Flash Memory

49.1.1.1 Flash: Flash Programming

When writing data into the Flash memory plane (either through the EEFC, using the IAP function or FFPI), the data may not be correctly written (i.e., the data written is not the one expected).

Problem Fix/Workaround

Set the number of Wait States (WS) at 6 (FWS = 6) during the programming.

49.1.1.2 Flash: Fetching Error after Reading the Unique Identifier

After reading the Unique Identifier (or using the STUI/SPUI command), the processor may fetch wrong instructions. It depends on the code and on the region of the code.

Problem Fix/Workaround

In order to avoid this problem, follow the steps below:

1. Set bit 16 of EEFC Flash Mode Register to 1
2. Send the Start Read Unique Identifier command (STUI) by writing the Flash Command
3. Register with the STUI command.
4. Wait for the FRDY bit to fall
5. Read the Unique ID (and next bits if required)
6. Send the Stop Read Unique Identifier command (SPUI) by writing the Flash Command
7. Register with the SPUI command.
8. Wait for the FRDY bit to rise
9. Clear bit 16 of EEFC Flash Mode Register

Note: During the sequence, the software cannot run out of Flash (so needs to run out of SRAM).

49.1.1.3 Flash: Boot Flash Programming Mapping Is Wrong

GPNVM2 enables to select if Flash0 or Flash1 is used for the boot. But when GPNVM2 is set, only the first 64 Kbytes of the Flash 1 are seen in the Boot Memory. The rest of the Boot memory corresponds to Flash 0 content.

Problem Fix/Workaround

No fix required if the code size is less than 64Kbytes. Otherwise the software needs to take into account this limitation if GPNVM2 is used.

49.1.1.4 Flash: Flash Programming

When writing data into the Flash memory plane (either through the EEFC, using the IAP function or FFPI), the data may not be correctly written (i.e the data written is not the one expected).

Problem Fix/Workaround

Set the number of Wait States (WS) at 6 (FWS = 6) during the programming.

49.1.2 Backup Mode

49.1.2.1 Backup mode: The PIO States Are Not Kept

When entering in Backup mode, the PIO states are not kept. All the PIOs go into input with pull-up state in Backup mode.

Problem Fix/Workaround

The issue is of course only present when VDDIO is still supplied in Backup mode. If the VDDIO is required in this low power mode, either adapt the hardware so that the input with pull-up state is taken into account in the schematics or use Wait mode instead.

49.1.2.2 Backup Mode: VDDIO/VDDANA

There is an overcurrent consumption in Backup mode on VDDIO and/or VDDANA when VDDIO and/or VDDANA are supplied.

Problem Fix/Workaround

Two workarounds are possible:

1) Do not supply VDDANA and VDDIO in backup (by using a switch controlled by SHDN pin for example)

2) Use Wait mode instead of Backup Mode.

49.1.2.3 Backup Mode Power-Up Sequence

When waking up from Backup Mode with all VDDIN, VDDIO, VDDANA, VDDUTMI power supplies rising at the same time, there is an over power consumption on VDDIO.

Problem Fix/ Workaround

The user has to make sure VDDCORE has reached 1.6V before VDDIO starts rising. This might be implemented with the following control circuitry.

If the internal voltage regulator is used, VDDIN raising 1 ms before VDDIO allows meeting this requirement.

49.1.3 Pulse Width Modulation (PWM)

49.1.3.1 PWM: Write Protection with PIO Lock Feature Not Usable

Write protection mode with PIO lock (WPCMD = 2) can not be used. In this case, the PWM outputs will not be locked but other PIOs will be locked instead

Problem Fix/Workaround

Use Write protect mode without locking the PIO (WPCMD = 1)

49.1.4 Analog to Digital Converter (ADC)

49.1.4.1 ADC: First Conversion After Sleep

In sleep mode, first converted data incorrect after Wakeup.

Problem Fix/Workaround

One or more channels must be converted using sleep mode, a dummy channel as first channel of the series can be used.

49.1.4.2 ADC: Last Conversion Error

In sleep mode, last converted data incorrect before Shutdown.

Problem Fix/Workaround

One or more channels must be converted using sleep mode, a dummy channel as first channel of the series can be used.

49.1.4.3 ADC: Wrong First Conversions

The first conversions done by the ADC may be erroneous if the maximum gain (x4 in single ended or x2 in differential mode) is not used. The issue appears after the power-up or if a conversion has not been occurred for 1 minute.

Problem Fix/Workaround

Three workarounds are possible:

1) Perform 16 dummy conversions on one channel (whatever conditions used in term of setup of gain, single/differential, offset, and channel selected). The next conversions will be correct for any channels and any settings. Note that these dummy conversions need to be performed if no conversion has occurred for 1 minute or for a new chip startup.

2) Perform a dummy conversion on a single ended channel on which an external voltage of ADVREF/2 (+/-10%) is applied. And use the following conditions for this conversion: gain at 4, offset set at 1. The next conversions will be correct for any channels and any settings. Note that this dummy conversion needs to be performed if no conversion has occurred for 1 minute or for a new chip startup.

3) Perform a dummy conversion on a differential channel on which the two inputs are connected together and connected to any voltage (from 0 to ADVREF). And use the following conditions for this conversion: gain at 4, offset set at 1. The next conversions will be correct for any channels and any settings. Note that this dummy conversion needs to be performed if no conversion has occurred for 1 minute or for a new chip startup.

49.1.5 JTAG Boundary

49.1.5.1 JJTAG Boundary: PC0/ERASE

In JTAG Boundary Scan mode, the PIO PC0 which is by default the ERASE pin may erase the Flash content if the pin is set to high level.

Problem Fix/Workaround

Do not use PC0/ERASE (or tie to 0) or set PC0/ERASE at high level during a time lower than 150 ms in Boundary JTAG Scan mode or program the Flash after using this mode.

50. Revision History

In the table that follows, the most recent version of the document appears first.

Table 50-1. SAM3X/SAM3A Datasheet Rev. 11057C Revision History

Issue Date	Comments
23-Mar-15	Reformatted document; editorial changes throughout; renamed package 100-ball LFBGA to 100-ball TFBGA throughout
	"Description" Changed "9x general-purpose 32-bit timers, an RTC" to "three 3-channel general-purpose 32-bit timers, a low-power RTC, a low-power RTT, 256-bit General Purpose Backup Registers" Added paragraph relating to low-power modes
	Section 1. "Features" Updated description of "Low-power Modes" Under "Peripherals": - changed "32-bit Real Time Timer (RTT) and RTC" to "32-bit low-power Real-time Timer (RTT) and low-power Real-time Clock (RTC)" - added bullet "256-bit General Purpose Backup Registers (GPBR)" Table 1-1 "Configuration Summary": added footnote "Available only on SAM3X8H in LFBGA217 package (not commercially available)" to SDRAM Controller
	Section 2. "SAM3X/A Block Diagram": in all block diagrams (Figure 2-1 through Figure 2-4), - renumbered three Timer Counter blocks as 0-1-2 (were previously A-B-C) - corrected JTAG/SWD pin name "TDO" to "TDO/TRACESWO" - updated SPI signal names Figure 2-1 "SAM3A4/8C (100 pins) Block Diagram": removed duplicated DMA cells from blocks SPI0, SSC, and HSMCI
	Section 3. "Signal Description" Table 3-1 "Signal Description List": updated SPI signal names Section 3.1 "Design Considerations": updated to provide title and number of referenced application note
	Section 4. "Package and Pinout" Table 4-2 "100-ball TFBGA SAM3X4/8E Package and Pinout": renamed 'WAKEUP' to 'FWUP' for ball H5
	Section 5. "Power Considerations" Added Section 5.2 "Power-up Considerations" Figure 5-2 "Single Supply": corrected main supply range "1.8V-3.6V" to "1.62–3.6 V" Figure 5-4 "Backup Batteries Used": corrected main supply range "1.8V-3.6V" to "1.62–3.6 V" Updated Section 5.7 "Wake-up Sources" Updated Section 5.8 "Fast Startup"
	Section 6. "Input/Output Lines" Table 6-1 "System I/O Configuration Pin List": changed header in first column from "SYSTEM_IO Bit Number" to "CCFG_SYSIO Bit No." Section 6.6 "ERASE Pin": in first paragraph, added details relative to reprogramming Flash content; in last sentence, changed "asserting the pin to low does not erase the Flash" to "asserting the pin to high does not erase the Flash"
	Removed Section 7. "Processor and Architecture"

Table 50-1. SAM3X/SAM3A Datasheet Rev. 11057C Revision History (Continued)

Issue Date	Comments
23-Mar-15	Section 7. “Memories” Inserted Section 7.1 “Product Mapping” (was previously section 8. “Product Mapping”) Updated Section 7.2.1 “Internal SRAM” Section 7.2.2 “Internal ROM”: changed “At any time, the ROM is mapped at address 0x0018 0000” to “At any time, the ROM is mapped at address 0x0010 0000” Updated Section 7.2.3.10 “GPNVM Bits” Section 7.2.4 “Boot Strategies”: deleted sentence “The GPNVM bit can be cleared or set respectively through the “Clear General-purpose NVM Bit” and “Set General-purpose NVM Bit” commands of the EEFC User Interface” (this information is already provided in Section 7.2.3.10 “GPNVM Bits”)
	Section 8. “System Controller” Removed Figure 10-1. “System Controller Block Diagram” (redundant with Figure 16-1 “Supply Controller Block Diagram”)
	Section 9. “Peripherals” Table 9-2 ”Multiplexing on PIO Controller A (PIOA)”: added footnotes providing information on selecting extra functions Table 9-3 ”Multiplexing on PIO Controller B (PIOB)”: deleted “See the notes” from the ‘Comments’ column for rows PB0–PB9; added footnotes providing information on selecting extra functions Table 9-4 ”Multiplexing on PIO Controller C (PIOC)”: added footnote providing information on selecting ERASE as extra function
	Section 11. “Debug and Test Features” Section 11.5.2 “Debug Architecture”: corrected “Cortex-M3 embeds four functional units” to “Cortex-M3 embeds five functional units”
	Section 20. “SAM3X/A Boot Program” Section 20.4.3 “USB Device Port”: - in first paragraph, replaced “from Windows 98SE to Windows XP” with “beginning with Windows 98SE” - updated link to www.usb.org - deleted sentence “Unauthorized use of assigned or unassigned USB Vendor ID Numbers and associated Product ID Numbers is strictly prohibited” Section 20.4.4 “In Application Programming (IAP) Feature”: replaced two instances of “MC_FSR register” with “EEFC_FSR”; changed “by reading the NMI vector in ROM (0x00800008)” to “by reading the NMI vector in ROM (0x00100008)”
	Section 23. “External Memory Bus” Table 23-3 ”EBI Pins and External Static Device Connections”: edited and reorganized column headers Section 23.8 “Implementation Examples”: reorganized contents
	Section 36. “Timer Counter (TC)” Throughout, replaced instances of “Master clock” or “MCK” d with “peripheral clock” Throughout, replaced instances of ‘quadrature decoder logic’ with ‘quadrature decoder’ or ‘QDEC’ Section 36.1 “Description”: rewrote text; in second paragraph, TIOA1 replaced with TIOB1; removed Table 37-1. “Timer Counter Clock Assignment” Section 36.2 “Embedded Characteristics”: - corrected number of TC channels from “Three” to “9” - changed “Two Multi-purpose Input/Output Signals” to “Two multi-purpose input/output signals acting as trigger event” - deleted “Two global registers that act on all TC channels” - changed “Configuration Registers can be write protected” to “Register Write Protection” Section 36.3 “Block Diagram”: inserted Table 36-1 ”Timer Counter Clock Assignment” (was previously in Section 36.1 “Description”) and updated table footnote

Table 50-1. SAM3X/SAM3A Datasheet Rev. 11057C Revision History (Continued)

Issue Date	Comments
23-Mar-15	Section 36. “Timer Counter (TC)” (cont’d) Table 36-2 ”Signal Name Description”: updated descriptions of signals ‘INT’ and ‘SYNC’ Table 36-3 ”TC Pin List”: removed ‘FAULT’ pin Table 36-4 ”I/O Lines”: added TIOA3/PB0, TIOA4/PB2, TIOA5/PB4, TIOB3/PB1, TIOB4/PB3, and TIOB5/PB5 Section 36.5.3 “Interrupt Sources”: changed title (was “Interrupt”) and added Table 36-5 ”Peripheral IDs” Section 36.6.2 “32-bit Counter”: revised description Section 36.6.3 “Clock Selection”: renamed internal clock signals to “MCK/2, MCK/8, MCK/32, MCK/128, SLCK” Section 36.6.11.1 “WAVSEL = 00”: replaced “0xFFFF” with “2^{32}-1” in first paragraph Figure 36-9 “WAVSEL = 10 without Trigger” and Figure 36-10 “WAVSEL = 10 with Trigger”: replaced “0xFFFF” with “2^{n}-1” (with “n” representing counter size) Section 36.6.11.3 “WAVSEL = 01”: replaced “0xFFFF” with “2^{32}-1” in first paragraph Figure 36-13 “WAVSEL = 11 without Trigger” and Figure 36-14 “WAVSEL = 11 with Trigger”: replaced “0xFFFF” with “2^{n}-1” (with “n” representing counter size) Section 36.6.14.1 “Description”: in first paragraph, corrected instance of TIOA1 to TIOB1; corrected link to Figure 36-15 Section 36.6.14.2 “Input Pre-processing”: deleted sentence “Filters can be disabled using the FILTER bit in the TC_BMR” Figure 36-16 “Input Stage”: replaced “FILTER” with “MAXFILTER > 0” Section 36.6.14.3 “Direction Status and Change Detection”: rewrote sixth paragraph for clarity Section 36.6.14.4 “Position and Rotation Measurement”: rewrote first paragraph for clarity; corrected instance of TIOA1 to TIOB1; at end of second paragraph, defined External Trigger Edge and External Trigger configuration in TC_CMR Section 36.6.14.5 “Speed Measurement”: replaced “EDGTRG can be set to 0x01” with “ETRGEDG must be set to 0x01”; replaced “The speed can be read on TC_RA0 register in TC_CMR0” with “The speed can be read on field RA in register TC_RA0”; Section 36.6.17 “Register Write Protection”: changed title (was “Write Protection System”) and revised content Table 36-6 ”Register Mapping”: in last row, defined offset range 0xE8–0xFC as reserved Section 36.7.2 “TC Channel Mode Register: Capture Mode”: in ‘Name’ line, replaced “(WAVE = 0)” with “(CAPTURE_MODE)”; updated TCCLKS field description Section 36.7.3 “TC Channel Mode Register: Waveform Mode”: in ‘Name’ line, replaced “(WAVE = 1)” with “(WAVEFORM_MODE)”; updated TCCLKS field description; added a note for ENETRG description Section 36.7.9 “TC Status Register”: added “(cleared on read)” to bit descriptions Section 36.7.14 “TC Block Mode Register”: removed FILTER bit (register bit 19 now reserved); corrected TC2XC2S field configuration values: value 2 is TIOA0 (was TIOA1); value 3 is TIOA1 (was TIOA2) Section 36.7.20 “TC Write Protection Mode Register”: updated descriptions of WPEN bit and WPKEY field
	Section 39. “USB On-The-Go Interface (UOTGHS)” Changed index from [x=1..7] to [x=1..6] in “Name” line of following register descriptions: - Section 39.6.2.17 “Device DMA Channel x Next Descriptor Address Register” - Section 39.6.2.18 “Device DMA Channel x Address Register” - Section 39.6.2.19 “Device DMA Channel x Control Register” - Section 39.6.2.20 “Device DMA Channel x Status Register” - Section 39.6.3.22 “Host DMA Channel x Next Descriptor Address Register” - Section 39.6.3.23 “Host DMA Channel x Address Register” - Section 39.6.3.24 “Host DMA Channel x Control Register” - Section 39.6.3.25 “Host DMA Channel x Status Register”

Table 50-1. SAM3X/SAM3A Datasheet Rev. 11057C Revision History (Continued)

Issue Date	Comments
23-Mar-15	Section 45. “Electrical Characteristics” Updated and harmonized parameter symbols Table 45-1 Absolute Maximum Ratings*: - added “(except VBUS)” to description “Voltage on Input Pins with Respect to Ground” - added “Voltage on VBUS Pin with Respect to Ground” Table 45-2 DC Characteristics: removed parameter “Input Capacitance” Table 45-3 1.8V Voltage Regulator Characteristics: updated condtions for CD_{IN} and CD_{OUT} Table 45-4 Core Power Supply Brownout Detector Characteristics: added parameter “Reset Period” Added Table 45-6 ”Threshold Selection” Table 45-7 Backup Power Supply Zero-Power-on Reset Characteristics: modified parameter name “Reset Time-out Period” to “Reset Period” Section 45.3.1 “Backup Mode Current Consumption”: deleted instances of “Configuration A” Section 45.4.3 “32.768 kHz Crystal Oscillator Characteristics”: added parameter “Allowed Crystal Capacitance Load” Table 45-17 ”32.768 kHz Crystal Oscillator Characteristics”: updated conditions for parameters “Startup Time” and “Current Consumption” Table 45-19 ”XIN32 Clock Electrical Characteristics (In Bypass Mode)”: in table footnote, corrected “32768 kHz” to “32.768 kHz” Updated Figure 45-13 “XIN32 Clock Timing” Table 45-20 ”3 to 20 MHz Crystal Oscillator Characteristics”: added parameter “Allowed Crystal Capacitance Load”; deleted all footnotes Updated Figure 45-15 “XIN Clock Timing” Section 45.4.9.1 “Choosing a Crystal”: under ‘Load Capacitance’, changed “This is the equivalent capacitor” to “$C_{crystal}$ is the equivalent capacitor” Section 45.4.9.2 “Printed Circuit Board (PCB)”: deleted sentence “A board design example is given on Atmel’s website in the MCU Technical Center Section.” Table 45-24 ”PLLA Characteristics”: - updated Input Frequency values (was 8 MHz min/16 MHz max; is 1 MHz min/40 MHz max) - updated Output Frequency values (was 96 MHz min/192 MHz max; is 80 MHz min/240 MHz max) Table 45-25 ”UPLL Characteristics for USB High Speed Device Port”: added max value for Current Consumption Table 45-30 ”Channel Conversion Time and ADC Clock”: changed max value of ADC Clock Frequency from 20 MHz to 22 MHz Section 45.6 “USB On-the-Go High Speed Port”: removed section 46.6.5 “USB High Speed Design Guidelines” Added Section 45.6.2.2 “VBUS Pin Characteristics” Section 45.7.1 “Static Performance Characteristics”: updated conditions at beginning of section Inserted new heading Section 45.7.2 “Dynamic Performance Characteristics” and updated content Figure 45-18 “Simplified Acquisition Path”: inserted correct diagram (was previously graph of offset and gain definitions) Section 45.8 “Temperature Sensor”: specified instances of “27°C” as ambient temperature (T_A) Added Table 45-44 ”Analog Outputs” Figure 45-21 “SPI Slave Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)”: added ‘MOSI’ Section 45.10.3.1 “Maximum SPI Frequency”: updated description under “Master Write Mode” and “Master Read Mode” Table 45-49 ”SSC Timings”: deleted footnote “Timings SSC4 and SSC7 depend...added to timings” Section 45.10.6 “SMC Timings”: updated conditions

Table 50-1. SAM3X/SAM3A Datasheet Rev. 11057C Revision History (Continued)

Issue Date	Comments
23-Mar-15	Section 45. “Electrical Characteristics” (cont’d) Section 45.10.7 “USART in SPI Mode Timings”: updated conditions Figure 45-39 “EMAC RMII Timings”: inserted RMII-specific diagram (was previously extract of “EMAC MII Mode” diagram) Table 45-61 ”Two-wire Serial Bus Requirements”: in bottom row, replaced duplicated parameter “Hold Time (repeated) START Condition” with new parameter ““Bus free time between a STOP and START condition” Section 45.10.10 “Embedded Flash Characteristics”: corrected “field FWS of the MC_FMR register” to “field FWS in the EEFC_FMR”; updated text and replaced two wait state tables with single Table 45-62 ”Embedded Flash Wait State - VDDCORE 1.62V/1.80V”
	Section 46. “Mechanical Characteristics” Figure 46-2 “100-ball TFBGA Package Drawing”: corrected ‘A’ maximum dimension in inches from 0.0575 to 0.0433 Updated Table 46-4 ”100-ball TFBGA Package Reference - Soldering Information (Substrate Level)” Updated Table 46-5 ”100-ball TFBGA Device and Package Maximum Weight” Updated Table 46-6 ”100-ball TFBGA Package Characteristics” Updated Table 46-7 ”100-ball TFBGA Package Reference” Updated Table 46-8 ”144-lead LQFP Device and Package Maximum Weight” Added Section 46.1 “Soldering Profile” Added Section 46.2 “Packaging Resources”
	Inserted Section 47. “Marking” (was previously subsection of Section 49. “SAM3X/A Series Errata”)
	Section 48. “Ordering Information” Section 48-1 “SAM3X/A Ordering Information”: removed “Package Type” column (this information is provided on the Atmel website); renamed “LFBGA100” to “TFBGA100”
	Section 49. “SAM3X/A Series Errata” Added Section 49.1.2.3 “Backup Mode Power-Up Sequence”

Doc. Rev. 11057B	Comments	Change Request Ref.
	SDRAM and 217-pin package information removed only from Section "Features", Section 1.1 "Configuration Summary" (introduction) and Section 48. "Ordering Information".	8251
	"Write protected Registers" added to Section "Features".	8213
	Section "Features", Section 1.1 "Configuration Summary" (introduction) updated. Figure 2-3 "SAM3X4/8E (144 pins) Block Diagram" updated. Section 1.1 "Configuration Summary", added a note "The SAM3X-EK evaluation kit for the SAM3X and SAM3A series is mounted with a SAM3X8H in a LFBGA217 package. This device is not commercially available." Section 46. "Mechanical Characteristics", deleted Figure 46-5 "217-ball LFBGA Package Drawing" and the corresponding tables (46-15, 46-16, 46-17, and 46-18). Section 48. "Ordering Information" updated, ATSAM3X8H-CU information and the note deleted in Table 48-1.	8316
	Added a note explaining that SAM3X8H is not commercially available, and replaced "217-pin" / "217-pin SAM3X" by "217-pin SAM3X8H" in Figure 2-4, Section 5. "Power Considerations", Section 7.2.1 "Internal SRAM", Section 7.3 "External Memories", Section 7.3.5 "SDR-SDRAM Controller (217-pin SAM3X8H(I) only)", Section 9.3 "Peripheral Signal Multiplexing on I/O Lines", Section 22.2 "Embedded Characteristics" (DMAC), Section 23.2 "Embedded Characteristics" (External Memory Bus), Section 26.2 "Embedded Characteristics" (PDC), Section 29.2 "Embedded Characteristics" (CHIPID), Section 31.2 "Embedded Characteristics" (PIO), and Section 49.1 "Errata Revision A Parts". Section 7. "Memories" added to the Overview.	8318
	Section 1.1 "Configuration Summary", removed SAM3X8H information from Table 1-1 on page 3 and added a note explaining that SAM3X8H is not commercially available. Removed entirely Section 4.3 SAM3X8H Package and Pinout. Updated new notes style: changed the color from red to black.	rfo[1]
	Fixed incorrectly displayed figures (due to Adobe Illustrator to PDF conversion problem): Figure 5-5 "Wake-up Source", Figure 25-36 "NAND Flash Controller Timing Engine", Figure 25-38 "Program Page Flow Chart", Figure 27-2 "Typical Slow Clock Crystal Oscillator Connection", Figure 27-3 "Main Clock Block Diagram", Figure 28-1 "General Clock Block Diagram", Figure 28-3 "Master Clock Controller", Figure 36-1 "Timer Counter Block Diagram", and Figure 38-3 "Functional View of the Channel Block Diagram".	rfo[1]

Note: 1. "rfo" indicates changes requested during document review and approval loop.

Doc. Rev. 11057A	Comments	Change Request Ref.
	First Issue	

Table of Contents

Description . . . 1

1. Features . . . 2
1.1 Configuration Summary . . . 3

2. SAM3X/A Block Diagram . . . 4

3. Signal Description . . . 8
3.1 Design Considerations . . . 13

4. Package and Pinout . . . 14
4.1 SAM3A4/8C and SAM3X4/8C Package and Pinout . . . 14
4.2 SAM3X4/8E Package and Pinout . . . 17

5. Power Considerations . . . 20
5.1 Power Supplies . . . 20
5.2 Power-up Considerations . . . 20
5.3 Voltage Regulator . . . 21
5.4 Typical Powering Schematics . . . 22
5.5 Active Mode . . . 25
5.6 Low Power Modes . . . 25
5.7 Wake-up Sources . . . 28
5.8 Fast Startup . . . 28

6. Input/Output Lines . . . 28
6.1 General Purpose I/O Lines (GPIO) . . . 28
6.2 System I/O Lines . . . 29
6.3 Test Pin . . . 29
6.4 NRST Pin . . . 30
6.5 NRSTB Pin . . . 30
6.6 ERASE Pin . . . 30

7. Memories . . . 31
7.1 Product Mapping . . . 31
7.2 Embedded Memories . . . 32
7.3 External Memories . . . 35

8. System Controller . . . 37
8.1 System Controller and Peripherals Mapping . . . 37
8.2 Power-on-Reset, Brownout and Supply Monitor . . . 37

9. Peripherals . . . 38
9.1 Peripheral Identifiers . . . 38
9.2 APB/AHB Bridge . . . 39
9.3 Peripheral Signal Multiplexing on I/O Lines . . . 39

10. ARM Cortex® M3 Processor . . . 47
10.1 About this section . . . 47
10.2 Embedded Characteristics . . . 47
10.3 About the Cortex-M3 processor and core peripherals . . . 47
10.4 Programmers model . . . 50

10.5 Memory model . . . 62
10.6 Exception model . . . 70
10.7 Fault handling . . . 78
10.8 Power management . . . 80
10.9 Instruction set summary . . . 81
10.10 Intrinsic functions . . . 85
10.11 About the instruction descriptions . . . 86
10.12 Memory access instructions . . . 94
10.13 General data processing instructions . . . 109
10.14 Multiply and divide instructions . . . 123
10.15 Saturating instructions . . . 127
10.16 Bitfield instructions . . . 129
10.17 Branch and control instructions . . . 133
10.18 Miscellaneous instructions . . . 141
10.19 About the Cortex-M3 peripherals . . . 152
10.20 Nested Vectored Interrupt Controller . . . 152
10.21 System control block . . . 165
10.22 System timer, SysTick . . . 192
10.23 Memory protection unit . . . 197
10.24 Glossary . . . 210

11. Debug and Test Features . . . 214
11.1 Description . . . 214
11.2 Embedded Characteristics . . . 214
11.3 Application Examples . . . 215
11.4 Debug and Test Pin Description . . . 216
11.5 Functional Description . . . 217

12. Reset Controller (RSTC) . . . 222
12.1 Description . . . 222
12.2 Embedded Characteristics . . . 222
12.3 Block Diagram . . . 222
12.4 Functional Description . . . 223
12.5 Reset Controller (RSTC) User Interface . . . 230

13. Real-time Timer (RTT) . . . 234
13.1 Description . . . 234
13.2 Embedded Characteristics . . . 234
13.3 Block Diagram . . . 234
13.4 Functional Description . . . 235
13.5 Real-time Timer (RTT) User Interface . . . 237

14. Real-time Clock (RTC) . . . 242
14.1 Description . . . 242
14.2 Embedded Characteristics . . . 242
14.3 Block Diagram . . . 242
14.4 Product Dependencies . . . 242
14.5 Functional Description . . . 243
14.6 Real-time Clock (RTC) User Interface . . . 246

15. Watchdog Timer (WDT) . . . 260
15.1 Description . . . 260

15.2 Embedded Characteristics . . . 260
15.3 Block Diagram . . . 260
15.4 Functional Description . . . 261
15.5 Watchdog Timer (WDT) User Interface . . . 263

16. Supply Controller (SUPC) . . . 268
16.1 Description . . . 268
16.2 Embedded Characteristics . . . 268
16.3 Block Diagram . . . 269
16.4 Supply Controller Functional Description . . . 270
16.5 Supply Controller (SUPC) User Interface . . . 279

17. General Purpose Backup Registers (GPBR) . . . 289
17.1 Description . . . 289
17.2 Embedded Characteristics . . . 289
17.3 General Purpose Backup Registers (GPBR) User Interface . . . 290

18. Enhanced Embedded Flash Controller (EEFC) . . . 292
18.1 Description . . . 292
18.2 Embedded Characteristics . . . 292
18.3 Product Dependencies . . . 292
18.4 Functional Description . . . 293
18.5 Enhanced Embedded Flash Controller (EEFC) User Interface . . . 303

19. Fast Flash Programming Interface (FFPI) . . . 308
19.1 Description . . . 308
19.2 Parallel Fast Flash Programming . . . 308

20. SAM3X/A Boot Program . . . 318
20.1 Description . . . 318
20.2 Flow Diagram . . . 318
20.3 Device Initialization . . . 318
20.4 SAM-BA Monitor . . . 319
20.5 Hardware and Software Constraints . . . 322

21. Bus Matrix (MATRIX) . . . 323
21.1 Description . . . 323
21.2 Embedded Characteristics . . . 323
21.3 Memory Mapping . . . 324
21.4 Special Bus Granting Techniques . . . 324
21.5 Arbitration . . . 325
21.6 System I/O Configuration . . . 327
21.7 Write Protect Registers . . . 327
21.8 Bus Matrix (MATRIX) User Interface . . . 328

22. AHB DMA Controller (DMAC) . . . 338
22.1 Description . . . 338
22.2 Embedded Characteristics . . . 338
22.3 Block Diagram . . . 340
22.4 Functional Description . . . 341
22.5 DMAC Software Requirements . . . 356
22.6 Write Protection Registers . . . 357

22.7 AHB DMA Controller (DMAC) User Interface 358

23. External Memory Bus 382
23.1 Description 382
23.2 Embedded Characteristics 382
23.3 Block Diagram 383
23.4 I/O Lines Description 384
23.5 Application Example 385
23.6 Product Dependencies 386
23.7 Functional Description 387
23.8 Implementation Examples 388

24. AHB SDRAM Controller (SDRAMC) 392
24.1 Description 392
24.2 Embedded Characteristics 392
24.3 I/O Lines Description 393
24.4 Application Example 394
24.5 Product Dependencies 395
24.6 Functional Description 398
24.7 AHB SDRAM Controller (SDRAMC) User Interface 405

25. Static Memory Controller (SMC) 419
25.1 Description 419
25.2 Embedded Characteristics 420
25.3 Block Diagram 421
25.4 I/O Lines Description 421
25.5 Multiplexed Signals 422
25.6 Application Example 422
25.7 Product Dependencies 423
25.8 External Memory Mapping 423
25.9 Connection to External Devices 425
25.10 Standard Read and Write Protocols 427
25.11 Scrambling/Unscrambling Function 434
25.12 Automatic Wait States 435
25.13 Data Float Wait States 438
25.14 External Wait 442
25.15 Slow Clock Mode 448
25.16 NAND Flash Controller Operations 450
25.17 SMC Error Correcting Code Functional Description 461
25.18 Static Memory Controller (SMC) User Interface 465

26. Peripheral DMA Controller (PDC) 503
26.1 Description 503
26.2 Embedded Characteristics 503
26.3 Block Diagram 505
26.4 Functional Description 506
26.5 Peripheral DMA Controller (PDC) User Interface 508

27. Clock Generator 519
27.1 Description 519
27.2 Embedded Characteristics 519
27.3 Block Diagram 520

27.4 Slow Clock . . . 520
27.5 Main Clock . . . 522
27.6 Divider and PLL Block . . . 524
27.7 UTMI Phase Lock Loop Programming . . . 525

28. Power Management Controller (PMC) . . . 526
28.1 Description . . . 526
28.2 Embedded Characteristics . . . 526
28.3 Block Diagram . . . 527
28.4 Master Clock Controller . . . 528
28.5 Processor Clock Controller . . . 528
28.6 SysTick Clock . . . 528
28.7 Peripheral Clock Controller . . . 528
28.8 Free Running Processor Clock . . . 529
28.9 Programmable Clock Output Controller . . . 529
28.10 Fast Startup . . . 530
28.11 Main Crystal Clock Failure Detector . . . 531
28.12 Programming Sequence . . . 531
28.13 Clock Switching Details . . . 534
28.14 Write Protection Registers . . . 537
28.15 Power Management Controller (PMC) User Interface . . . 538

29. Chip Identifier (CHIPID) . . . 567
29.1 Description . . . 567
29.2 Embedded Characteristics . . . 567
29.3 Chip Identifier (CHIPID) User Interface . . . 568

30. Synchronous Serial Controller (SSC) . . . 574
30.1 Description . . . 574
30.2 **Embedded Characteristics** . . . 574
30.3 Block Diagram . . . 575
30.4 Application Block Diagram . . . 575
30.5 Pin Name List . . . 576
30.6 Product Dependencies . . . 576
30.7 Functional Description . . . 577
30.8 SSC Application Examples . . . 588
30.9 Synchronous Serial Controller (SSC) User Interface . . . 591

31. Parallel Input/Output Controller (PIO) . . . 618
31.1 Description . . . 618
31.2 Embedded Characteristics . . . 618
31.3 Block Diagram . . . 619
31.4 Product Dependencies . . . 620
31.5 Functional Description . . . 621
31.6 I/O Lines Programming Example . . . 629
31.7 Parallel Input/Output Controller (PIO) User Interface . . . 631

32. Serial Peripheral Interface (SPI) . . . 676
32.1 Description . . . 676
32.2 Embedded Characteristics . . . 676
32.3 Block Diagram . . . 677
32.4 Application Block Diagram . . . 677

32.5 Signal Description . . . 678
32.6 Product Dependencies . . . 679
32.7 Functional Description . . . 680
32.8 Serial Peripheral Interface (SPI) User Interface . . . 692

33. Two-wire Interface (TWI) . . . 708
33.1 Description . . . 708
33.2 Embedded Characteristics . . . 708
33.3 List of Abbreviations . . . 709
33.4 Block Diagram . . . 709
33.5 Application Block Diagram . . . 710
33.6 Product Dependencies . . . 710
33.7 Functional Description . . . 712
33.8 Master Mode . . . 713
33.9 Multi-master Mode . . . 725
33.10 Slave Mode . . . 728
33.11 Two-wire Interface (TWI) User Interface . . . 735

34. Universal Asynchronous Receiver Transceiver (UART) . . . 750
34.1 Description . . . 750
34.2 Embedded Characteristics . . . 750
34.3 Block Diagram . . . 751
34.4 Product Dependencies . . . 751
34.5 UART Operations . . . 752
34.6 Universal Asynchronous Receiver Transceiver (UART) User Interface . . . 758

35. Universal Synchronous Asynchronous Receiver Transmitter (USART) . . . 769
35.1 Description . . . 769
35.2 Embedded Characteristics . . . 770
35.3 Block Diagram . . . 771
35.4 Application Block Diagram . . . 772
35.5 I/O Lines Description . . . 772
35.6 Product Dependencies . . . 773
35.7 Functional Description . . . 775
35.8 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface . . . 823

36. Timer Counter (TC) . . . 856
36.1 Description . . . 856
36.2 Embedded Characteristics . . . 856
36.3 Block Diagram . . . 857
36.4 Pin Name List . . . 858
36.5 Product Dependencies . . . 858
36.6 Functional Description . . . 860
36.7 Timer Counter (TC) User Interface . . . 879

37. High Speed MultiMedia Card Interface (HSMCI) . . . 909
37.1 Description . . . 909
37.2 Embedded Characteristics . . . 909
37.3 Block Diagram . . . 910
37.4 Application Block Diagram . . . 911
37.5 Pin Name List . . . 911
37.6 Product Dependencies . . . 912

37.7 Bus Topology . . . 913
37.8 High Speed MultiMediaCard Operations . . . 916
37.9 SD/SDIO Card Operation . . . 936
37.10 CE-ATA Operation . . . 937
37.11 HSMCI Boot Operation Mode . . . 938
37.12 HSMCI Transfer Done Timings . . . 939
37.13 Write Protection Registers . . . 940
37.14 High Speed MultiMedia Card Interface (HSMCI) User Interface . . . 941

38. Pulse Width Modulation (PWM) . . . 970
38.1 Description . . . 970
38.2 Embedded Characteristics . . . 971
38.3 Block Diagram . . . 972
38.4 I/O Lines Description . . . 972
38.5 Product Dependencies . . . 973
38.6 Functional Description . . . 976
38.7 Pulse Width Modulation (PWM) User Interface . . . 1003

39. USB On-The-Go Interface (UOTGHS) . . . 1053
39.1 Description . . . 1053
39.2 Embedded Characteristics . . . 1053
39.3 Block Diagram . . . 1054
39.4 Product Dependencies . . . 1057
39.5 Functional Description . . . 1058
39.6 USB On-The-Go Interface (UOTGHS) User Interface . . . 1086

40. Controller Area Network (CAN) . . . 1185
40.1 Description . . . 1185
40.2 Embedded Characteristics . . . 1185
40.3 Block Diagram . . . 1186
40.4 Application Block Diagram . . . 1186
40.5 I/O Lines Description . . . 1187
40.6 Product Dependencies . . . 1187
40.7 CAN Controller Features . . . 1188
40.8 Functional Description . . . 1199
40.9 Controller Area Network (CAN) User Interface . . . 1211

41. Ethernet MAC 10/100 (EMAC) . . . 1242
41.1 Description . . . 1242
41.2 Embedded Characteristics . . . 1242
41.3 Block Diagram . . . 1243
41.4 Functional Description . . . 1244
41.5 Programming Interface . . . 1254
41.6 Ethernet MAC 10/100 (EMAC) User Interface . . . 1257

42. True Random Number Generator (TRNG) . . . 1309
42.1 Description . . . 1309
42.2 Embedded Characteristics . . . 1309
42.3 True Random Number Generator (TRNG) User Interface . . . 1310

43. Analog-to-Digital Converter (ADC) . . . 1317
43.1 Description . . . 1317

43.2 Embedded Characteristics . . . 1318
43.3 Block Diagram . . . 1318
43.4 Signal Description . . . 1319
43.5 Product Dependencies . . . 1319
43.6 Functional Description . . . 1321
43.7 Analog-to-Digital Converter (ADC) User Interface . . . 1331

44. Digital-to-Analog Converter Controller (DACC) . . . 1355
44.1 Description . . . 1355
44.2 Embedded Characteristics . . . 1355
44.3 Block Diagram . . . 1356
44.4 Signal Description . . . 1356
44.5 Product Dependencies . . . 1357
44.6 Functional Description . . . 1358
44.7 Analog Converter Controller (DACC) User Interface . . . 1362

45. Electrical Characteristics . . . 1378
45.1 Absolute Maximum Ratings . . . 1378
45.2 DC Characteristics . . . 1379
45.3 Power Consumption . . . 1385
45.4 Crystal Oscillators Characteristics . . . 1394
45.5 UPLL, PLLA Characteristics . . . 1400
45.6 USB On-the-Go High Speed Port . . . 1401
45.7 12-bit ADC Characteristics . . . 1403
45.8 Temperature Sensor . . . 1410
45.9 12-bit DAC Characteristics . . . 1411
45.10 AC Characteristics . . . 1413

46. Mechanical Characteristics . . . 1435
46.1 Soldering Profile . . . 1439
46.2 Packaging Resources . . . 1439

47. Marking . . . 1440

48. Ordering Information . . . 1441

49. SAM3X/A Series Errata . . . 1442
49.1 Errata Revision A Parts . . . 1442

50. Revision History . . . 1445

Table of Contents . . . i

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

© 2015 Atmel Corporation. / Rev.: Atmel-11057C-ATARM-SAM3X-SAM3A-Datasheet_23-Mar-15.

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo, and others are the registered trademarks or trademarks of ARM Ltd. Windows® is a registered trademark of Microsoft Corporation in U.S. 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 ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL 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 ATMEL 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 ATMEL 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.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.